8-1-1999

Unconstrained instantaneous center integration (ICI) algorithms

Nathan Eggleton

UNCONSTRAINED INSTANTANEOUS CENTER INTEGRATION (ICI) ALGORITHMS

by

Nathan J. Eggleton

A Thesis Submitted in
Partial Fulfillment of the
Requirement for the

MASTER OF SCIENCE
IN
MECHANICAL ENGINEERING

Approved by:

Dr. Michael P. Hennessey _____
Department of Mechanical Engineering (Thesis Advisor)

Dr. Wayne W. Walter _____
Department of Mechanical Engineering

Dr. Joseph S. Torok _____
Department of Mechanical Engineering

Dr. Charles W. Haines _____
Head of Department of Mechanical Enigineering

DEPARTMENT OF MECHANICAL ENGINEERING
ROCHESTER INSTITUTE OF TECHNOLOGY

AUGUST, 1999

Permission Granted

Title of Thesis: Unconstrained Instantaneous Center Integration (ICI) Algorithms

I, Nathan Eggleton, hereby grant permission to the Wallace Library of Rochester Institute of Technology to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

Date: 8/20/99 Signature of Author: _____

Unconstrained Instantaneous Center Integration (ICI) Algorithms

by

Nathan J. Eggleton
Michael P. Hennessey
Department of Mechanical Engineering
College of Engineering
Rochester Institute of Technology
Rochester, NY 14623-5604
Phone: (716) 594-0842
FAX: (716) 475-7710
Email: nje3835@rit.edu

August 1999

ABSTRACT

This paper presents several unique "instantaneous center integration" (ICI) algorithms for use in studying unconstrained vehicle motion using computer simulations and animations. Specifically, these algorithms are most appropriate for the case when high angular rates are present, such as would occur during tight cornering and/or possibly during an accident. The three ICI algorithms presented are the "pseudo-exact", first order approximation and the second order approximation. These ICI algorithms are capable of producing valid, accurate, and efficient simulation results, which describe the behavior of a vehicle under these conditions. The "pseudo-exact" is the most accurate, yet the most computationally expensive. The first order approximation is the least accurate and least computationally expensive. The second order approximation's accuracy is close to that of the "pseudo-exact" algorithm yet is much less computationally accurate, and therefore the most appropriate one to use. The ICI algorithms are derived and compared to one another by observing the position errors, the computational intensities, and the actual trajectories for a baseline two-dimensional scenario. This scenario consists of a vehicle traversing a circular path at a constant speed. A practical study of the stability of the algorithm is presented as well. The ICI algorithms introduced in this paper can be useful for producing accurate computer simulations for both the simulation and entertainment fields.

NOMENCLATURE

Parameters

k:	Number of data points taken per revolution of the circular trajectory of the vehicle, [unitless]
Δt:	Time increment, [sec]

Variables

ε:	Percent error of the approximation algorithms with respect to the exact solution [%]
ϕ:	Angle that the vehicle has rolled about **V** for the time increment, [rad]
θ:	Angle that the vehicle has traveled through for the time increment, [rad]
R:	Radius of curvature of the vehicle's path, [m]
t:	Time, [sec]
$\hat{\mathbf{u}}_x$:	Heading unit vector of the vehicle, [unitless]
$\hat{\mathbf{u}}_y$:	Unit vector perpendicular to the heading unit vector, [unitless]
$\mathbf{V}$:	Longitudinal vehicle velocity, measured at the center of gravity [m/sec]
V_x:	Vehicle's speed in the direction of the heading vector (vehicle coordinate frame's x-axis), [m/sec]
V_y:	Vehicle's speed in the direction perpendicular to the heading vector vehicle coordinate frame's y-axis), [m/sec]
$\mathbf{V}^{\perp}$:	Vector perpendicular to the velocity vector (heading toward (X_c, Y_c)) having the same magnitude as **V**, [m/sec]
Ω_z:	Angular velocity about the instantaneous global center of curvature of the vehicle's path, [rad/sec]
$X(t)$, $Y(t)$:	Global position of the center of mass of the vehicle at any given time [m]
$X(t+\Delta t)$, $Y(t+\Delta t)$:	Global position of the center of mass of the vehicle at any given time [m]
X_c, Y_c:	Global position of the center of curvature of the vehicle's instantaneous path of travel [m]

1.0 INTRODUCTION

1.1 The Instantaneous Center Integration Algorithm Concept

Several unique unconstrained vehicle motion integration algorithms, referred to as Instantaneous Center Integration (ICI) algorithms, are developed and illustrated. The main objective of the ICI algorithms is to more accurately portray the simulated motion of a solid object, such as a road vehicle, in the most general environment, i.e. unconstrained. Therefore, any contact constraints, such as rolling constraints, will be ignored; such as would be the case for a dynamic model. In general, in three-dimensional space, the ICI algorithms involve six degrees of freedom (DOF). These DOFs include lateral, longitudinal and vertical velocities ($\mathbf{V}_X$, $\mathbf{V}_Y$ and $\mathbf{V}_Z$ respectively), and the yaw, roll and pitch rates ($\Omega_\perp$, $\Omega_{||}$ and Ω_p).

The basic concept behind the ICI algorithms is illustrated in Figure 1. The idea behind the ICI algorithms originates from the well-known instantaneous centers principle for performing velocity analysis on mechanisms [Erdman, 1984]. Erdman uses instantaneous centers for studying two-dimensional mechanisms with constrained motion. The ICI algorithms presented here focus on determining the motion of an unconstrained system through numerical integration. The circle, shown in Figure 1, which represents the instantaneous path of the vehicle is determined first by choosing an arbitrary $\mathbf{V}$ and Ω. The circle is constructed from moving perpendicular to the plane implied by $\mathbf{V}$ and Ω. The direction of Ω implies the direction of travel via the right-hand-rule, which determines the side of the plane that the circle is to be constructed. The motion of a solid object (vehicle) can be completely defined by the vehicle's translational velocities $\mathbf{V}$ and Ω relative to a fixed point on the vehicle, such as the center of mass (CM). The approach used in ICI algorithms

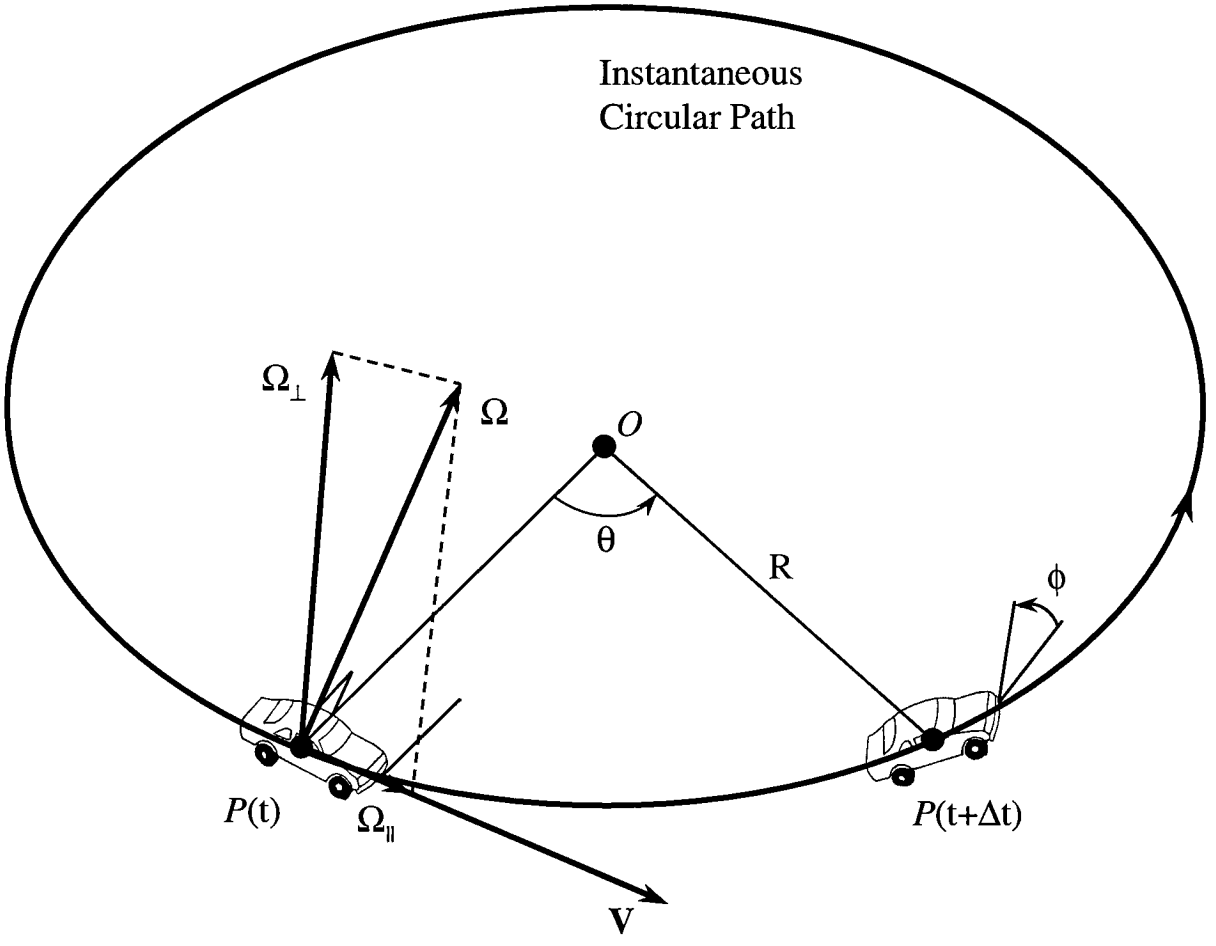

Figure 1: Illustration of the "Instantaneous Center Integration" (ICI) Algorithm -Three-Dimensional Case

