INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

**Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600**

UMI[®]

A New Digital Differential Analyzer Approach for Inertial Technologies

by

Nelson L. P. M. Monnerat

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of
Master of Engineering

Ottawa-Carleton Institute for
Mechanical and Aerospace Engineering

Department of
Mechanical and Aerospace Engineering
Carleton University
Ottawa, Ontario
May, 2000

©Copyright
2000, Nelson Monnerat

**National Library
of Canada**

**Acquisitions and
Bibliographic Services**

**395 Wellington Street
Ottawa ON K1A 0N4
Canada**

**Bibliothèque nationale
du Canada**

**Acquisitions et
services bibliographiques**

**395, rue Wellington
Ottawa ON K1A 0N4
Canada**

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-52392-6

Canada

The undersigned recommend to
the Faculty of Graduate Studies and Research
the acceptance of the thesis

A New Digital Differential Analyzer Approach
for Inertial Technologies

submitted by

Nelson Luiz de Paula Menezes Monnerat

in partial fulfillment of the requirements for the degree of
Master Engineering

Prof. Douglas A. Staley
Thesis Supervisor

Prof. John Sinkiewicz
Thesis Co-Supervisor

Prof. Robert Bell
Chair

Department of
Mechanical and Aerospace Engineering
Carleton University

May, 2000

Abstract

This thesis presents a new approach to digitizing and implementing the navigation algorithm for the Heading and Reference Unit (HRU) under development at the Defense Research Establishment Ottawa (DREO), based on the Digital Differential Analyzers (DDA).

An inertial strapdown HRU usually needs a dynamic range of 0.01 deg/h to 400 deg/h for most applications, which corresponds to a voltage range of $1\mu\text{V}$ to 10V. For this dynamic range a 20 bit resolution with an output rate of 1Hz up to 200Hz is needed [Fejes 94]. The DDA technology is an approach to solve this requirement.

The DDA is the digital approach for the analog differential analyzer. Its basic element is the integrator, which is similar to the analog integrator, but uses counters and adders to calculate the integral of a differential equation or set of differential equations.

The use of the DDA, when implemented in hardware, allows for the implementation of the navigation algorithm to be performed in parallel .

In this thesis the self-alignment and navigation algorithms proposed by Carole Bolduc in "Self-Alignment and Navigation Algorithms for DREO Navigation Laboratory Heading Reference Unit" are implemented using the DDA approach.

Acknowledgements

My deepest thanks are extended to my supervisor Dr. Douglas A. Staley for his invaluable advice and supervision through the course of this research, and to my co-supervisor Prof. John Sinkiewicz.

I would also like to express my thanks to Michael Vinnins, Derek Hartl and Lloyd Gallop for their assistance and advice and for making this project possible.

The author also wishes to thank the Mechanical and Aerospace Engineering office staff and the Writing Tutorial Service for their assistance.

Finally, I would like to express my deepest appreciation to my wife and sons, this work would not have been completed without their endless encouragement and patience.

Thank You Lord for giving me the strength and guidance.

I dedicate this work to my wife and sons.

Contents

Abstract	i
Acknowledgments	ii
List of Figures	vi
List of Tables	viii
List of Abbreviations	x
List of Symbols	xii
1 Introduction	1
1.1 Motivation and Objective	2
1.2 Thesis Outline	3
2 HRU and The Reference Frames	6
2.1 Description of the HRU	6
2.2 Reference Coordinate Frames	7
3 Digital Differential Analyzers (DDA)	11

3.1	Digital Differential Analyzer Principles	12
3.2	Incremental Computation	13
3.3	The Integrator	15
3.4	Programming the DDA	18
3.4.1	The Integrator	18
3.4.2	Flow Diagram Example	19
3.4.3	Scaling	20
3.5	Basic Integrators	25
3.6	Error Analysis	26
3.6.1	Basic Concepts	26
3.6.2	Errors in the DDA	30
3.7	Sample Data Theory Applied to DDA	33
3.7.1	Definitions	33
3.7.2	Z-Transform	35
3.7.3	The Integrator	36
4	HRU Algorithms Implemented in DDA Form	38
4.1	HRU Alignment Scheme	39
4.2	Compensation algorithms	42
4.2.1	Acceleration compensator	44
4.2.2	Angular Velocity Compensator	47
4.3	Navigation Algorithm	54
4.3.1	Damped Spatial Rates	54
4.3.2	The Direction Cosine Matrix Q	59
4.3.3	The Direction Cosine Matrix D	62

4.3.4	True Angular Velocity	67
4.3.5	True Acceleration	70
4.3.6	Quaternions	72
4.3.7	True Velocity	75
4.4	Orthogonalization	79
4.4.1	Quaternions	80
4.4.2	Direction Cosine Matrices	82
4.5	Constraints	85
5	Software Implementation	94
5.1	Software Implementation	96
5.2	Configuration Files	98
6	Results	99
6.1	Simulation Results	100
6.1.1	Ideal Sample Data	101
6.1.2	Static Vehicle Sample Data	102
7	Conclusion	107
7.1	Future Research	108
A	Analog to Digital Converters	113
B	Description of the Simulation Software Files	129
C	Configuration Files	158

List of Figures

1.1	Inertial navigation system (INS) diagram	2
2.1	Reference coordinate frames	8
3.1	Digital Integrator with respect to time	16
3.2	Digital integrator with incremental output	17
3.3	The integrator	19
3.4	Second-order differential equation	20
3.5	DDA multiplier	25
3.6	DDA square implementation	26
4.1	Damped Platform Alignment Model for the North-Axis Accelerometer .	41
4.2	DDA implementation of the acceleration compensator	45
4.3	DDA Implementation of the angular velocity compensator	50
4.4	DDA implementation of damped angular rates	56
4.5	DDA implementation of the DCM Q terms	61
4.6	DDA implementation of the DCM D	65
4.7	DDA implementation of the true angular velocity	68
4.8	DDA implementation of the true acceleration	71

4.9	Simple DDA integrators	73
4.10	DDA implementation of the quaternions	74
4.11	DDA Implementation of the true velocity	76
4.12	DDA implementation of quaternion's orthogonalization	81
4.13	DDA Implementation of the error term of the orthogonalization of the DCM elements	84
6.1	Results of the first simulation	104
6.2	Results of the simulation with HRU's sample file	106
A.1	Charge reset V/F converter [Hoeshele 94].	123
A.2	Charge balancing V/F converter [Hoeshele 94].	125

List of Tables

3.1	Numerical example of digital integration	16
3.2	Scaling table for the second-order differential equation	23
3.3	Common Z-Transforms.	35
3.4	Z-Transform Properties	36
4.1	Acceleration compensator integrator scale factors.	46
4.2	Parameters for the angular velocity integrators	51
4.3	Angular velocity compensator integrator scale factors	53
4.4	Damped spatial rates integrators' parameters	59
4.5	Integrator scale factors referent to the $\mathbf{Q}$ matrix diagonal elements . .	63
4.6	Integrator scale factors referent to the $\mathbf{Q}$ matrix non-diagonal elements	63
4.7	Integrator scale factors of the DDA implementation of the DCM $\mathbf{D}$. .	66
4.8	Integrator scale factors of the DDA implementation of the true angular velocity	69
4.9	Integrator scale factors of the DDA implementation of the true acceler- ation	71
4.10	Integrator scale factors of the DDA implementation of the quaternions	73
4.11	DDA integrator scale factors of the true velocity	77

4.12 DDA integrator scale factors of the orthogonalization of the quaternions	81
4.13 DDA integrator scale factors of the orthogonalization of the DCM terms	85
4.14 Table of algorithm constraints	86
4.15 Relations between the size of the registers and the supremum of registers	92
6.1 Configuration file <code>hru.cfg</code>	103

List of Abbreviations

A/D	Analog to Digital Converter
ANSI	American National Standards Institute
BLT	Extention of the Tk toolkit
CEP	Circular Error Probable
CSG-2	Canadian Strapdown Gyroscope
D/A	Digital to Analog Converter
DCM	Direction Cosine Matrix
DDA	Digital Differential Analyzer
DREO	Defense Research Establishment Ottawa
EPLD	Erasable Programmable Logic Device
FPGA	Field-Programmable Gate Array
GMT	Greenwich Mean Time
GNU	A recursive acronym for “GNU’s Not Unix”
GPS	Global Positioning System
GUI	Graphical User Interface
HRU	Heading Reference Unit
INS	Inertial Navigation System
ISA	Inertial Sensor Assembly

LIST OF ABBREVIATIONS

LPF	Low Pass Filter
MSB	Most Significant Bit
PCM	Pulse Code Modulator
PMW	Python Megawidgets
RMS	Root Mean Square
RTD	Resistance Thermal Device
ROC	Region of Convergence
SBC	Single Board Computer
S/H	Sample and Hold circuit
SPU	Signal Processing Unit
SQNR	Signal-to-Quantization Noise Ratio
TINORU	The Inertial Navigation Organized Research Unit