takes into account the fact that the vehicle is simultaneously moving in the direction of the velocity vector, while also turning and rolling. The linear velocity vector is defined as the vector sum of the longitudinal velocity, $\mathbf{V}_x$, the lateral velocity, $\mathbf{V}_y$ and the vertical velocity, $\mathbf{V}_z$ and it depicts the direction and speed that the vehicle is moving at any instant of time, t. The second vector, Ω, is the angular velocity vector, which can be broken down into three separate angular velocities. The yaw rate, $\Omega_\perp$, is the projection of Ω perpendicular to the plane in which the vehicle is traveling. The roll rate, $\Omega_\parallel$, is the projection of Ω along the direction of $\mathbf{V}$. Lastly, the pitch rate, Ω_p is the projection of Ω along the radial direction [Almeida, 1997]. For the case shown in Figure 1, Ω_p, by construction, is equal to zero. When numerically integrating, a small enough time increment, Δt, is chosen so that the velocity vector and the yaw rate can be considered approximately constant over the time increment. This is key to how the integration algorithms function. At any time the instantaneous radius of curvature, R, can be determined by dividing the magnitude of the velocity vector, $|\mathbf{V}|$, by the magnitude of the yaw rate vector [Hibbeler, 1998].

$$R = \frac{|\mathbf{V}|}{|\Omega_\perp|} \tag{1}$$

It should be noted that if $\Omega_\perp$ is equal to zero then the radius becomes indeterminate. As $\Omega_\perp$ approaches zero the radius approaches infinity. This shows that the vehicle is translating in a straight line with a velocity $|\mathbf{V}|$, while rolling about the $\mathbf{V}$ direction with a magnitude equal to $|\Omega|$. The ICI algorithms are not recommended for cases in which there are sufficiently small angular velocities, therefore this problem will, most likely, never affect the performance of

the ICI algorithms. The instantaneous center of the radius of curvature is constantly changing for each time increment. The curve produced by the positions of the instantaneous center is called the space centrode [Beer, 1988]. The angle that the vehicle has traveled through, θ, in a given time increment, Δt, can be determined by multiplying the magnitude of the yaw rate by the time increment.

$$\theta = |\Omega_{\perp}|\Delta t \tag{2}$$

The angle that the vehicle has been rolled about **V**, ϕ, in a given time increment can be determined by multiplying the magnitude of the roll rate by the time increment.

$$\phi = |\Omega_{||}|\Delta t \tag{3}$$

The resultant combined translational and rotational motion will cause the vehicle to traverse along a circular path, while rolling along a circular trajectory.

An effort has been made to determine whether or not an algorithm similar to the ICI algorithms has been previously developed in the academic world as well as in the computer simulation industry. Current International Society for Optical Engineering (SPIE), Institute of Electrical and Electronics Engineers (IEEE), and American Society of Mechanical Engineers (ASME) Proceedings, Society of Automotive Engineers (SAE) papers and journal articles were searched with respect to vehicle related integration algorithms (i.e. [Almeida *et al.*, 1997]; [Hennessey *et al.*, 1995]; and [Bicchi *et al.*, 1996]). No mention of an algorithm similar to the ICI algorithms was found. Also, several books were searched which related to the topic of computer simulations and integration algorithms (i.e. [Beckett and Hurt, 1967]; [Elgeln-Mullges and Uhlig, 1996]; [Haug, 1992]; [Haug and Deyo, 1991]; and [Will and Zak, 1997]). Again, no mention of an algorithm similar to the ICI algorithms was found.

Companies in the fields of educational computer simulation tools and computer entertainment (such as video games) were contacted. Unfortunately, much of the sought after information is proprietary. But, after receiving information from the following companies that produce computer simulation software, it is likely that they do not use an algorithm similar to the ICI algorithms introduced in this paper. Rather, they use more generic numerical integration algorithms, such as those studied in numerical analysis. The companies who responded were Knowledge Revolution, Mechanical Dynamics, CADSI, and The Mathworks, Inc. Knowledge Revolution, developers of Working Model, use a form of the Kutta-Merson algorithm [Reckdahl, 1995]. Mechanical Dynamics, developers of ADAMS™, use the Gear Stiff Integration algorithm, a PECE (Predict-Evaluate-Correct-Evaluate) algorithm as well as the Newton-Raphson algorithm [Sadjak, 1998]. CADSI, developers of DADS™, use a PECE algorithm, a variable algorithm (Adams-Bashforth), a Backward Differentiation Formula algorithm, and a Fourth-Order Runge-Kutta algorithm [Kollman, 1998]. Lastly, The Mathworks, developers of MATLAB™ and SIMULINK™, use various basic integration algorithms, such as those mentioned above [Ashforth, 1998]. These algorithms, while complicated, are very general in purpose and not specifically designed for vehicle motion simulations. The ICI algorithms are specialized for motion simulation, especially unconstrained vehicle motion.

The ICI algorithms can accurately describe the motion of a solid object, such as a road vehicle and will be able to be used in creating very realistic computer simulations and animations. Previous vehicle simulations and animations performed by Hennessey, et al. [Hennessey *et al.*, 1995] indicate that the quality of the animation is sensitive to the choice of integration algorithm, as well as the time increment. Three specific ICI algorithms are

developed. These algorithms include the (1) "pseudo-exact", (2) first-order approximation, and (3) second-order approximation algorithms. The "pseudo-exact" algorithm is the most accurate and is the basis for the other algorithms. The first-order and second-order approximations are simplifications of the "pseudo-exact" algorithm and are less accurate, though not as computationally expensive. Higher-order approximations can be made (i.e. greater than second order), but the second-order approximation results in sufficiently accurate results for typical applications, as will be shown later. Also, there is a certain amount of noise inherent in the computer system, which limits the accuracy of the simulation. The use of higher-order approximations will only be able to result in a simulation with the accuracy that is governed by the noise. Therefore, the higher-order approximations will not be examined. The following sections contain derivations of each algorithm and their results are compared computationally, and through the use of simulations. For simplification, only the two-dimensional case will be studied.

2.0 DERIVATION OF ALGORITHMS

2.1 "PSEUDO-EXACT" ALGORITHM FOR TWO-DIMENSIONAL CASE

For the two-dimensional problem, consider a road vehicle traveling on a nominally circular path of radius R and with a constant speed |V|. The motion of this vehicle can be simulated through the use of numerical integration, such as the ICI algorithms. The "pseudo-exact" ICI algorithm is the most general form of the ICI algorithms and therefore, it is derived first. This algorithm will be referred to as a "pseudo-exact" algorithm because it assumes that |V| and R are constant over the given time increment, Δt, which in general will not be the case. For other scenarios though, a small enough Δt may be chosen that make |V|