List of Symbols

a_{max}	Supremum of compensated acceleration
a_{ir}	Raw data from the accelerometers for the “i” axis
a_{rmax}	Supremum value of a_r
a_i	Compensated acceleration for the “i” axis
a_{Gi}	Acceleration along the gyroscope axis “i”
A_{ij}	Accelerometer misalignment angles, the first subscript refers to the body axis and the second refers to the system axis
AB_i	Accelerometer bias drift for the “i” axis
A_c	True acceleration in computational frame of reference
AK_i	Accelerometers scale factor, where $i = \{x, y, z\}$
C	DCM that transforms from system frame to computational frame as function of Eüler angles
D	DCM that transforms from earth-centered earth-fixed frame to computational frame
$e_q(n)$	Quantization error
E, N, U	Local Geographical Frame (East, North, Up)
g_m	Gravity acceleration, $9.806m/s^2$
G	Geographic Align Matrix

G_{ij}	Gyroscope misalignment angles, the first subscript refers to the body axis and the second refers to the system axis
GB_i	Gyroscope bias drift for the “i” axis
GK_i	Gyroscope scale factors, where $i = \{x, y, z\}$
I	Integral register
i_n	is the input of the Y register
K_{ij}	Terms defined in section 4.2.2 for the angular velocity compensation
K_a	Acceleration compensator integrator scale factor
K_{Ac}	True acceleration integrator scale factor
K_D	DCM D terms integrator scale factor
K_q	Quaternions integrator scale factor
K_Q	DCM Q terms integrator scale factor
K_{Vc}	True velocity integrator scale factor
K_μ	True angular velocity integrator scale factor
K_ρ	Damped angular rate integrator scale factor
K_ω	Angular velocity compensator integrator scale factor
L	Number of quantization levels
MU_i	Gyroscope mass unbalance drift term for the “i” axis
M_{Ac}	Supremum of true acceleration
M_D	Supremum of DCM D terms
M_q	Supremum of quaternions
M_Q	Supremum of DCM Q
M_{Vc}	Supremum of true velocity
M_μ	Supremum of true angular

M_ρ	Supremum of damped angular rate
M	Supremum of the integrand variable y
M_k	Supremum of the AK terms
M_{kw}	Supremum of the K_{ij} terms
M_{ka}	Supremum of the GK terms
n	Number of bits in the integrator registers
N	Integrator register capacity
N_a	Acceleration compensator integrator register capacity
N_{Ac}	True acceleration integrator register capacity
N_D	DCM D terms integrator register capacity
N_q	Quaternions integrator register capacity
N_Q	DCM Q terms integrator register capacity
N_{Vc}	True velocity integrator register capacity
N_μ	True angular velocity integrator register capacity
N_ρ	Damped angular rate integrator register capacity
N_w	Angular velocity compensator integrator register capacity
o_n	is the output of the integral
P_q	Mean square error power
q	Quaternions
Q	DCM that transforms from system frame to computational frame as function of quaternions
Q_i	Gyroscope quadrature drift term for the “ i ” axis
r	Radix of the DDA operator
R	Integral Register

R_o	Earth's radius
τ_n	the round-off
R_g	A/D range
V_c	True velocity in the computational frame of reference
V_R	Reference voltage
x_a	Analog signal
x_q	Quantized value of x
x	Analog variable of interest
X_i, Y_i, Z_i	Earth-centered inertial frame
X_e, Y_e, Z_e	Earth-centered earth-fixed frame
X_c, Y_c, Z_c	Local-level wander-azimuth frame
X, Y, Z	Integrators variables
Y^*, R^*, I^*	Corresponding integrator's registers updated
α	Wander angle
δx	Increment
Δ	Quantization level
ϕ_l	Latitude
λ	Longitude
μ_c	True angular velocity in the computational frame of reference
ρ	Damped rate in the computational frame of reference
$\hat{\rho}$	Angular rate of the computational frame of reference with respect to the earth-centered earth-fixed frame of reference
σ_x	Standard deviation of the signal x
τ	Damping parameter

τ_q	Interval where x_a is between quantization levels
ω	Compensated angular velocity
ω_{max}	Supremum of compensated angular velocity
ω_r	Raw data from the gyroscopes
Ω_{ie}	Earth angular ratio in the earth-centered inertial frame of reference
Ω_c	Earth angular ratio in the computational frame of reference

Chapter 1

Introduction

Inertial navigation and guidance systems are applied to determine the physical position and velocity of aircrafts and ships relative to a reference frame, such as their geographical position. They are also used to guide the vehicle from one point to another. The advantage of the inertial navigation over radar, sonar, or GPS (Global Positioning System) is that it is completely independent of external information or influences such as weather, magnetic disturbances, electronic jamming and distortion.

Modern inertial navigation and guidance systems consist of an inertial sensor assembly, a signal processing unit and navigation algorithms. A typical inertial sensor assembly consists of three linear accelerometers and their associated electronics, and either three single-degree-of-freedom gyroscopes, or two 2-degree-of-freedom gyroscopes.

An inertial navigation system is called gimbaled when it uses a stable platform with three or more gimbals, which is kept either nonrotating with respect to an inertial frame, or establishes a reference frame which precesses from the inertial frame at a known rate [Siouris 93].

A strapdown inertial navigation system differs from a gimballed system in that the inertial sensors are mounted directly onto the vehicle frame instead of on a stable platform. The navigation computer keeps track of the vehicle's attitude, with respect to some preselected reference frame, based on information from the inertial sensors.

The gyroscopes' output is used to either drive the gimbal torque motors in such a way as to maintain the platform in alignment with the inertial coordinate frame in a gimballed system, or to keep the computational frame oriented in a strapdown system. The output of the accelerometers is transformed to the computational frame. Performing computations through the navigation algorithm generates the navigation information .

1.1 Motivation and Objective

In a very simple schematic approach, an inertial navigation system can be represented by the diagram in figure 1.1.

Figure 1.1: Simple schematic diagram of an inertial navigation system (INS).

The interface to the sensors is digital, so the information can then be directly applied by other systems that are integrated to the vehicle.

An inertial navigation system, such as the strapdown Heading Reference Unit (HRU),

usually needs a dynamic range of 0.01 deg/h to 400 deg/h for most applications, which corresponds to a voltage range of $1\mu\text{V}$ to 10V. For this dynamic range a 20 bit resolution with an output rate of 1Hz up to 200Hz is needed [Fejes 94].

Different techniques have been applied to digitize the sensor signals. Reference [Fejes 94] presents an approach for digitizing the signal from the HRU from the Defense Research Establishment Ottawa (DREO) using Delta-Sigma analog-to-digital converters (A/D) with resolution of 21 bits.

The A/D presented in [Fejes 94] was used in reference [Bolduc 95], where the navigation algorithm for the Heading Reference Unit (HRU) from DREO is presented.

As presented in [Hartal 97] and [Hartl-1 97], the stability of the least significant bits caused the reduction of the resolution of the A/D to 17 bits, and motivated the design of new A/D boards and configurations to be implemented in the HRU using new A/D converters.

The objective of this thesis is to improve the precision and the frequency response of the HRU system using the Digital Differential Analyzers (DDA).

The algorithm development and testing has been done by Carole Bolduc and is very well presented in her work [Bolduc 95]. In this work, her results will be used as a reference for comparison, since the HRU hardware implementation does not allow the DDA implementation and testing, which would require frequencies around 500 kHz.

1.2 Thesis Outline

This work has been organized as follows:

Chapter 1 Introduction, where a very brief overview of this work is presented.

- Chapter 2 The frames of reference used by the navigation algorithm are presented. Also the HRU system is briefly described.
- Chapter 3 In this chapter the DDA is presented, its basic concepts and principles, and how they interact with each other. Some basic building blocks that are used to implement the navigation algorithm are also presented.
- Chapter 4 In this chapter the navigation algorithm developed by Carole Bolduc is implemented in a DDA form. Each section corresponds to a part of the algorithm, beginning with a description of the algorithm and then their DDA implementation. The final section of this chapter presents the relations of each part of the algorithm and a more general description of the parameters is deduced. These relations are implemented in the simulation software as constraints or to find the parameter values.
- Chapter 5 An explanation of how the simulation software has been built is presented in this chapter, with a description of each file and classes that implement the algorithm components. Since the software has around 10,000 lines, the source is not part of this work, but only the description of the classes' implementations.
- Chapter 6 The results achieved from the same data files used by Carole Bolduc in her work are presented in this chapter. This has been done in order to allow a comparison between these two works, since the DDA was not implemented in hardware.
- Chapter 7 This is the final chapter where the conclusions are presented with proposals

for future work.

Chapter 2

Description of The HRU and The Reference Frames of Coordinates

In this Chapter the description of the Heading Reference Unit (HRU) from DREO will be presented. The parameters used in this work are from this system. The description that follows is based on the reports in references [Hartal 97] and [Hartl-1 97].

Also a description of the reference frames of coordinates used through out this work will be presented.

2.1 Description of the HRU

The HRU is designed to be a stand-alone strapdown attitude and heading reference unit with full navigation capabilities. The intended accuracy of the system is to be of 0.1 degree RMS in roll, pitch and yaw, and less than one nautical mile per hour Circular Error Probable (CEP) in position [Hartl-1 97].

CEP is a concept derived from the accuracy of the delivery of a weapon such as bombs or missiles. It is defined as the radius of a circle centered at the mean point of impact that encloses 50% of the probability of a hit. Since CEP has the error distribution information in its formulation, it can be stated that the smaller the CEP, the smaller the errors, and the more accurate the system. More detailed information about the CEP and its applications can be found in reference [Siouris 93].

The HRU is comprised of two separate hardware assemblies, the Inertial Sensor Assembly (ISA) and the Signal Processing Unit (SPU). The ISA is a six inch cube in which three Q-Flex QA-700 pendulous accelerometers and two CSG-2 dynamically tuned two-degree-of-freedom gyroscopes are mounted. The block is thermally controlled and monitored using cartridge heaters and resistance thermal detectors (RTD) mounted on the block. The ISA is connected to the SPU via shielded cables.

The SPU is a standard rack-mountable 19 inch cardcage which houses all the signal processing, supply and control electronics. Signals from the sensors are digitized in the SPU. The 486 single board computer (SBC) reads data over the backplane bus.

2.2 Reference Coordinate Frames

There are several different reference coordinate frames available to the inertial navigation system designer for use in navigation computation of position, velocity and attitude. The reference frames presented in this section are the ones used in Carole Bolduc's work [Bolduc 95] which follows the same approach as [Britting 71] and [Siouris 93].

Geographic Frame (n) the *geographic frame* is a local navigational frame (also called *navigation frame*) that has its origin at the HRU location and its axes can be

Figure 2.1: Reference coordinate frames used through out this work.

- X_i, Y_i, Z_i earth-centered inertial frame
- X_e, Y_e, Z_e earth-centered earth-fixed frame
- E, N, U local geographic frame
- X_c, Y_c, Z_c local-level wander-azimuth frame

defined either as North, East and Down or East, North and Up directions. The Up (U) and Down (D) axes are defined as normals to the reference ellipsoid. The North axis (N) is in the direction of the projection of earth's inertial angular velocity vector into the local horizontal plane. The East direction completes the right-handed orthogonal set. This is a local-level frame with its x and y axes in a plane tangent to the reference ellipsoid. The convention adopted in this work is East=X, North=Y and Up=Z.

Body Frame (b) the *body frame* (also called *system frame*) consists of the roll, pitch and yaw axes of the vehicle with its origin at the center of mass of the vehicle. In general, the vehicle's center of mass does not coincide with the location of the Attitude and Heading Reference System. It is a convenient coordinate system for developing the equations of motion of a vehicle, since the vehicle's motion equations are normally written in this frame of reference. The relations between the sensors, gyroscopes and accelerometers, frames of reference and the body frame must be determined for each specific application.

Local-Level Wander-Azimuth Frame (c) the *local-level wander-azimuth frame* is also referred to as the *computational* or *navigational* frame. Its origin coincides with the geographic frame. The horizontal axes lie in a plane tangent to the local vertical. Eulerian rotations of longitude, geodetic latitude and wander angle are used to relate the local-level wander-azimuth frame to the earth-centered earth-fixed frame. The latitude (ϕ_l) rotation is defined to be positive in the northern hemisphere, the wander angle (α) is defined positive west of true north and measured in the geodetic horizontal plane, and the longitude (λ) is defined

positive east of the Greenwich meridian and is measured in the equatorial plane. When α the wander angle is equal to zero the local-level wander azimuth frame is coincident with the geographic frame.

Earth-Centered Earth-Fixed Frame (e) the *earth-centered earth-fixed frame* has its origin at the earth's center of mass. This frame rotates with the earth. The Y_e axis is directed North along the earth's spin axis, while the X_e and Z_e axis are in the equatorial plane with X directed through the Greenwich Meridian (0° latitude, 0° longitude), and the Y axis is directed 90° east longitude.

Earth-Centered Inertial Frame (i) the *earth-centered inertial frame* has its origin at the earth's center of mass and is non-rotating relative to the inertial space (i.e. the fixed stars). However, this frame accelerates with respect to inertial space since it moves with the earth. It coincides with the earth-centered earth-fixed frame once every 24 hours. The X_i and Z_i axes lie in the earth's equatorial plane, the Y_i axis is aligned with the earth's spin axis.

Chapter 3

Digital Differential Analyzers (DDA)

The Digital Differential Analyzer (DDA) was applied in the 1960's for real time operation, combining the speed of analog machines with the accuracy and flexibility of digital ones. It has also been applied as a solution for navigation and attitude control in some Inertial Navigation Systems. However, large increases in the computing speed and the considerable reduction in size of microprocessors led to the relegation of the DDA to historical interest only.

In order to understand the DDA, some concepts related to analogue computers need to be addressed. Generally, analogue computers fall into two categories, differential analyzers and simulators. The differential analyzer solves a particular equation or set of equations within a range of coefficients. The simulator, on the other hand, is a detailed working model of a physical system, which enables its behavior to be examined under a wide range of conditions.

The DDA is the digital approach for the analog differential analyzer. Its basic element is the integrator, which is similar to the analogue integrator, but uses counters

and adders to calculate the integral, as will be shown in section 3.1.

Nowadays, one can find only a few DDA applications where incremental integration with high accuracy and real time response is needed, such as numerical control of machine solutions for interpolators [Chai 94], and inverse solutions for robot kinematics using incremental integration through a DDA [Park 94].

3.1 Digital Differential Analyzer Principles

A DDA consists of a set of integrators mechanized by a digital process. The DDA integrator is an incremental device with its principle of operation based on the hypothesis that physical systems change only by a small amount in a small interval of time [Sizer 68].

Following the approach in [Sizer 68], to present the basic principle of the DDA, let the increment δX always be small, then

$$\sum Y.\delta X$$

will be a close approximation to

$$\int Y.dX$$

so the integral can be approximated as a successive sum of small increments.

In order to allow the DDA to be connected to other integrators, it is arranged so that the output ΔZ of any integrator may be used as the incremental input, either as ΔX or ΔY , of any other. With the adder approximation of an integral as shown above,

the result of the addition will be

$$\begin{aligned} Y^* &= Y + \sum \Delta Y \\ \Delta Z &= \frac{1}{r^n} \cdot Y^* \cdot \Delta X \\ Z &= \frac{1}{r^n} \cdot \int Y \cdot dX \end{aligned}$$

where n is the number of bits in the Y register, and r is the radix of the numbers on which the DDA operates, usually $r = 2$. In section 3.3 it will be described how it is implemented in such a way that the output of the integrator is in increment form using correct scaling.

3.2 Incremental Computation

An important property of physical systems is that they behave in a continuous manner, which means that values change a small amount in a small interval of time. So if the state of the system is known in one instant of time, it is only necessary to calculate a small change, which occurs during an iteration period, to obtain the state at the next instant. The time frequency in which it is calculated should be high enough to satisfy the Nyquist condition avoiding alias.

With the approach usually used for digital computers, a succession of discrete values is calculated in each iteration. To illustrate the principle of incremental computation, an example from [Sizer 68] will be shown.

Suppose that the value of the product of the variables X and Y is known at a given

instant of time

$$Z = X.Y \quad (3.1)$$

To calculate the value after one iteration, where $X^* = X + \delta X$ and $Y^* = Y + \delta Y$, the new product value will be

$$(X + \delta X) . (Y + \delta Y) = X.Y + X.\delta Y + Y.\delta X + \delta X.\delta Y \quad (3.2)$$

Assuming that the values of the increments are sufficiently small that the second order term can be neglected, the product change can be approximated to

$$\delta(X.Y) \approx X.\delta Y + Y.\delta X \quad (3.3)$$

and this is the increment that should be calculated.

At first sight it seems that, instead of one multiplication, now there are two multiplications and one addition. However, if by suitable scaling and sufficiently shortening the iteration period, each of the variables never changes by more than one bit, or in other words, δX and δY are limited to values $\frac{1}{2^n}$, 0 and $-\frac{1}{2^n}$, where n defines the size of the digital word. Then the multiplication is reduced to merely adding or subtracting X and Y at a suitable level of significance.

3.3 The Integrator

One of the fundamental premises for the DDA integrator is that the increment δX should always be small, in order for the expression $\sum Y.\delta X$ to be a good approximation of $\int Y.dX$.

To show the principle of the DDA integrator, let δX be constant and the smallest representable digit, that is let δX be one bit. During one iteration period $Y.\delta X$ is the area of the corresponding rectangle, and because δX is one, the area is equal to Y .

With these premises, an integrator as presented in figure 3.1 can be built. It consists of two digital registers, called Y and I . The Y register holds the current value of the integrand and the I register holds the current value of the integral. During each iteration two operations are made:

- The value of the integrand Y is updated to Y^* by adding the increment of Y

$$Y^* = Y + \delta Y \quad (3.4)$$

- The current value of $Y^*.\delta X$ is added to I to form the current value of the integral I^*

$$I^* = I + Y^*.\delta X \quad (3.5)$$

However we have assumed that δX is equal to 1, in the more general case, using a correct scale¹, δX can assume the values $\{-1, 0, +1\}$. The rules for the integration process will then become:

¹See section 3.4.3 for more details.

Figure 3.1: Diagram of the digital integrator with respect to time.

- Add Y if δX is 1;
- Subtract Y if δX is -1 ;
- Do nothing if δX is 0.