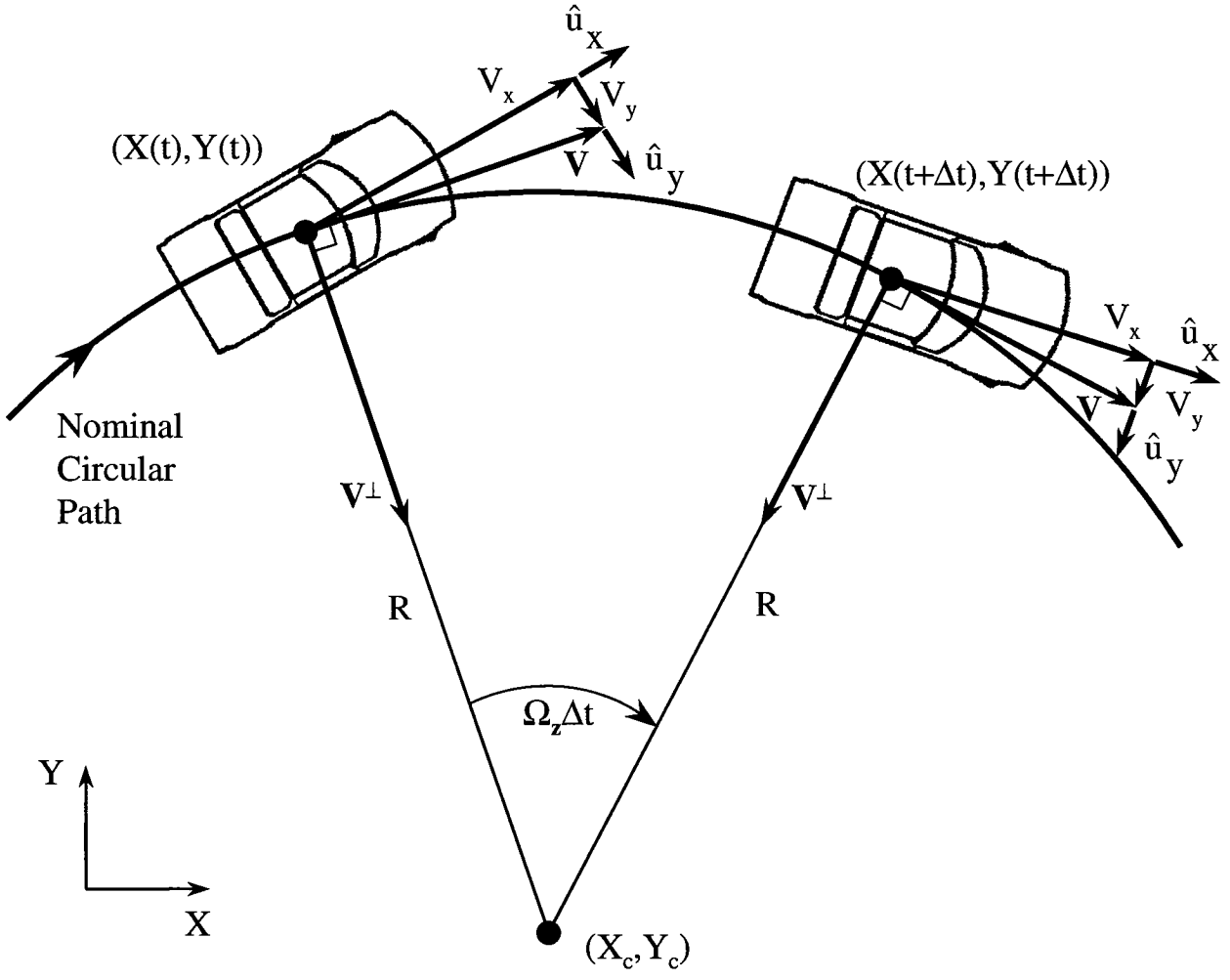

Figure 2: Instantaneous Center Integration Algorithm for Two-Dimensional Case

and R approximately constant. The following is a derivation of the equations used in this algorithm.

Figure 2 depicts the problem that these algorithms intend to solve [Wong, 1993]. The problem is to determine the position and orientation of a vehicle after a time increment, Δt, when the position and orientation of the vehicle are known at any time t. The vehicle is traveling on the nominally circular path, at the radius R, with a velocity **V**. For any given time the instantaneous radius of curvature of the vehicle's path, R, can be determined by equation (1). It should be noted that a left-hand coordinate frame is used for the vehicle's coordinates. Normally, a right-hand coordinate frame is used in which the z-axis would be down (into the paper), but the left-hand frame was chosen so that, for clarity, the velocities are positive. The velocity vector can be separated into its two components V_x and V_y. The heading unit vector is defined as the actual direction the vehicle is facing at any time:

$$\hat{\mathbf{u}}_x = \begin{bmatrix} u_x \\ u_y \end{bmatrix} \tag{4}$$

Therefore, the unit vector perpendicular to $\hat{\mathbf{u}}_x$ is:

$$\hat{\mathbf{u}}_y = \begin{bmatrix} u_y \\ -u_x \end{bmatrix} \tag{5}$$

The velocity vector is defined as the vector sum of the velocity components and their respective unit vectors and can be written as:

$$\mathbf{V} = V_x \hat{\mathbf{u}}_x + V_y \hat{\mathbf{u}}_y \tag{6}$$

Substituting for $\hat{\mathbf{u}}_x$ and $\hat{\mathbf{u}}_y$ yields:

$$\mathbf{V} = \left(V_x u_x + V_y u_y, V_x u_y - V_y u_x\right) \tag{7}$$

Therefore, the vector perpendicular to **V** can be shown as:

$$\mathbf{V}^{\perp} = \left(V_x u_y - V_y u_x, -V_x u_x - V_y u_y\right) \tag{8}$$

At any given time there exists a center of curvature of the path of the vehicle's motion, (X_c, Y_c), which is constantly being re-evaluated. For this case, since the path is a circle, the center of curvature does not change. However, in most scenarios the center of curvature will constantly be changing. Therefore, there will exist a need to continually re-evaluate the center of curvature, as well as other quantities. This instantaneous position of the center of curvature can be determined by adding the radial distance of the vehicle to the center of curvature to the current position of the vehicle (X(t), Y(t)):

$$\left(X_c, Y_c\right) = \left(X(t), Y(t)\right) + \frac{R\mathbf{V}^{\perp}}{\left\|\mathbf{V}^{\perp}\right\|} \tag{9}$$

This instantaneous position of the center of curvature is then used to determine the new position of the vehicle. The motion of the vehicle can be discretized according to the chosen Δt. To determine the new position of the CM of the vehicle after the time increment Δt a rotation matrix for a two-dimensional problem is applied [Haug, 1992]. The angle that the vehicle has traveled through during this time period is defined as $\Omega_z \Delta t$. Therefore, the new position of the CM of the vehicle is:

$$\begin{bmatrix} X(t+\Delta t) \\ Y(t+\Delta t) \end{bmatrix} = \begin{bmatrix} \cos\Omega_z \Delta t & \sin\Omega_z \Delta t \\ -\sin\Omega_z \Delta t & \cos\Omega_z \Delta t \end{bmatrix} \begin{bmatrix} X(t) - X_c \\ Y(t) - Y_c \end{bmatrix} + \begin{bmatrix} X_c \\ Y_c \end{bmatrix} \tag{10}$$

The X-component will be examined first. The X-component of the position of the CM of the vehicle after Δt has elapsed, $X(t+\Delta t)$. is determined by:

$$X(t+\Delta t) = \cos\Omega_z\Delta t(X - X_c) + \sin\Omega_z\Delta t(Y - Y_c) + X_c \tag{11}$$

After substituting for $X\text{-}X_c$, $Y\text{-}Y_c$ and X_c and simplifying:

$$X(t+\Delta t) = (\cos\Omega_z\Delta t - 1)\frac{(V_y u_x - V_x u_y)}{\Omega_z} + \sin\Omega_z\Delta t\frac{(V_x u_x + V_y u_y)}{\Omega_z} + X(t) \tag{12}$$

This is the "pseudo-exact" solution for the X-component of the position of the CM of the vehicle at time $t + \Delta t$. Now, in the limit as $\Delta t \to 0$ the change of the X-component becomes, simply:

$$X(t+\Delta t) - X(t) = V_x u_x + V_y u_y \tag{13}$$

The time-rate of change of the X-component is:

$$\dot{X} = \lim_{\Delta t \to 0}\left[\frac{X(t+\Delta t) - X(t)}{\Delta t}\right] = V_x u_x + V_y u_y \tag{14}$$

or: $\dot{X} = -\mathbf{V}\cdot\hat{\mathbf{u}}_x$ when $\Delta t \to 0$.

Similarly, for the Y-component of the position of the CM of the vehicle:

$$Y(t+\Delta t) = -\sin\Omega_z\Delta t(X - X_c) + \cos\Omega_z\Delta t(Y - Y_c) + Y_c \tag{15}$$

After substituting for $X\text{-}X_c$, $Y\text{-}Y_c$ and Y_c and simplifying:

$$Y(t+\Delta t) = \sin\Omega_z\Delta t\frac{(V_x u_y - V_y u_x)}{\Omega_z} + (\cos\Omega_z\Delta t - 1)\frac{(V_x u_x + V_y u_y)}{\Omega_z} + Y(t) \tag{16}$$

This is the "pseudo-exact" solution for the Y-component of the position of the CM of the vehicle at time $t + \Delta t$. In the limit as $\Delta t \to 0$, the change in Y-component becomes, simply:

$$Y(t+\Delta t) - Y(t) = V_y u_x - V_x u_y \tag{17}$$

The time-rate of change of the Y-component is:

$$\dot{Y} = \lim_{\Delta t \to 0}\left[\frac{Y(t+\Delta t) - Y(t)}{\Delta t}\right] = V_y u_x - V_x u_y \tag{18}$$

or: $$\dot{Y} = \mathbf{V} \cdot \hat{\mathbf{u}}_y \quad \text{when } \Delta t \to 0.$$

The new heading vector after Δt is determined by:

$$u_x(t+\Delta t) = \cos\Omega_z \Delta t \cdot \mathbf{u}_x(t) + \sin\Omega_z \Delta t \cdot u_y(t) \tag{19}$$

$$u_y(t+\Delta t) = -\sin\Omega_z \Delta t \cdot u_x(t) + \cos\Omega_z \Delta t \cdot u_y(t) \tag{20}$$

In equations (19) and (20) the old heading vector is multiplied by the rotation matrix to determine the new heading vector. In this "pseudo-exact" ICI algorithm both sine and cosine terms appear. Evaluation of these transcendental terms can become computationally expensive to a simulation program. A simpler algorithm must be developed in order to reduce the amount of calculations required, yet retaining reasonable accuracy. The following two sections contain a first-order approximation algorithm and a second-order approximation algorithm, both of which require fewer calculations, thereby making the algorithm easier to compute.