Y	I	R		Integral
4	4	4		0
5	9	9		0
6	1 5	5	Overflow	1
6	2 1	1	Overflow	2
7	2 8	8		2
8	3 6	6	Overflow	3
8	4 4	4	Overflow	4
8	5 2	2	Overflow	5
8	6 0	0	Overflow	6

Table 3.1: Numerical example of how a digital integrator register R is implemented in order to generate outputs in a incremental form.

In order to allow the integrator to be interconnected to others integrators, the integral should be modified to a incremental form, such as its inputs, instead of a whole

number form in the register I . To show how it is done [Sizer 68] presents an example through the table 3.1

Figure 3.2: Digital integrator with incremental output generated as an overflow of the integral register R .

It could be noted that, while the integrand Y consists only of a single integer, the integral builds up to an integer with double the size of Y . Thus the part of I of interest is the most significant half, and this represents the value of the integral to some scale. So the digital integrator register I should be divided into two halves as shown in figure 3.2.

The final form of the integrator is a device consisting of two equal length registers, Y and R . The value of Y is kept according to $Y^* = Y + \delta Y$ at each iteration. The current value of Y is multiplied by the increment δX , which can take the values $\{-1, 0, 1\}$, and is then added to the register R . This causes the register R to overflow periodically. The overflow represents the increments of the integral as shown by the equation

$$R + Y^* \cdot \delta X = R^* + \delta Z \quad (3.6)$$

Now the interconnection of integrators is made possible and it could be used to solve problems in the same way as analogue computers.

There are two main ways in which DDA may be organized. There is the type in which each integrator is a self-contained unit and the integration process takes place simultaneously in all integrators. This type of DDA is known as 'simultaneous'.

In the other type of DDA organization the Y and R values are kept in memory registers and read out in turn to undergo the integration process in a common arithmetic unit. This is known as the sequential type of DDA.

In [Sizer 68] there are some examples and a more detailed explanation of these implementations.

3.4 Programming the DDA

3.4.1 The Integrator

The principles of the integrator have been described in section 3.1. It will now be treated from the point of view of programming, consisting of flow charting and scaling.

The notation that will be used has been proposed in [Sizer 68] as shown in figure 3.3 (a). The inputs are one ΔX point, dx , and a ΔY point capable of accepting $dy_1, dy_2, dy_3, \dots$ increments. There is a ΔZ output providing several equivalent outputs dz . Although several dz outputs and dy inputs are shown, usually in flow charts it will appear as in figure 3.3 (b) with only one output and only one input. Note that the small circle means that the output is changed from $+dz$ to $-dz$.

Each integrator is defined by a unique number, j , in the diagram.

Figure 3.3: The integrator representation.

3.4.2 Flow Diagram Example

A problem is presented for solution using DDA in the form of either one *differential equation* or, possibly, several simultaneous *differential equations*.

The input to the register Y of an integrator represents the derivative of the quantity in that register. For example, consider the second-order differential equation

$$\frac{d^2y}{dx^2} + A\frac{dy}{dx} + By = 0. \quad (3.7)$$

If the solution is required for y and $\frac{dy}{dx}$, they must each be held as the Y register of an integrator. The input to these two integrators are the respective derivatives of y and $\frac{dy}{dx}$. Re-writing the original equation it will be

$$\frac{d}{dx} \left(\frac{dy}{dx} \right) = -A\frac{dy}{dx} - By \quad (3.8)$$

multiplying the equation by dx gives

$$d \left(\frac{dy}{dx} \right) = -A dy - B y dx \quad (3.9)$$

which is the equation linking dy and $d \left(\frac{dy}{dx} \right)$.

Based on equation 3.9 the flow diagram in figure 3.4 can be built.

Figure 3.4: DDA model of a second-order differential equation.

3.4.3 Scaling

Integrator Scale Factors

In order to use the variables x , y and z in a DDA integrator, each one should be represented by X , Y and Z , where each increment corresponds to one bit. The following relations should also be true:

$$\begin{aligned} x &= K_x \cdot X \\ y &= K_y \cdot Y \\ z &= K_z \cdot Z \end{aligned} \tag{3.10}$$

in such a way that K_x , K_y and K_z represent the change in x , y and z produced by one increment. K_x , K_y and K_z are known as the *scale constants* or *integrator scale factors*.

Suppose an integrator with variables dx and dy for ΔX and ΔY , respectively, and

let dz be such that

$$dz = y \cdot dx \quad (3.11)$$

so y is obtained by integrating dy into the register Y .

If the capacity of the integrator is N , the basic integrator equation will be

$$dz = \frac{y \cdot dx}{N} \quad (3.12)$$

taking the scale factors K_x , K_y and K_z the equation (3.11) will be

$$\frac{dz}{K_z} = \frac{y \cdot dx}{K_y \cdot K_x \cdot N} \quad (3.13)$$

Using equation 3.12 with 3.13, the relation between the scaling constants can be found as

$$K_z = K_x \cdot K_y \cdot N \quad (3.14)$$

Now let M be the supremum² of y , and let y_m be the maximum value the problem variable y can have, then

$$\left. \begin{array}{l} y = K_y Y \\ y_m \leq M \end{array} \right| \Rightarrow \frac{M}{K_y} = N \quad (3.15)$$

²Let G be the set of possible values of the variable y , a number M is called the *last upper bound* (*l.u.b*) or the *supremum* (*sup*) of G if $y \leq M$ for all $y \in G$ [Friedman 71].

from equation 3.14

$$K_z = M.K_x \quad (3.16)$$

Scaling Table

When the configuration is simple, like an integrator loops back ΔZ as ΔY , the solution to equation 3.14 will give the scale constant values. Usually there is insufficient information to solve equation 3.14 alone and equations 3.15 and 3.16 should be used to solve for each integrator. In order to solve these equations reference [Sizer 68] proposes a convenient way of constructing a table in conjunction with the flow diagram. This table is called the *integrator scaling table*.

To construct the integrator scaling table, the integrators are numbered, and these numbers are transferred to the table, with a line corresponding to each integrator. The table should have the following columns:

- The integrator numbers (j);
- The quantity (in problem variables) held in the register Y ;
- The number of the integrator (i) whose output is the source of (j)'s ΔX input, or ΔT if the ΔX source is machine time, or the quantity in problem variables;
- The numbers of the integrators whose outputs are the sources of ΔY inputs to (j), or the quantity in problem variables;
- The maximum value that y can have in problem variable form y_m ;
- The maximum value calculated for y in machine variable form M ;

- The scale constant on the ΔX input K_x ;
- The scale constant on the ΔZ input K_z ;
- The scale constant on the ΔY input K_y ;
- The register length of the integrator N .

When all the columns have been filled the scaling is complete. It may be necessary to insert arbitrary values of K_x , K_y , or K_z if there is insufficient knowledge about the problem, and from there work out N and M for each integrator.

Using the second-order differential equation represented by the flow diagram in figure 3.4, assuming that $A = 0.271$ and $B = 12.3$, $0 \leq x < 8$ and y is represented to one part in 1,000, the corresponding scaling table is presented in table 3.2.

Integrator number	Y register	ΔX	ΔY	y_m	M	K_x	K_z	K_y	N
1	$\frac{dy}{dz}$	ΔT	3, 4	-	2^2	2^{-12}	2^{-10}	2^8	2^{10}
2	y	ΔT	1	-	2^0	2^{-12}	2^{-12}	2^{-10}	2^{10}
3	0.271 (A)	1	-	0.271	2^2	2^{-10}	2^{-8}	2^{2-n}	2^n
4	12.3 (B)	2	-	12.3	2^4	2^{-12}	2^{-8}	2^{4-n}	2^n

Table 3.2: Scaling table for the second-order differential equation modeled in figure 3.4.

To fill the table 3.2:

1. Fill the left column with integrator numbers in order.
2. Fill the next column, "Y register", with the respective contents described in problem variables.
3. Fill the columns " ΔX " and " ΔY " referring to the flow diagram.
4. The y_m may be inserted if known.

5. The N value may be inserted, for our example of second-order differential equation with one part in 1,000 is chosen $N = 1024 = 2^{10}$. The other values should be the maximum capacity of the register, in the example 2^n .
6. Consider each integrator applying the equation 3.14, in the example:

$$K_{z1} = K_{x1}K_{y1}2^{10} \quad (3.17)$$

$$K_{z2} = K_{x2}K_{y2}2^{10} \quad (3.18)$$

7. Consider each integrator applying the equation 3.16, in the example:

$$K_{z3} = K_{x3}M_3 \quad (3.19)$$

$$K_{z4} = K_{x4}M_4 \quad (3.20)$$

8. Look from the flow diagram the relations between the integrators. In figure 3.4 the output of (1) is the ΔY input of (2) and the ΔX input of (3), which means that:

$$K_{z1} = K_{y2} = K_{x3} \quad (3.21)$$

similarly:

$$K_{z2} = K_{x4} \quad (3.22)$$

$$K_{y1} = K_{z3} = K_{z4} \quad (3.23)$$

From these relations and the equations 3.14 and 3.16, the scale constants are found and the corresponding columns filled.

In the following integrator scaling tables of the navigation algorithm implementation, for the sake of simplicity, the y_m column will be suppressed.

Reference [Sizer 68] shows simple sub-programs for *exponential loop*, *sine-cosine loop*, *arcsine* and *arccosine*, *product*, *square*, *square-root*, *quotient* and *adders*. Also, other forms of implementing *servo elements*, integrators as *decision units* and as *store elements* are presented.

The following section will present simple integrator implementations that will be used through out this work.

3.5 Basic Integrators

This section presents basic integrators that are used as building blocks for the implementation of the navigation algorithm in chapter 4.

Multiplier

Let $v = g.h$ so the differential of v is $\delta v = \delta g.h + g.\delta h$. In a DDA form this can be implemented as in figure 3.5.

Figure 3.5: This is the DDA implementation of the differential of the multiplication of two function.

Square

Let $v = u^2$ so the differential of v is $\delta v = 2.u.\delta u$. In a DDA form this can be implemented as in figure 3.6 except for the 2 multiplier constant, so figure 3.6 actually implements $\frac{1}{2}\delta v = u.\delta u$.

Figure 3.6: This is the DDA implementation of the differential of the square of a function.

3.6 Error Analysis

This section introduces some basic concepts that are applicable to any process of A/D conversion and later some of these concepts are applied to the DDA implementations.

3.6.1 Basic Concepts

Quantization Error

The general process of A/D conversion is basically a three step process consisting of *Sampling*, *Quantization* and *Coding*. If the sampling theorem is respected, in other words, the maximum frequency of the analog system is less than the half of the sampling frequency, then the *sampling* process is reversible without loss of information. The *quantization* process is irreversible because an error is introduced. The *quantization*

error or quantization noise

$$e_q(n) = x_q(n) - x(n) \quad (3.24)$$

where $e_q(n)$ is the *quantization error*, $x_q(n)$ is the quantized value of x , and $x(n)$ is the sampled value of the analog variable of interest.

If x_{min} and x_{max} represent the minimum and maximum values of $x(n)$, and L is the number of quantization levels, then the *quantization level* Δ can be define as

$$\Delta = \frac{x_{max} - x_{min}}{L - 1} \quad (3.25)$$

There are basically two methods to *discard* the excess digits from the sampled value, which theoretically has infinite precision, *truncation* where the excess digits are discarded, and *rounding* where the number is rounded to the nearest value, which means that $|x(n) - x_q(n)| \leq \frac{\Delta}{2}$.

From the definition of Δ the quantization level, the range of the quantization error in rounding is

$$-\frac{\Delta}{2} \leq e_q(n) \leq \frac{\Delta}{2} \quad (3.26)$$

Signal-to-Quantization Noise Ratio (SQNR)

The quality of the output of an A/D converter is usually measured by the *signal-to-quantization noise ratio* (SQNR), which is the ratio between the *signal power* and the *noise power*. To evaluate this parameter, let the signal be a sinusoid such as

$x_a = A \cos \Omega_0 t$. If the sampling theorem is satisfied, the only source of error is the quantization error. To evaluate the quantization error the analog signal $x_a(t)$ is quantized instead of the discrete-time signal $x(n) = x_a(nT)$.

Suppose the quantization intervals are small enough to approximate $x_a(t)$ as a linear function between two quantization levels, so the quantization error will be:

$$e_q(t) = x_a(t) - x_q(t) \quad (3.27)$$

where $e_q(t)$ is the quantization error and $x_q(t)$ is the quantized value of $x_a(t)$. Let τ_q be the interval $x_a(t)$ stays between two quantization levels Δ .

Since the quantization process is rounding, the quantization error is:

$$e_q(t) = \frac{\Delta}{2\tau_q}t, \quad -\tau_q \leq t \leq \tau_q \quad (3.28)$$

The mean-square error power P_q is:

$$P_q = \frac{1}{2\tau_q} \cdot \int_{-\tau_q}^{\tau_q} e_q^2(t) \cdot dt = \frac{1}{\tau_q} \cdot \int_0^{\tau_q} e_q^2(t) \cdot dt \quad (3.29)$$

With equations 3.28 and 3.29:

$$P_q = \frac{1}{\tau_q} \cdot \int_0^{\tau_q} \left(\frac{\Delta}{2\tau_q} \right)^2 \cdot t^2 \cdot dt = \frac{\Delta^2}{12} \quad (3.30)$$

If the quantizer has b bits of accuracy and the quantizer covers the entire range $2A$,

the quantization step is $\Delta = \frac{2A}{2^b}$. Hence:

$$P_q = \frac{\frac{A^2}{3}}{2^{2b}} \quad (3.31)$$

The average power of $x_a(t)$ is

$$P_a = \frac{1}{T} \int_0^T (A \cos \Omega_0 t)^2 dt = \frac{A^2}{2} \quad (3.32)$$

From 3.31 and 3.32

$$\text{SQNR} = \frac{P_a}{P_r} = \frac{3}{2} \cdot 2^{2b} \quad (3.33)$$

Expressed in decibels (dB), the SQNR is

$$\text{SQNR} = 10 \log_{10} \text{SQNR} = 1.76 + 6.02b \quad (3.34)$$

Equation 3.34 implies that the SQNR increases approximately 6 dB for every bit added to the word length, that is, for each doubling of the quantization levels [Proakis 96].

For a generic signal, if the quantization error is uniformly distributed in the range $(-\frac{\Delta}{2}, \frac{\Delta}{2})$ the mean value of the error is zero. In reference [Proakis 96] it is shown that the variance in dB is:

$$\text{SQNR} = 6.02b + 16.81 - 20 \log \frac{R_g}{\sigma_x} \text{ dB} \quad (3.35)$$

where R_g is the range of the A/D converter and σ_x is the standard deviation of the

signal. The equation 3.35 is used to specify the precision needed in an A/D converter.

Due to limitations in the fabrication of A/D converters, the performance falls short of the theoretical value given by equation 3.35. The effective number of bits may be somewhat less than the number of bits in the A/D converter.

3.6.2 Errors in the DDA

The first error to be considered is the effect of the conversion from analog to digital representation of the input signal. Because of the finite step size of the converter, the digital number which is in Y_n , where the index n means the n th iteration of the integrator, differs from the analog variable y_n by e_{qn} , the round-off. This may be represented by the equation

$$Y_n = y_n - e_{qn} \quad (3.36)$$

As described in section 3.3, after n steps of integrations, at the beginning of the $(n+1)$ th step, the converter is sampled and its output Y_n is inserted into the Y register of the first integrator. Y_n is then added to R_n , the value in register R , and changes it to R_{n+1} . An overflow δZ is then generated and added to the integrand of the second integrator, and changes its contents from Z_n to Z_{n+1} .

To find out the relation between the overflow δZ and the values in the registers Y and R in the first integrator it is necessary to define the capacity of the R register. Let M units be the capacity of R , which means that if M units are added to the register R an overflow is produced. When Y_n is added into the R register three possibilities may occur:

1. zero overflow, in which case: $\delta Z = 0$ and $R_{n+1} = R_n + Y_n$
2. positive overflow, in which case: $\delta Z = 1$ and $R_{n+1} = R_n + Y_n - M$
3. negative overflow, in which case: $\delta Z = -1$ and $R_{n+1} = R_n + Y_n + M$

Reference [Sizer 68] presents a formula where these three possibilities are combined

$$\delta Z = \frac{1}{M} (Y_n + R_n - R_{n+1}) \quad (3.37)$$

Summing the equation 3.37 from $n = 0$ to $n = N - 1$, where N is the total number of steps, gives

$$Z_n - Z_0 = \frac{1}{M} (Y_0 + Y_1 + \cdots + Y_{N-1}) + \frac{1}{M} (R_0 - R_N) \quad (3.38)$$

Introducing the round-off effect into equation 3.38, as presented in equation 3.36, gives

$$\begin{aligned} Z_n - Z_0 &= \frac{1}{M} (y_0 + y_1 + \cdots + y_{N-1}) + \frac{1}{M} (R_0 - R_N) \\ &+ \frac{1}{M} (e_{q0} + e_{q1} + \cdots + e_{qN-1}) \end{aligned} \quad (3.39)$$

As was presented in section 3.3, the integration is the sum of the areas of the rectangles of height Y and width δX , which is an approximation of the area under the curve. A better approximation for the area under the curve is obtained using the trapezoidal integration, where the area under the curve between two consecutive values

of y is

$$\frac{1}{2}\delta X (y_i + y_{i+1}) \quad (3.40)$$

Thus

$$\begin{aligned} \int_{x_0}^{x_N} y \cdot dX &\cong \frac{1}{2}\delta X (y_0 + y_1) + \frac{1}{2}\delta X (y_1 + y_2) + \cdots + \frac{1}{2}\delta X (y_{N-2} + y_{N-1}) \\ &= \frac{1}{2}\delta X (y_N - y_0) + \delta X (y_0 + y_1 + \cdots + y_{N-1}) \end{aligned} \quad (3.41)$$

Substituting equation 3.41 into 3.39 gives

$$\begin{aligned} Z_n - Z_0 &= \frac{1}{M\delta X} \int_{x_0}^{x_N} y \cdot dX - \frac{1}{2M} (y_N - y_0) + \frac{1}{M} (R_0 - R_N) \\ &\quad + \frac{1}{M} (e_{q0} + e_{q1} + \cdots + e_{qN-1}) \end{aligned} \quad (3.42)$$

Equation 3.42 gives the relation between the values of the register Z and the approximated area under the curve and shows the various errors involved. The first point to note is the introduction of the scale factor $\frac{1}{M\delta X}$ which makes the equation non-dimensional.