2.2 FIRST-ORDER APPROXIMATION ALGORITHM

The first-order approximation ICI algorithm, which traditionally is used in computer simulations and animations, is a simplification of the "pseudo-exact" ICI algorithm. The first-order approximation algorithm approximates the transcendental components and therefore eliminates them, thus making the algorithm less computationally expensive. This first-order approximation algorithm involves sequentially translating the vehicle and then rotating it for every time increment Δt. To determine the new vehicle's position the vehicle is translated in

the direction of the heading vector $\hat{\mathbf{u}}_x$ then rotated according to the angular velocity Ω_z.

Derivation of the first-order approximation algorithm is similar to that of the "pseudo-exact" algorithm. Consider equations (12) and (16) from the previous section. After rearranging these equations, they become:

$$X(t+\Delta t) = (\cos\Omega_z\Delta t - 1)\frac{(V_y u_x - V_x u_y)}{\Omega_z} + \sin\Omega_z\Delta t\frac{(V_x u_x + V_y u_y)}{\Omega_z} + X(t)$$

$$Y(t+\Delta t) = \sin\Omega_z\Delta t\frac{(V_x u_y - V_y u_x)}{\Omega_z} + (\cos\Omega_z\Delta t - 1)\frac{(V_x u_x + V_y u_y)}{\Omega_z} + Y(t)$$

Substitute the power series expansions for sine and cosine:

$$\sin\Omega_z\Delta t = \Omega_z\Delta t - \frac{(\Omega_z\Delta t)^3}{3!} + \frac{(\Omega_z\Delta t)^5}{5!} - \ldots \tag{21}$$

$$\cos\Omega_z\Delta t = 1 - \frac{(\Omega_z\Delta t)^2}{2!} + \frac{(\Omega_z\Delta t)^4}{4!} - \ldots \tag{22}$$

For the first-order approximation:

$$\sin\Omega_z\Delta t \cong \Omega_z\Delta t \tag{22}$$

$$\cos\Omega_z\Delta t \cong 1 \tag{24}$$

The new position of the CM of the vehicle after Δt is determined by:

$$X(t+\Delta t) = X(t) + (V_x u_x + V_y u_y)\Delta t \tag{25}$$

$$Y(t+\Delta t) = Y(t) + (V_x u_y - V_y u_x)\Delta t \tag{26}$$

In equations (25) and (26), the new position of the CM of the vehicle is determined by adding the amount the CM of the vehicle moves during the time increment Δt to the previous position (X(t), Y(t)).

The new values of the heading vector are determined by:

$$u_x(t+\Delta t) = u_x(t) + \Omega_z \Delta t \cdot u_y(t) \tag{27}$$

$$u_y(t+\Delta t) = -\Omega_z \Delta t \cdot u_x(t) + u_y(t) \tag{28}$$

In equations (27) and (28) the new heading vector of the vehicle is found by multiplying the first-order approximation of the rotation matrix by the previous heading vector.

To compensate for the seemingly inconsistent motion that can be produced by this algorithm, Δt needs to be made sufficiently small. Since this algorithm does not contain complicated formulas or mathematical equations, decreasing Δt does not result in any computational complications, which worsen the performance of the simulation. However, if the angular velocity is too large (i.e. tight cornering), the trajectory recommended by this algorithm tends to drift away from the desired path (the exact solution) quite rapidly, as will be depicted in the simulation section of this paper. This drifting can result in a non-realistic simulation.

2.3 SECOND-ORDER APPROXIMATION ALGORITHM

The second-order approximation ICI algorithm takes the approximation of the "pseudo-exact" algorithm one more step by substituting the power series expansions of equations (12) and (16) for the second-order case, which are:

$$\sin \Omega_z \Delta t \cong \Omega_z \Delta t \tag{29}$$

$$\cos \Omega_z \Delta t \cong 1 - \frac{(\Omega_z \Delta t)^2}{2!} \tag{30}$$

The new value of the X-component of the position of the CM of the vehicle is to be determined.

$$X(t+\Delta t) \cong X(t) + \Omega_z \Delta t \left[\frac{V_x u_x + V_y u_y}{\Omega_z} \right] - \left[\frac{V_y u_x - V_x u_y}{\Omega_z} \right] \frac{(\Omega_z \Delta t)^2}{2} \tag{31}$$

Simplifying equation (31) results in:

$$X(t+\Delta t) \cong X(t) + (V_x u_x + V_y u_y)\Delta t + \frac{\Omega_z}{2}(V_y u_x - V_x u_y)\Delta t^2 \tag{32}$$

The new value of the Y-component of the position of the CM of the vehicle is to be determined.

$$Y(t+\Delta t) \cong Y(t) + \Omega_z \Delta t \left[\frac{V_x u_y - V_y u_x}{\Omega_z} \right] - \left[\frac{V_x u_x + V_y u_y}{\Omega_z} \right] \frac{(\Omega_z \Delta t)^2}{2} \tag{33}$$

Simplifying equation (33) results in:

$$Y(t+\Delta t) \cong Y(t) + (V_x u_y - V_y u_x)\Delta t - \frac{\Omega_z}{2}(V_x u_x + V_y u_y)\Delta t^2 \tag{34}$$

Finally, the new heading vector is determined by:

$$u_x: \; u_x(t+\Delta t) = \left(1 - \frac{\Omega_z \Delta t^2}{2}\right) u_x(t) + \Omega_z \Delta t \cdot u_y(t) \tag{35}$$

$$u_y: \; u_y(t+\Delta t) = \Omega_z \Delta t \cdot u_x(t) + \left(1 - \frac{\Omega_z \Delta t^2}{2}\right) u_y(t) \tag{36}$$

In this second-order approximation algorithm sine and cosine are not present; therefore, it is less computationally expensive for the simulation program than the "pseudo-exact" algorithm. It is more computationally expensive, per iteration, than the first-order approximation, but, as will be illustrated, it is more accurate. Under tight cornering this algorithm will deviate from the desired path much more slowly than with the first-order algorithm.

2.4 SUMMARY OF ALGORITHMS:

A summary of the ICI algorithms is provided below with both position and orientation update information given. To avoid wrap-around ambiguity effects, a unit vector, containing two pieces of information is preferred, versus a heading angle, which contains only one piece of information.

"Pseudo-Exact"

$$\text{X: } X(t+\Delta t) = (\cos\Omega_z\Delta t - 1)\frac{(V_y u_x - V_x u_y)}{\Omega_z} + \sin\Omega_z\Delta t\frac{(V_x u_x + V_y u_y)}{\Omega_z} + X(t)$$

$$\text{Y: } Y(t+\Delta t) = \sin\Omega_z\Delta t\frac{(V_x u_y - V_y u_x)}{\Omega_z} + (\cos\Omega_z\Delta t - 1)\frac{(V_x u_x + V_y u_y)}{\Omega_z} + Y(t)$$

$$u_x:\ u_x(t+\Delta t) = \cos\Omega_z\Delta t\cdot u_x(t) + \sin\Omega_z\Delta t\cdot u_y(t)$$

$$u_y:\ u_y(t+\Delta t) = -\sin\Omega_z\Delta t\cdot u_x(t) + \cos\Omega_z\Delta t\cdot u_y(t)$$

First-Order

$$\text{X: } X(t+\Delta t) = X(t) + (V_x u_x + V_y u_y)\Delta t$$

$$\text{Y: } Y(t+\Delta t) = Y(t) + (V_x u_y - V_y u_x)\Delta t$$

$$u_x:\ u_x(t+\Delta t) = u_x(t) + \Omega_z\Delta t\cdot u_y(t)$$

$$u_y:\ u_y(t+\Delta t) = -\Omega_z\Delta t\cdot u_x(t) + u_y(t)$$

Second-Order

$$\text{X: } X(t+\Delta t) \cong X(t) + (V_x u_x + V_y u_y)\Delta t + \frac{\Omega_z}{2}(V_y u_x - V_x u_y)\Delta t^2$$

$$\text{Y: } Y(t+\Delta t) \cong Y(t) + (V_x u_y - V_y u_x)\Delta t - \frac{\Omega_z}{2}(V_x u_x + V_y u_y)\Delta t^2$$

$$u_x:\ u_x(t+\Delta t)=\left(1-\frac{\Omega_z\Delta t^2}{2}\right)u_x(t)+\Omega_z\Delta t\cdot u_y(t)$$

$$u_y:\ u_y(t+\Delta t)=\Omega_z\Delta t\cdot u_x(t)+\left(1-\frac{\Omega_z\Delta t^2}{2}\right)u_y(t)$$

3.0 SIMULATION OF ALGORITHMS

The simulation results of the three ICI integration algorithms studied in this paper for a baseline scenario are developed in order to compare the three algorithms to each other. Microsoft Excel© is used to compute and plot the results. The baseline scenario, as stated previously, is a vehicle traveling in a nominally circular path with a constant speed. The results of the three algorithms are simultaneously plotted for comparison purposes. Developing a simulation of the results shows their relevancy and significance for producing accurate computer simulations. In this instance, the "pseudo-exact" algorithm will always generate the exact solution, which is convenient for comparison purposes. This is due to the fact that |V| and Ω_Z are constant. However, for other curvilinear paths the "pseudo-exact" algorithm will not result in the exact solution, but it can accurately approximate it.

Initial conditions need to be established. For simplification, the vehicle begins at the position (-10,0) [m] with a constant speed |V| of 100 m/sec, and an angular velocity Ω_Z of 10 rad/sec about the origin. The initial heading vector quantities are $u_x(0)=0$ and $u_y(0)=1$. The radius is set at 10 m. These initial conditions and variables are chosen so that the desired result is a circle with R = 10 m and (X_C,Y_C) = (0,0) [m].