The term $\frac{1}{2M} (y_N - y_0)$ is due to the sampling of the curve at discrete instants of time and is the quantization error. As presented in reference [Sizer 68], this term does not represent the complete error but it is the most significant component. It is in fact the contribution of the term $(\delta X)^2$ in the Taylor expansion of the integral. In the worst case $y_N = M$ and $y_0 = -M$ and the magnitude of this term will be one bit. Thus for

simple integration this error will never exceed one unit in register Y .

The term $\frac{(R_0 - R_N)}{M}$ is the first rounding error and it arises from the number in the R register of the first integrator. As the maximum value of the register R is M , the maximum value for this error is one unit.

The term $\frac{(e_{q0} + e_{q1} + \dots + e_{qN-1})}{M}$ is due to the rounding effect of the A/D converter and it differs from the term $\frac{(R_0 - R_N)}{M}$ in that it depends on the number of integration steps taken. Since the average value of the terms e_{qi} is one-half unit, this term contributes at the rate of one-half bit in M iterations steps.

Reference [Sizer 68] presents other error analysis for implementations of DDA for multiplication, square, and exponential function.

In order to correctly evaluate the error through the DDA implemented for a particular problem, an analysis similar to this one, for simple integration, should be done, but for all the integrators in the solution model.

3.7 Sample Data Theory Applied to DDA

3.7.1 Definitions

Some Elementary Discrete-Time Functions

1. *Unit Sample Sequence*

$$\delta(n) \equiv \begin{cases} 1, & \text{for } n = 0 \\ 0, & \text{for } n \neq 0 \end{cases} \quad (3.43)$$

2. *Unit Step*

$$u(n) \equiv \begin{cases} 1, & \text{for } n \geq 0 \\ 0, & \text{for } n < 0 \end{cases} \quad (3.44)$$

3. *Unit Ramp*

$$u_r(n) \equiv \begin{cases} n, & \text{for } n \geq 0 \\ 0, & \text{for } n < 0 \end{cases} \quad (3.45)$$

4. *Exponential*

$$x(n) = a^n, \text{ for all } n \quad (3.46)$$

Causal and Noncausal Systems

From [Proakis 96], a system is said to be *causal* if the output of the system at any time depends only on present and past inputs. In mathematical terms, the output of a causal system satisfies an equation of the form

$$y(n) = F[x(n), x(n-1), x(n-2), \dots] \quad (3.47)$$

where F is an arbitrary function.

If a system does not satisfy this definition, it is called *noncausal*. Such a system has an output that is dependent on the present, past and future inputs. The future values of a signal are not observable in real-time signal processing applications, hence

a noncausal system is physically unrealizable.

3.7.2 Z-Transform

Given a discrete function $x(n)$ the Z-Transform is defined as the power series

$$X(z) = \sum_{n=-\infty}^{\infty} x(n) \cdot z^{-n} \quad (3.48)$$

Since the z-transform is an infinite power series, it exists only for those values of z for which this series converges. The *region of convergence* (ROC) of $X(z)$ is the set of all values of z for which $X(z)$ has a finite value.

Signal, $x(n)$	z-Transform	ROC
$\delta(n)$	1	All z
$u(n)$	$\frac{1}{1-z^{-1}}$	$ z > 1$
$a^n u(n)$	$\frac{1}{1-az^{-1}}$	$ z > a $
$na^n u(n)$	$\frac{az^{-1}}{(1-az^{-1})^2}$	$ z > a $
$-a^n u(-n-1)$	$\frac{1}{1-az^{-1}}$	$ z < a $
$-na^n u(-n-1)$	$\frac{az^{-1}}{(1-az^{-1})^2}$	$ z < a $
$(\cos \omega_0 n) u(n)$	$\frac{1-z^{-1} \cos \omega_0}{1-2z^{-1} \cos \omega_0 + z^{-2}}$	$ z > 1$
$(\sin \omega_0 n) u(n)$	$\frac{z^{-1} \sin \omega_0}{1-2z^{-1} \cos \omega_0 + z^{-2}}$	$ z > 1$
$(a^n \cos \omega_0 n) u(n)$	$\frac{1-az^{-1} \cos \omega_0}{1-2az^{-1} \cos \omega_0 + a^2 z^{-2}}$	$ z > a $
$(a^n \sin \omega_0 n) u(n)$	$\frac{az^{-1} \sin \omega_0}{1-2az^{-1} \cos \omega_0 + a^2 z^{-2}}$	$ z > a $

Table 3.3: Z-transform of common functions [Proakis 96].

All the properties and Z-transforms in tables 3.3 and 3.4 can be found in reference [Proakis 96].

Defining the operator *backward difference* ∇ as:

$$\nabla f = 1 - f(n-1)$$

so from the time shifting property presented in table 3.4

$$Z[\nabla f(n)] = (1 - z^{-1}) \cdot F(z)$$

Property	Time Domain	z-Domain	ROC
Linearity	$a_1 x_1(n) + a_2 x_2(n)$	$a_1 X_1(z) + a_2 X_2(z)$	At least the intersection of ROC_1 and ROC_2
Time shifting	$x(n - k)$	$z^{-k} X(z)$	That of $X(z)$, except $z = 0$ if $k > 0$ and $z = \infty$ if $k < 0$
Scaling	$a^n x(n)$	$X(a^{-1}z)$	$ a r_2 < z < a r_1$
Time reversal	$x(-n)$	$X(z^{-1})$	$\frac{1}{r_1} < z < \frac{1}{r_2}$
Backward difference	$\nabla x(n)$	$(1 - z^{-1}) X(z)$	That of $X(z)$, except $z = 0$ if $k > 0$ and $z = \infty$ if $k < 0$

Table 3.4: Properties of the Z-transform of common functions [Proakis 96].

3.7.3 The Integrator

Assuming a sequential DDA and defining all functions in relation to a discrete iteration cycle beginning at the first integrator, an expression for the output of the integrator may be written down at the n th iteration using backward difference operators, as proposed in [Sizer 68].

In section 3.3 the integrator has been defined as

$$Y^* = Y + \delta Y$$

$$I^* = I + Y^* \cdot \delta X$$

To make it easier to understand the integration process a change in the notation is necessary. Let $I^* = o_n + r_n$ where the index n means the n th iteration, o_n is the output,

and r_n the round-off. Let $Y^* = i_n$ where i_n is the input and $\delta X = t_n$. Using the ∇ operator the 'integration' could be represented by

$$\nabla (o_n + r_n) = t_n \cdot i_n \quad (3.49)$$

the scale factors are implicit in the variables t_n and r_n .

Taking the Z-transform of equation 3.49

$$\begin{aligned} (1 - z^{-1}) \cdot [O(z) + R(z)] &= Z[t_n \cdot i_n] \\ \therefore O(z) + R(z) &= \frac{1}{1 - z^{-1}} \cdot Z[t_n \cdot i_n] \end{aligned}$$

where the capital Z means the Z-transform.

Finding the inverse of the Z-transform

$$o_n + r_n = Z^{-1} \left(\frac{1}{1 - z^{-1}} \{Z[t_n \cdot i_n]\} \right) \quad (3.50)$$

Chapter 4

HRU Algorithms Implemented in DDA Form

This chapter will describe the algorithms implemented with the DDA technology for the HRU System.

Work [Bolduc 95] is based on a 21 bit digitizer and a procedural software implemented using 2 CPU's, one CPU for the I/O (input/output) scheduling, timer interfaces and processing interrupts, and the other CPU to calculate the navigation solution. In this configuration the compensation is performed by the first CPU at a faster rate, while the navigation algorithm is run at a slower rate due to the complexity associated with it.

Due to the time response of the digitizer and the required frequency of 200Hz, the precision of 21 bits could not be reached, instead an effective 17 bits was the precision of the system [Hartl-1 97].

In order to solve these problems, this work provides a different approach with the

DDA technology. Under this approach, 2 CPU's are no longer needed and it can be implemented in by FPGA (Field-Programmable Gate Array) or EPLD (Erasable Programmable Logic Device), reducing costs and improving reliability, since the number of components are reduced.

The algorithm implemented is the same as that presented in [Bolduc 95]. The next sections begins by describing the algorithms and then the corresponding DDA implementation.

4.1 HRU Alignment Scheme

Alignment for an inertial navigation systems (INS) is the process of determining the angular relationship between the platform and the computational frame [Britting 71]. External techniques such as optical alignment can be used for system alignment, but in cases like in a submarine this could not be a feasible method.

A self-alignment technique addresses the questions of how to use the outputs of the inertial sensors, which are resolved in the platform frame, to determine the relation between the platform frame and some reference frame. The reference frame that is of interest is the local-level wander-azimuth frame of reference.

[Britting 71] presents in detail gyrocompass alignment and analytic alignment. The alignment scheme presented in reference [Bolduc 95] is divided into a two phase scheme the coarse and the fine alignment.

The first phase is similar to the analytical phase described in [Britting 71], with a large sampling period to minimize the errors due to the sensors turn-on to turn-on bias repeatability, instruments non-orthogonality, and external perturbations, such as

vibration or accelerations, which can affect the gravity and earth measurements.

Since the approach presented in this work is based on DDA, the first stage of coarse alignment was not implemented, but just the fine alignment stage.

The fine alignment stage consists of a Schuler-tuned filter used to correct the small angle misalignments between the indicated computational frame of reference and the true frame of reference.

To explain the concept of the Schuler filter consider a simple pendulum suspended over the Earth with pivot on the vehicle. When the vehicle is at rest the pendulum axis is coincident with the true vertical direction, but when the vehicle is accelerated the pendulum becomes aligned with the apparent vertical axis, which is offset from the true vertical axis. A pendulum that remains aligned with the true vertical axis in the presence of acceleration would have its pivot at the Earth's surface and the center of mass at the center of the Earth. The frequency of such a pendulum is the Schuler frequency. This pendulum is physically unrealizable, by tuning a pendulum at the Schuler frequency the angular velocity of the pendulum will be equal to the angular velocity of the vertical. So the pendulum remains aligned with the vertical in the presence of accelerations, and initial misalignments oscillate about the vertical with a period of 84.4 minutes, the Schuler frequency.

In a Schuler-tuned mechanization, the accelerometer outputs are integrated and divided by the earth's radius to determine the rate to correct for linear motions. The integrator can be replaced by a low pass filter with gain $\frac{1}{R}$. With this approach R can be adjusted until optimum damping is achieved.

[Bolduc 95] and [Staley 84] presented the model in figure 4.1, the damped platform alignment model for the North axis accelerometer.

Figure 4.1: Damped Platform Alignment Model for the North-Axis Accelerometer.

It is presented in [Bolduc 95] that this approach does not require the vehicle to be stationary, which is advantageous because it does not need the user to wait for long periods of time before departure, and allows alignment in a ship at sea subjected to rolling and pitching motions.

In this approach navigation and fine alignment are concurrent whether or not the vehicle is in motion, improving the navigation solution as gradually as the alignment progresses.

In the following sections the various parts of the algorithm developed by [Bolduc 95] will be presented and how they are implemented in a DDA form will be described.

4.2 Compensation algorithms

The compensation of raw data from the accelerometer and from the gyros can be found from equations 4.1 to 4.3.

$$\begin{aligned}
 \begin{bmatrix} -\omega_x \\ -\omega_y \\ \omega_z \end{bmatrix} &= \begin{bmatrix} 1 & G_{yx} & G_{zx1} \\ G_{xy} & 1 & G_{zy1} \\ G_{xz1} & G_{yz1} & 1 \end{bmatrix} \cdot \left(\begin{bmatrix} GK_x & 0 & 0 \\ 0 & GK_y & 0 \\ 0 & 0 & GK_z \end{bmatrix} \cdot \begin{bmatrix} \omega_{xr} \\ \omega_{yr} \\ \omega_{zr} \end{bmatrix} - \right. \\
 &\quad \left. - \begin{bmatrix} GB_x \\ GB_y \\ GB_z \end{bmatrix} - \begin{bmatrix} Q_x \cdot a_{Gz1} \\ Q_y \cdot a_{Gz2} \\ Q_z \cdot a_{Gy} \end{bmatrix} - \begin{bmatrix} MU_x \cdot a_{Gx} \\ MU_y \cdot a_{Gy} \\ MU_z \cdot a_{Gz1} \end{bmatrix} \right) \quad (4.1)
 \end{aligned}$$

$$\begin{aligned}
 \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} &= \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & A_{yx} & A_{zx} \\ A_{xy} & 1 & A_{zy} \\ A_{xz} & A_{yz} & 1 \end{bmatrix} \cdot \\
 &\quad \cdot \left(\begin{bmatrix} AK_x & 0 & 0 \\ 0 & AK_y & 0 \\ 0 & 0 & AK_z \end{bmatrix} \cdot \begin{bmatrix} a_{xr} \\ a_{yr} \\ a_{zr} \end{bmatrix} - \begin{bmatrix} AB_x \\ AB_y \\ AB_z \end{bmatrix} \right) \quad (4.2)
 \end{aligned}$$

$$\begin{bmatrix} a_{Gx} \\ a_{Gy} \\ a_{Gz1} \\ a_{Gz2} \end{bmatrix} = \begin{bmatrix} 1 & G_{xy} & G_{xz1} & 0 \\ G_{yx} & 1 & G_{yz1} & 0 \\ G_{zx1} & G_{zy1} & 1 & 0 \\ G_{zx2} & G_{zy2} & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} a_x \\ a_y \\ a_z \\ a_z \end{bmatrix} \quad (4.3)$$

where:

- a_{ir} is the raw data from the accelerometers for the “i” axis;
- a_i is the compensated acceleration for the “i” axis;
- a_{Gi} is the acceleration along the gyroscope axis “i”;
- A_{ij} is the accelerometer misalignment angles, the first subscript refers to the body axis and the second refers to the system axis;
- AB_i is the accelerometer bias drift for the “i” axis;
- AK_i is the accelerometers scale factor, where $i = \{x, y, z\}$;
- G_{ij} is the gyroscope misalignment angles, the first subscript refers to the body axis and the second refers to the system axis;
- GB_i is the gyroscope bias drift for the “i” axis;
- GK_i is the gyroscope scale factors, where $i = \{x, y, z\}$;
- MU_i is the gyroscope mass unbalance drift term for the “i” axis; and
- Q_i is the gyroscope quadrature drift term for the “i” axis;
- ω Compensated angular velocity;
- ω_r Raw data from the gyroscopes.

These equations are implemented in a DDA form in the following sections. These equations are applied to the raw data from the sensors before alignment and navigation algorithms.

As can be seen in figures 4.2 and 4.3 this part of the algorithm consists of constant multipliers and the bias are part of the initial values of the integrators. Special attention should be paid, since the values are propagated through all the others integrators.

4.2.1 Acceleration compensator

From equation 4.2, the compensator of the raw acceleration is found to be

$$\begin{aligned}
 a_x &= AK_x \cdot a_{xr} + AB_x + A_{yx} (AK_y \cdot a_{yr} - AB_y) + A_{zx} (AK_z \cdot a_{zr} - AB_z) \\
 a_y &= A_{xy} (AK_x \cdot a_{xr} + AB_x) + AK_y \cdot a_{yr} - AB_y + A_{zy} (AK_z \cdot a_{zr} - AB_z) \quad (4.4) \\
 a_z &= A_{xz} (AK_x \cdot a_{xr} + AB_x) + A_{yz} (AK_y \cdot a_{yr} - AB_y) + AK_z \cdot a_{zr} - AB_z
 \end{aligned}$$

The integrator scale factors of the integrators in figure 4.2 are presented in a table form in table 4.1 where:

- M_k is the supremum of the accelerometers scale factors, $M_k = \sup(AK_i)$. Since $-1 \leq A_{ij} \leq 1$, M_k is also the supremum of the $A_{ij} \cdot AK_i$ terms, where $i, j = x, y, z$;
- N_a is the size of the registers Y and R of the integrators, for a 21 bit register it will be $N_a = 2^{21} = 2,097,152$;
- a_{max} is the maximum value that any acceleration can assume, so $K_a = \frac{a_{max}}{N_a}$ is the integrator scale factor of the compensated acceleration ;

Figure 4.2: The DDA implementation of the equation 4.4, the compensation of the accelerometer raw data where the bias, misalignments and scale factors are applied.

- K_{ar} is the scale factor for the accelerometer's output, $K_{ar} = \frac{a_{rmax}}{N_a}$ where a_{rmax} is the maximum output from the accelerometers;
- K_t is the time base scale factor. It relates the time of the integrators and the time base of the system.

The relation between the integrators $\{1, 2, 3\}$ and 10 gives $K_{y10} = K_{z1} = K_{z2} = K_{z3}$.

The same relation can be applied to the integrators $\{4, 5, 6\}$ and 11, $\{7, 8, 9\}$ and 12.

With these relations the table 4.1 has been built.