An appropriate time increment is determined so that the simulation is most effective

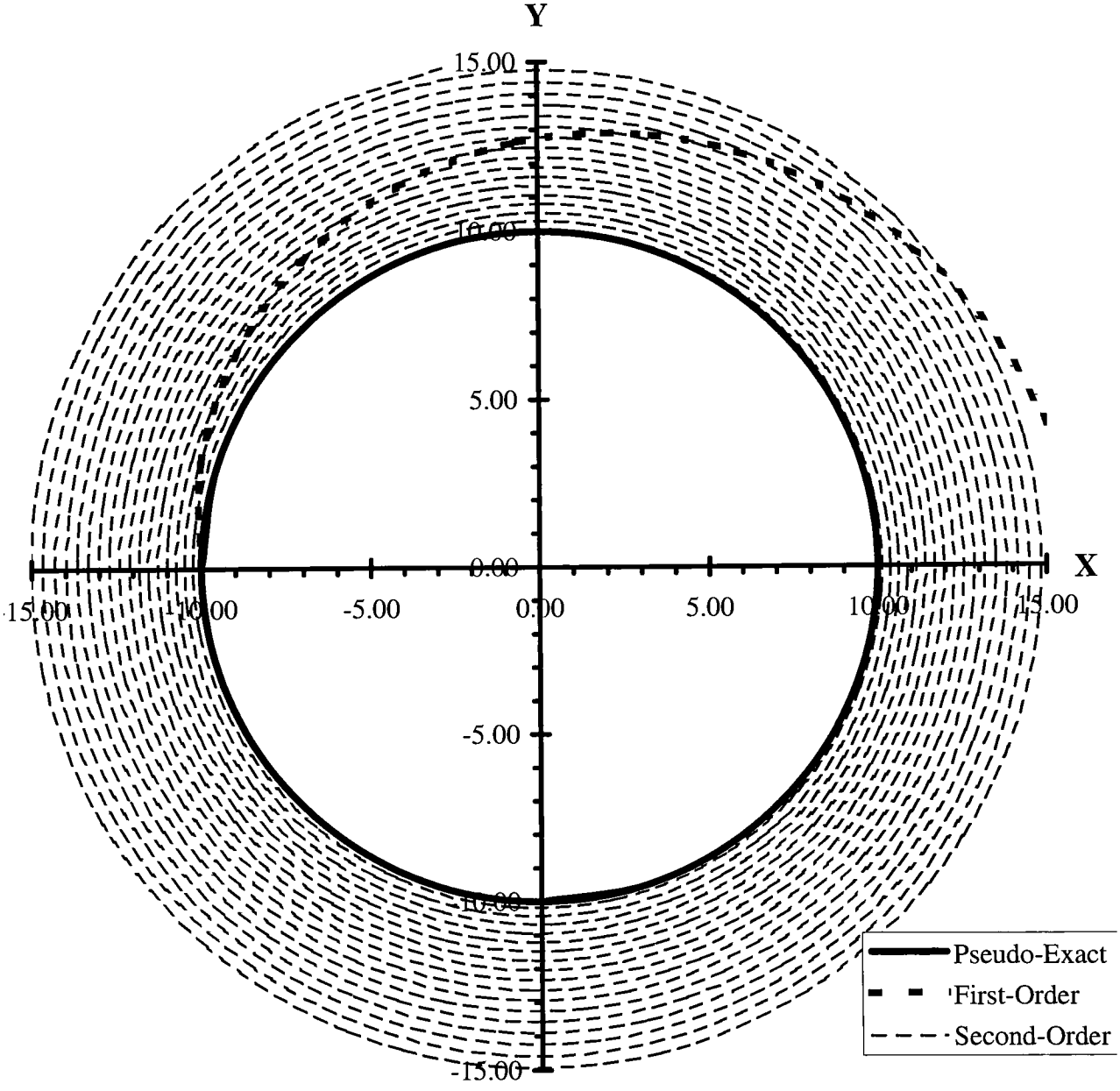

Figure 3: Comparison of Algorithms for Constant Velocities, k=20

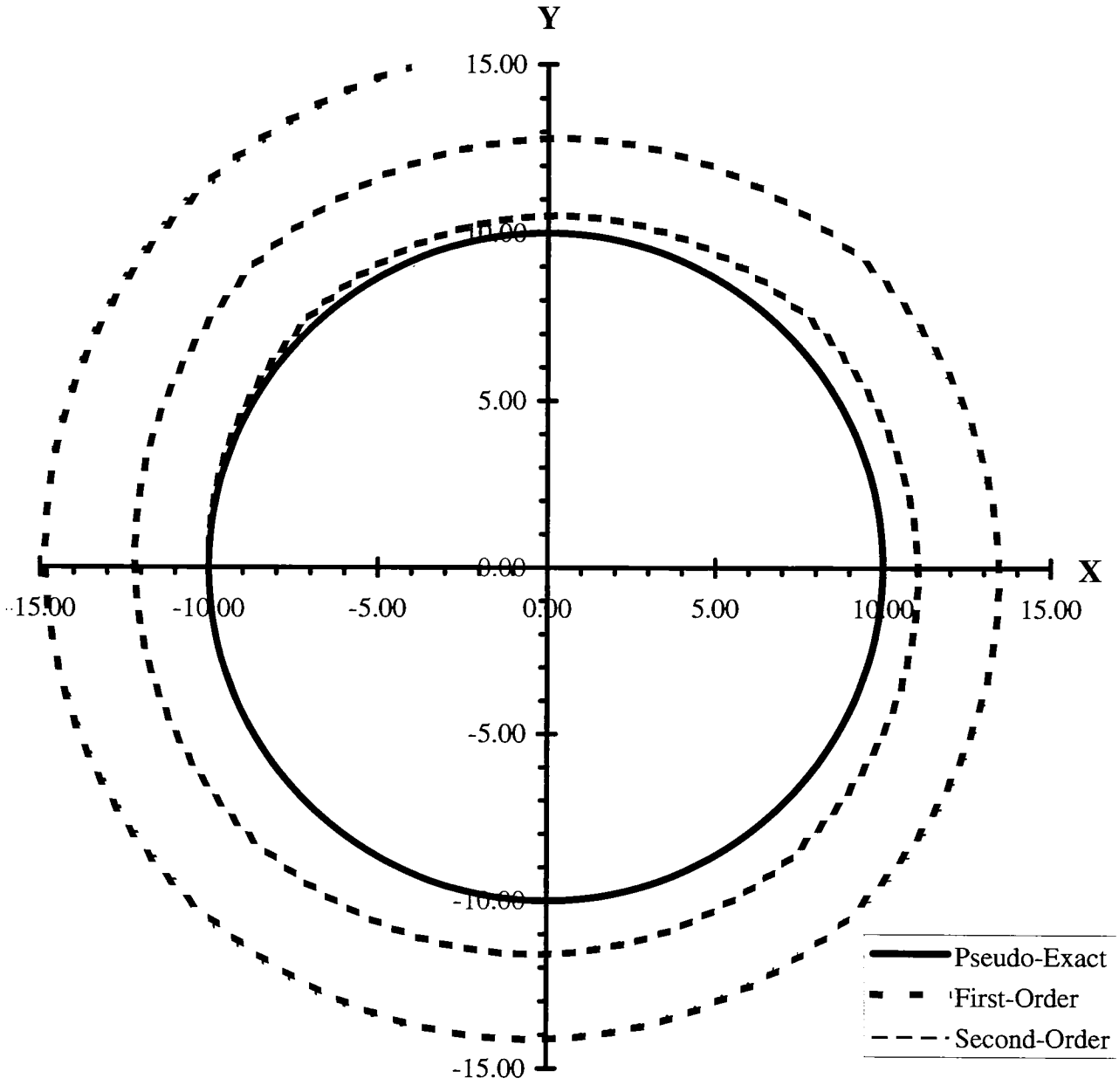

Figure 4: Comparison of Algorithms for Constant Velocities, k=100

The user determines the value k, which is defined as the number of data points per revolution. Figures 3 and 4 show the results when k is chosen to be 20 and 100 respectively. In Figure 4 the second-order approximation results are almost exactly the same as the "pseudo-exact" results. Naturally, the larger k is the smaller Δt is, and in turn the more accurate the results. As noted earlier, the "pseudo-exact" algorithm results in the exact answer, which, in this case, is a circle. The second-order approximation results in less accurate results and first-order approximation results in much less accurate results for a given time increment. For this example, the first-order approximation does not result in acceptable results with any of the k values chosen. Acceptable results are defined as results that are accurate enough to be used in a simulation for a given application. In fact, a k of about 1,000 will have to be chosen for the first-order approximation to give acceptable results.

In order for the most appropriate Δt value to be chosen the effect of changing Δt on the resultant error and the effect of the resultant error on Δt should be examined. The largest Δt is to be chosen while maintaining an acceptable error for the simulation to be most efficient. There are two possible methods for comparing the errors and choosing the most appropriate time step; they are the direct integration error comparison method and the indirect integration error comparison method.

The direct integration error comparison method involves setting a Δt, or equivalent k, and determining the corresponding error. These values are compared with each other and results for the chosen values of k are shown in Table 1. The formula for determining the percent error, ε, is:

$$\varepsilon = \frac{\left\| (X - X_e), (Y - Y_e) \right\|_2}{\left\| (X_e, Y_e) \right\|_2} \times 100\% \tag{38}$$

Table 1: Direct Integration Error Comparison

k	No. Revolutions	ε_1	ε_2	Ratio ($\varepsilon_2/\varepsilon_1$)
20 (Δt=0.0314 sec)	1	159.42	10.43	0.065
	2	565.79	21.09	0.035
	3	1601.57	31.97	0.020
	4	4245.78	43.02	0.010
	5	11007.44	54.25	0.005
30 (Δt=0.0209 sec)	1	91.24	4.61	0.051
	2	264.47	9.24	0.035
	3	594.41	13.91	0.023
	4	1221.04	18.60	0.015
	5	2412.44	23.31	0.010
	6	4678.53	28.05	0.006
	7	8990.11	32.81	0.004
	8	11338.80	37.59	0.003
	9	23870.55	42.38	0.002
	10	63891.44	47.12	0.001
50 (Δt=0.0126 sec)	1	48.11	1.65	0.034
	2	119.30	3.31	0.028
	3	224.54	4.97	0.022
	4	350.66	6.63	0.019
	5	610.42	8.30	0.014
	6	950.89	9.96	0.010
	7	1454.43	10.63	0.007
	8	2199.21	13.30	0.006
	9	3300.83	14.97	0.005
	10	4930.35	16.63	0.003
100 (Δt=0.0063 sec)	1	21.79	0.41	0.019
	2	48.33	0.83	0.017
	3	80.65	1.24	0.015
	4	120.00	1.65	0.014
	5	167.92	2.07	0.012
	6	220.73	2.48	0.011
	7	281.29	2.89	0.010
	8	349.68	3.32	0.009
	9	423.98	3.72	0.009
	10	506.63	4.14	0.008

where X_e and Y_e are the exact ("pseudo-exact" in this case) values and $|(X_e, Y_e)|$ is the magnitude of the vector from the center of the circle (in this case the origin) to the exact values of the point on the circle. It should be noted that $|(X_e, Y_e)|$ is never zero for the baseline scenario because the "pseudo-exact" algorithm never results in a value that intersects the origin. The error values are taken at the completion of every revolution. The first-order approximation error is defined as ε_1 and the second-order approximation error is defined as ε_2. For the first-order approximation, the resulting error from each Δt chosen is considerably higher than the respective error of the second-order approximation.

The indirect integration error comparison method involves setting a desired error value and determining the required Δt value to achieve this value after a pre-determined number of revolutions (i.e. five in this case). Table 2 shows the results of this method. The required Δt for the first-order approximation is much less than that of the second-order approximation. This shows that a smaller, therefore harder to computationally manipulate, time increment must be chosen in order for the first-order approximation to result in an equivalent error to the second-order approximation.

It is reasonable to say that for this application an error of 10% after five revolutions can be considered acceptable. This is also true for applications, such as accident reconstruction, where there is a high yaw rate and low number of revolutions. Both integration error comparison methods show that the Δt required for this result is about 6.1E-04 seconds for the first-order approximation algorithm and about 1.4E-02 seconds for the second-order approximation algorithm. In other words, the Δt for the first-order

Table 2: Indirect Integration Error Comparison

Error After 5 Revolutions	**Δt_1 – First-Order (sec)**	**Δt_2 – Second-Order (sec)**	**Ratio ($\Delta t_2/\Delta t_1$)**
50%	2.6E-03	2.9E-02	11.15
40%	2.1E-03	2.7E-02	12.86
30%	1.7E-03	2.5E-02	14.71
25%	1.4E-03	2.2E-02	15.36
20%	1.2E-03	1.8E-02	15.00
15%	9.3E-04	1.6E-02	17.78
10%	6.1E-04	1.4E-02	23.33
5%	3.1E-04	1.0E-02	32.26
2%	1.3E-04	6.5E-03	50.00
1%	6.3E-05	4.5E-03	75.00

Note: The "pseudo-exact" algorithm is not included in the indirect error comparison method because no error results from the "pseudo-exact" algorithm for the baseline scenario.

approximation algorithm is approximately 23 times as large as that of the second-order approximation method. Considering that there are relatively fewer extra calculations in the second-order approximation algorithm than in the first-order approximation algorithm (less than 3 times as many), it is clear that the second-order approximation algorithm is more efficient.

4.0 STABILITY OF THE INTEGRATION ALGORITHMS

The stability of the ICI algorithms must also be examined in order to ensure that the algorithms will result in accurate simulations. Stability, in this context, generally refers to the ability of the algorithm to produce sufficiently accurate results over the range of interest. If the algorithm becomes unstable, the results will be inconsistent with what should actually occur. The stability is directly affected by the chosen time increment, or in this case the chosen k.

There exists a range of values for k for which stable results are produced. The lower and upper bounds of k need to be determined. In the case studied there is not an upper bound for k, which equates to the lower bound of the time increment. Determination of the lower bound for k can be done by calculating the *Nyquist frequency* or *folding frequency* [Stanley *et al.*, 1984]. The Nyquist frequency, f_0, is defined as:

$$f_0 = \frac{f_s}{2} \tag{39}$$

where f_s is the sampling frequency (how often the data is recorded), and f_0 is the lowest frequency that can be processed and still be stable. Expressing f_0 in terms of the period T

yields:

$$f_0 = \frac{1}{2T} \tag{40}$$

The period is equal to the time increment multiplied by the number of data points in one revolution. Therefore, f_0 becomes:

$$f_0 = \frac{1}{2k \cdot \Delta t} \tag{41}$$

There must be at least two data points taken for each cycle, one minimum and one maximum. This is true of any cycle. If there are less than two data points per revolution there will be times during the simulation in which there is only one data point in a revolution, which is not possible. For the case studied in this paper this would translate into a k of 2 (two data points per revolution). Plugging 2 into equation (37) for k, with the same Ω used previously, 10 rad/sec, the corresponding Δt would be 0.314 seconds. When using this value, the algorithm becomes unstable, as illustrated in Figure A1, in the appendix. Therefore, a larger k must be chosen for the least lower bound. A k of 4 gives inaccurate, yet stable, results, as shown in Figure A2, in the appendix. Related to this is the fact that any k less than 4 will result in a Y-component of the vehicle's velocity in the opposite direction that the vehicle is actually traveling. For example, if the vehicle's initial velocity is set to be exclusively in the positive Y-direction, and k is less than 4, the velocity after the first time increment will have a negative Y-component. This will result in an unstable simulation. Thus, as a result of this analysis the conclusion can be made that there is no upper bound for k and the practical lower bound is 4.

5.0 COMPUTATIONAL COMPARISON

In comparing the computational efficiency of the ICI algorithms, the computational time to perform each algorithm is investigated. It is nearly impossible to determine the computational time required to perform the various operations used on modern computing machinery (i.e. N-bit floating-point additions, N-bit floating-point divisions, transcendental calls, etc.) due to the large number of factors that affect the processor speed. Therefore, representative approximations must be made in order to compare the ICI algorithms developed in this paper. The relative time required to perform an N-bit floating-point addition/subtraction is represented as 1 time unit. The relative time required to perform an N-bit floating-point multiplication/division is represented as 10 time units. Finally, the relative time required to perform a transcendental call (sine and cosine only) is represented as 100 time units.

Analysis of the number of computations in each ICI algorithm is done first. For the "pseudo-exact" algorithm there are 12 N-bit floating-point additions and/or subtractions, 28 N-bit floating-point multiplications and/or divisions, and 8 N-bit transcendental calls. For the first-order approximation algorithm there are 6 N-bit floating-point additions and/or subtractions and 10 N-bit floating-point multiplications and/or divisions. For the second-order approximation algorithm there are 12 N-bit floating-point additions and/or subtractions and 32 N-bit floating-point multiplications and/or divisions. This translates to 1,092 time units per calculation for the "pseudo-exact" algorithm, 106 time units for the first-order algorithm and 342 time units for the second-order algorithm. (The "pseudo-exact" algorithm is obviously too computationally expensive and therefore, will no longer be analyzed in this section. The first and second-order approximation algorithms will be examined further). At

first, the above results may seem to conclude that the first-order algorithm is the least computationally exhausting since it requires the fewest time units to complete one iteration. But, Table 3 shows that when a certain error is desired, the computational time for the second-order algorithm is always much less than for that of the first-order algorithm. (Note: the computational times shown in the table are over one full revolution). This result is due to the fact that the needed time increment for the second-order approximation to result in the desired error is much larger than the needed time increment for the first-order approximation (see Table 2). Since the time increment is so much larger, there are fewer calculations performed per revolution for the second-order approximation algorithm.

This computational analysis shows that the second-order approximation algorithm, which results in more accurate motion simulations, saves computational time. Thus, the user has two options. He will be able to use the same time increment as the first-order approximation algorithm to get more accurate results, or to use a coarser time increment to get similarly accurate results. Either way, the second-order approximation algorithm is the most appropriate ICI algorithm to use.

Table 3: Comparison of computational time for the first and second-order approximation algorithms

Error After 5 Revolutions	**First-Order (FO), [units of time]**			**Second-Order (SO), [units of time]**			**Ratio**
	+/- Time	*×/÷ Time*	*Total Time*	*+/- Time*	*×/÷ Time*	*Total Time*	FO/SO
50%	1.4E+03	2.4E+03	3.3E+04	2.6E+02	6.9E+02	2.5E+04	1.30
40%	1.8E+03	3.0E+03	4.1E+04	2.8E+02	7.4E+02	2.7E+04	1.50
30%	2.2E+03	3.7E+03	5.0E+04	3.0E+02	8.0E+02	2.9E+04	1.71
25%	2.7E+03	4.5E+03	6.1E+04	3.5E+02	9.4E+02	3.4E+04	1.79
20%	3.1E+03	5.2E+03	7.1E+04	4.2E+02	1.1E+03	4.1E+04	1.75
15%	4.2E+03	7.0E+03	9.5E+04	4.7E+02	1.3E+03	4.6E+04	2.07
10%	6.3E+03	1.0E+04	1.4E+05	5.4E+02	1.4E+03	5.2E+04	2.72
5%	1.2E+04	2.0E+04	2.8E+05	7.5E+02	2.0E+03	7.3E+04	3.76
2%	2.9E+04	4.8E+04	6.6E+05	1.2E+03	3.1E+03	1.1E+05	5.82
1%	6.3E+04	1.0E+05	1.4E+06	1.7E+03	4.5E+03	1.6E+05	8.73

Note: The "pseudo-exact" algorithm's computational time is not included in Table 3 because no error results from the "pseudo-exact" algorithm for the baseline scenario

6.0 CONCLUSION

The three unique ICI algorithms: the "pseudo-exact" algorithm, the first-order approximation algorithm and second-order approximation algorithm have been derived and compared. These ICI algorithms offer significant advantages for vehicle simulations. They are designed specifically for vehicle simulations, especially ones in which there are high angular rates present, such as would occur during tight cornering and/or possibly during an accident.

These algorithms were used to simulate a vehicle traversing a nominally circular path with a constant speed. The three ICI algorithms' results were then simultaneously graphed, using Microsoft Excel©, to compare their appropriateness. For the baseline scenario presented in this paper, the "pseudo-exact" algorithm resulted in the actual exact solution. Both the first-order and second-order approximation algorithms drifted of course, yet the second-order approximation algorithm's results were much more accurate. For any given error, the second-order approximation algorithm is also much more computationally efficient than the other two algorithms. Overall, the second-order ICI algorithm is the most appropriate algorithm, of those considered, to use for computer simulations of vehicle motion.

REFERENCES

Almeida, J. F., F. Lobo, and J. B. Sousa. "A Hybrid Feedback Control System for a Nonholonomic Car-like Vehicle." *Proceedings-IEEE International Conference on Robotics and Automation Proceedings of the 1997 IEEE ICRA v.3.* April 20-25 1997. Albuquerque, NM. p. 2614-2619.

Ashforth, E., The Mathworks, Technical correspondence to Nathan Eggleton. 25 June 1998.

Beckett, R., and J. Hurt. *Numerical Calculations and Algorithms.* New York: McGraw-Hill, 1967.

Beer, Ferdinand P., and E. Russell Johnston, Jr. *Vector Mechanics for Engineers: Dynamics.* New York: McGraw-Hill, Inc. 1988.

Bernard, J. E., and C. L. Clover. "Validation of Computer Simulations of Vehicle Dynamics." SAE Paper No. 940231, February 1996.

Bicchi, A., G. Casalino, and C. Santilli. "Planning Shortest Bounded-Curvature Paths for a Class of Nonholonomic Vehicles Among Obstacles." *Journal of Intelligent and Robotic Systems. 16 (1996).* p. 387-405.

Elgeln-Mullges, G., and F. Uhlig. *Numerical Algorithms with Fortran.* Berlin: Springer-Verlag, 1996.

Erdman, Arthur G., and George N. Sandor. *Mechanism Design: Analysis and Synthesis.* London: Prentice-Hall, 1984.

Haug, E. J. *Intermediate Dynamics.* Englewood, NJ: Prentice-Hall, 1992.

Haug, E. J., and R. C. Deyo, ed. *Real-time Integration Methods for Mechanical System Simulation.* Berlin: Springer-Verlag, 1991.

Hennessey, M. P., C. Shankwitz, and M. Donath. *Sensor Based Virtual Bumpers for Collision Avoidance: Configuration Issues.* SPIE Photonics '95 Conference, Philadelphia, PA, October 1995.

Hibbeler, R. C. *Engineering Mechanics: Statics and Dynamics.* Upper Saddle River, NJ: Prentice-Hall, 1998.

Kollman, J. CADSI. Technical correspondence to Nathan Eggleton. 2 July 1998.

Reckdahl, K. "Faster Integration in Working Model v3.0." Simulation News-Fall '95. p. 3-4.

Sadjak, T. Technical correspondence to Nathan Eggleton. 1 July 1998.

Stanley, W. D., G. R. Dougherty, and R. Dougherty. *Digital Signal Processing.* Reston, VA: Reston Publishing Co., 1984.

Will, A. B., and S. H. Zak. "Modeling and Control of an Automated Vehicle." *Vehicle System Dynamics, v.27.* Lisse, Netherlands: Swets and Zeitlinger, 1997. p. 131-155.

Wong, J. Y. *Theory of Ground Vehicles.* New York: John Wiley and Sons, 1993.

Appendix

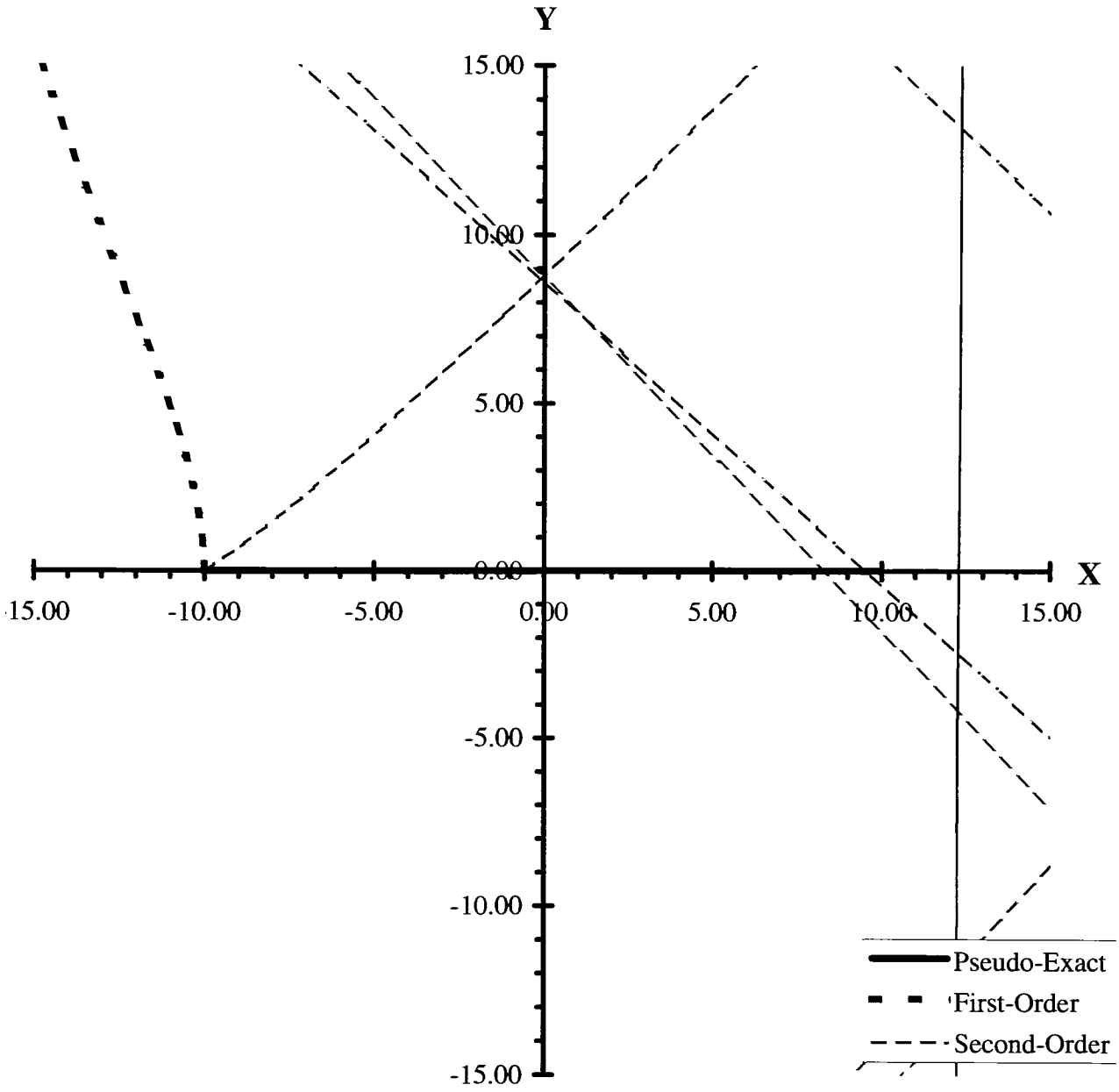

Figure A1: Comparison of Algorithms for Constant Velocities, k=2

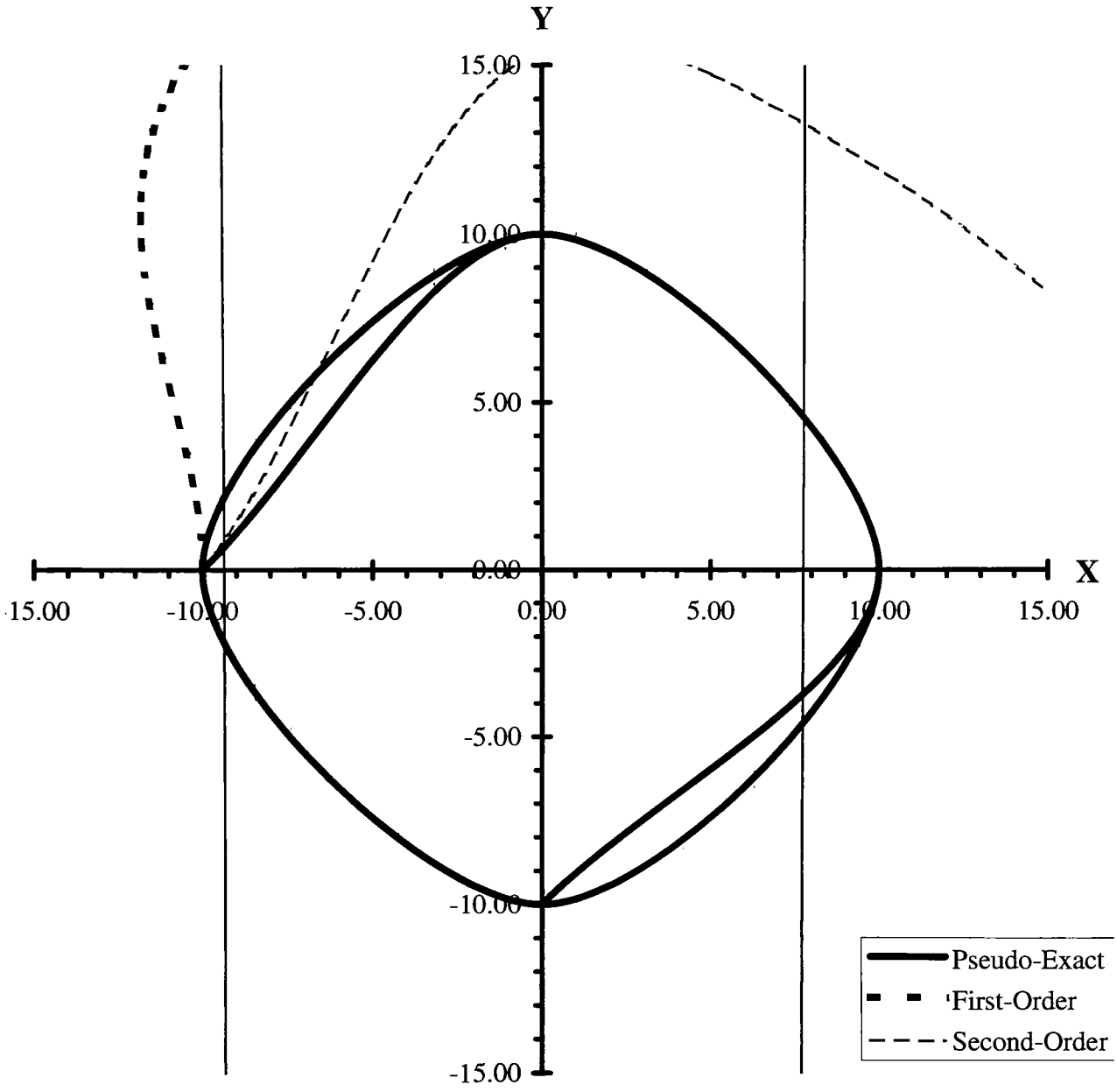

Figure A2: Comparison of Algorithms for Constant Velocities, k=4

DERIVATIONS OF EQUATIONS (12) AND (16)

Begin with Eq. (9)

$$(X_c, Y_c) = (X(t), Y(t)) + \frac{R\mathbf{V}^{\perp}}{\|\mathbf{V}^{\perp}\|} \tag{9}$$

Substituting for $\mathbf{V}^{\perp}$:

$$(X_c, Y_c) = (X(t), Y(t)) + \frac{R\left[(V_X u_Y - V_Y u_X, -(V_X u_X + V_Y u_Y)\right]}{\sqrt{(V_X u_Y - V_Y u_X)^2 + (V_X u_X + V_Y u_Y)^2}}$$

Rearranging yields:

$$(X(t), Y(t)) - (X_c, Y_c) = \frac{R\left[-V_X u_Y + V_Y u_X, V_X u_X + V_Y u_Y\right]}{\sqrt{(V_X u_Y - V_Y u_X)^2 + (V_X u_X + V_Y u_Y)^2}}$$

Using to rotation matrix for finding the new position of the CM of the vehicle:

$$\begin{bmatrix} X(t+\Delta t) \\ Y(t+\Delta t) \end{bmatrix} = \begin{bmatrix} \cos\Omega_z \Delta t & \sin\Omega_z \Delta t \\ -\sin\Omega_z \Delta t & \cos\Omega_z \Delta t \end{bmatrix} \begin{bmatrix} X(t) - X_c \\ Y(t) - Y_c \end{bmatrix} + \begin{bmatrix} X_c \\ Y_c \end{bmatrix} \tag{10}$$

Solving for X(t+Δt):

$$X(t+\Delta t) = \cos\Omega_Z \Delta t (X - X_c) + \sin\Omega_Z \Delta t (Y - Y_c) + X_c \tag{11}$$

$$X(t+\Delta t) = -\cos\Omega_Z \Delta t \left[\frac{R(V_X u_Y - V_Y u_X)}{\sqrt{(V_X u_Y - V_Y u_X)^2 + (V_X u_X + V_Y u_Y)^2}} \right]$$

$$+ \sin\Omega_Z \Delta t \left[\frac{R(V_X u_X + V_Y u_Y)}{\sqrt{(V_X u_Y - V_Y u_X)^2 + (V_X u_X + V_Y u_Y)^2}} \right]$$

$$+ \frac{R(V_X u_Y - V_Y u_X)}{\sqrt{(V_X u_Y - V_Y u_X)^2 + (V_X u_X + V_Y u_Y)^2}} + X(t)$$

Note that: $R = \frac{V}{\Omega_z} \therefore \frac{R}{V} = \frac{1}{\Omega_z}$

and $\sqrt{(V_x u_y - V_y u_x)^2 + (V_x u_x + V_y u_y)^2} = V$

Therefore:

$$X(t+\Delta t) = \cos\Omega_z\Delta t \frac{(V_y u_x - V_x u_y)}{\Omega_z} + \sin\Omega_z\Delta t \frac{(V_x u_x + V_y u_y)}{\Omega_z} + \frac{(V_x u_y - V_y u_x)}{\Omega_z} + X(t)$$

$$\boxed{X(t+\Delta t) = (\cos\Omega_z\Delta t - 1)\frac{(V_y u_x - V_x u_y)}{\Omega_z} + \sin\Omega_z\Delta t \frac{(V_x u_x + V_y u_y)}{\Omega_z} + X(t)} \quad (12)$$

Similarly, solving for Y(t+Δt):

$$Y(t+\Delta t) = -\sin\Omega_z\Delta t(X - X_c) + \cos\Omega_z\Delta t(Y - Y_c) + Y_c \quad (15)$$

$$Y(t+\Delta t) = \sin\Omega_z\Delta t\left[\frac{R(V_x u_y - V_y u_x)}{\sqrt{(V_x u_y - V_y u_x)^2 + (V_x u_x + V_y u_y)^2}}\right]$$

$$+ \cos\Omega_z\Delta t\left[\frac{R(V_x u_x + V_y u_y)}{\sqrt{(V_x u_y - V_y u_x)^2 + (V_x u_x + V_y u_y)^2}}\right]$$

$$- \frac{R(V_x u_x + V_y u_y)}{\sqrt{(V_x u_y - V_y u_x)^2 + (V_x u_x + V_y u_y)^2}} + Y(t)$$

Again, note that: $R = \frac{V}{\Omega_z} \therefore \frac{R}{V} = \frac{1}{\Omega_z}$

and $\sqrt{(V_x u_y - V_y u_x)^2 + (V_x u_x + V_y u_y)^2} = V$

Therefore:

$$Y(t+\Delta t) = \sin\Omega_z\Delta t \frac{(V_x u_y - V_y u_x)}{\Omega_z} + \cos\Omega_z\Delta t \frac{(V_x u_x + V_y u_y)}{\Omega_z} - \frac{(V_x u_x + V_y u_y)}{\Omega_z} + Y(t)$$

$$Y(t+\Delta t) = \sin\Omega_z \Delta t \frac{(V_x u_y - V_y u_x)}{\Omega_z} + (\cos\Omega_z \Delta t - 1)\frac{(V_x u_x + V_y u_y)}{\Omega_z} + Y(t) \qquad (16)$$