Integrator	Y Register	ΔX	ΔY	M	K_x	K_z	K_y
1	AK_x	δa_{xr}	-	M_k	K_{ar}	$M_k \cdot K_{ar}$	M_k / N_a
2	$AK_y A_{yx}$	δa_{yr}	-	M_k	K_{ar}	$M_k \cdot K_{ar}$	M_k / N_a
3	$AK_z A_{zx}$	δa_{zr}	-	M_k	K_{ar}	$M_k \cdot K_{ar}$	M_k / N_a
4	$AK_x A_{xy}$	δa_{xr}	-	M_k	K_{ar}	$M_k \cdot K_{ar}$	M_k / N_a
5	AK_y	δa_{yr}	-	M_k	K_{ar}	$M_k \cdot K_{ar}$	M_k / N_a
6	$AK_z A_{zy}$	δa_{zr}	-	M_k	K_{ar}	$M_k \cdot K_{ar}$	M_k / N_a
7	$AK_x A_{xz}$	δa_{xr}	-	M_k	K_{ar}	$M_k \cdot K_{ar}$	M_k / N_a
8	$AK_y A_{yz}$	δa_{yr}	-	M_k	K_{ar}	$M_k \cdot K_{ar}$	M_k / N_a
9	AK_z	δa_{zr}	-	M_k	K_{ar}	$M_k \cdot K_{ar}$	M_k / N_a
10	a_x	δt	δa_x	a_{max}	K_t	$M_k \cdot K_{ar} \cdot K_t \cdot N_a$	$M_k \cdot K_{ar}$
11	a_y	δt	δa_y	a_{max}	K_t	$M_k \cdot K_{ar} \cdot K_t \cdot N_a$	$M_k \cdot K_{ar}$
12	a_z	δt	δa_z	a_{max}	K_t	$M_k \cdot K_{ar} \cdot K_t \cdot N_a$	$M_k \cdot K_{ar}$

Table 4.1: These are the relations between all the scale factors of the integrators of figure 4.2, the acceleration compensator.

The maximum acceleration output from the compensator, based on the integrator's relations is

$$K_{z1} = K_{y10} = K_a = \frac{a_{max}}{N_a} \quad (4.5)$$

$$= M_k K_{ar} = M_k \cdot \frac{a_{rmax}}{N_a}$$

$$\therefore a_{max} = M_k \cdot a_{rmax} \quad (4.6)$$

4.2.2 Angular Velocity Compensator

From equations 4.1, the compensation of the angular velocity component ω_x is

$$\begin{aligned}
 \omega_x &= GK_x\omega_{xr} + GB_x + Q_x a_{Gz1} + MU_x a_{Gx} \\
 &+ G_{yx} [GK_y\omega_{yr} GB_y + Q_y a_{Gz2} + MU_y a_{Gy}] \\
 &+ G_{zx} [GK_z\omega_{zr} - GB_z - Q_z a_{Gy} - MU_z a_{Gz1}]
 \end{aligned} \tag{4.7}$$

but from equation 4.3

$$\begin{aligned}
 a_{Gx} &= a_x + G_{xy}a_y + G_{xz1}a_z \\
 a_{Gy} &= G_{yx}a_x + a_y + G_{yz1}a_z \\
 a_{Gz1} &= G_{z1x}a_x + G_{z1y}a_y + a_z \\
 a_{Gz2} &= G_{z2x}a_x + G_{z2y}a_y + a_z
 \end{aligned} \tag{4.8}$$

From equations 4.7 and 4.8

$$\begin{aligned}
\omega_x = & GK_x\omega_{xr} + G_{yx}GK_y\omega_{yr} + G_{z1x}GK_z\omega_{zr} \\
& + (GB_x + G_{yx}GB_y - G_{z1x}GB_{z2}) \\
& + a_x [Q_x G_{z1x} + MU_x + G_{yx} (Q_y G_{z2x} + MU_y G_{yx}) \\
& - G_{z1x} (Q_{z2} G_{yx} + MU_{z2} G_{z2x})] \\
& + a_y [Q_x G_{z1y} + MU_x G_{xy} + G_{yx} (Q_y G_{z2y} + MU_y) \\
& - G_{z1x} (Q_{z2} + MU_{z2} G_{z2y})] \\
& + a_z [Q_x + MU_x G_{xz1} + G_{yx} (Q_y + MU_y G_{yz2}) \\
& - G_{z1x} (Q_{z2} G_{yz2} + MU_{z2})]
\end{aligned} \tag{4.9}$$

With the same approach, but for the components ω_y and ω_z the following equations can be found

$$\begin{aligned}
\omega_y = & G_{xy}GK_x\omega_{xr} + GK_y\omega_{yr} + G_{z2y}GK_z\omega_{zr} \\
& + (G_{xy}GB_x + GB_y - G_{z1y}GB_{z2}) \\
& + a_x [G_{xy} (Q_x G_{z1x} + MU_x) + Q_y G_{z2x} + MU_y G_{yx} \\
& - G_{z2y} (Q_{z2} G_{yx} + MU_{z2} G_{z2x})] \\
& + a_y [G_{xy} (Q_x G_{z1y} + MU_x G_{xy}) + Q_y G_{z2y} + MU_y \\
& - G_{z2y} (Q_{z2} + MU_{z2} G_{z2y})] \\
& + a_z [G_{xy} (Q_x + MU_x G_{xz1}) + Q_y + MU_y G_{yz2} \\
& - G_{z2y} (Q_{z2} G_{yz2} + MU_{z2})]
\end{aligned} \tag{4.10}$$

$$\begin{aligned}
\omega_z = & G_{xz1}GK_x\omega_{xr} + G_{yz1}GK_y\omega_{yr} + GK_z\omega_{zr} \\
& + (G_{xz1}GB_x + G_{yz1}GB_y - GB_{z2}) \\
& + a_x [G_{xz2} (Q_x G_{z1x} + MU_x) + G_{yz2} (Q_y G_{z2x} + MU_y G_{yx}) \\
& - Q_{z2}G_{yz} - MU_{z2}G_{z2x}] \\
& + ay [G_{xz2} (Q_x G_{z1y} + MU_x G_{xy}) + G_{yz2} (Q_y G_{z2y} + MU_y) \\
& - Q_{z2} - MU_{z2}G_{z2y}] \\
& + a_z [G_{xz2} (Q_x + MU_x G_{xz1}) + G_{yz2} (Q_y + MU_y G_{yz2}) \\
& - Q_{z2}G_{yz2} - MU_{z2}]
\end{aligned} \tag{4.11}$$

The equations 4.9, 4.10 and 4.11 have the form of

$$\begin{aligned}
\omega_i = & \omega_{rx}K_{\omega_{zi}} + \omega_{yr}K_{\omega_{yi}} + \omega_{zr}K_{\omega_{zi}} + GB_i \\
& + a_x.K_{a_{zi}} + a_yK_{a_{yi}} + a_zK_{a_{zi}}
\end{aligned} \tag{4.12}$$

where $i = x, y, z$.

Based on equations 4.9, 4.10, 4.11 and 4.12, the compensator algorithm implementation using DDA is presented in figure 4.3.

With the same approach of section 4.2.1, but with respect to figure 4.3, the corresponding integrator scale factors are presented in table 4.3. The parameters are presented in table 4.2.

Figure 4.3: The DDA implementation of equation 4.9, the angular velocity compensator for ω_x component. The implementation of the components ω_y and ω_z compensators are analogous to this.

Parameter	Value	Parameter	Value
$K_{\omega_{zx}}$	GK_x	$K_{a_{zx}}$	$Q_x G_{z1x} + MU_x + G_{yx} (Q_y G_{z2x} + MU_y G_{yx}) + G_{z1x} (Q_z G_{yx} + MU_z G_{z1x})$
$K_{\omega_{zy}}$	$G_{xy} GK_x$	$K_{a_{zy}}$	$G_{xy} (Q_x G_{z1x} + MU_x) + Q_y G_{z2x} + MU_y G_{yx} + G_{z1y} (Q_z G_{yx} + MU_z G_{z1x})$
$K_{\omega_{zz}}$	$G_{xz1} GK_x$	$K_{a_{zz}}$	$G_{xz1} (Q_x G_{z1x} + MU_x) + G_{yz1} (Q_y G_{z2x} + MU_y G_{yx}) + Q_z G_{yx} + MU_z G_{z1x}$
$K_{\omega_{yx}}$	$G_{yz} GK_x$	$K_{a_{yx}}$	$Q_x G_{z1y} + MU_x G_{xy} + G_{yz} (Q_y G_{z2y} + MU_y) + G_{z1x} (Q_z + MU_z G_{z1y})$
$K_{\omega_{yy}}$	GK_y	$K_{a_{yy}}$	$G_{xy} (Q_x G_{z1y} + MU_x G_{xy}) + Q_y G_{z2y} + MU_y + G_{z1y} (Q_z + MU_z G_{z1y})$
$K_{\omega_{yz}}$	$G_{yz1} GK_z$	$K_{a_{yz}}$	$G_{xz1} (Q_x G_{z1y} + MU_x G_{xy}) + G_{yz1} (Q_y G_{z2y} + MU_y) + Q_z + MU_z G_{z1y}$
$K_{\omega_{zx}}$	$G_{z1x} GK_x$	$K_{a_{zx}}$	$Q_x + MU_x G_{xz1} + G_{yx} (Q_y + MU_y G_{yz1}) + G_{z1x} (Q_z G_{yz1} + MU_z)$
$K_{\omega_{zy}}$	$G_{z1y} GK_y$	$K_{a_{zy}}$	$G_{xy} (Q_x + MU_x G_{xz1}) + Q_y + MU_y G_{yz1} + G_{z1y} (Q_z G_{yz1} + MU_z)$
$K_{\omega_{zz}}$	GK_z	$K_{a_{zz}}$	$G_{xz1} (Q_x + MU_x G_{xz1}) + G_{yz1} (Q_y + MU_y G_{yz1}) + Q_z G_{yz1} + MU_z$

Table 4.2: Parameters for the angular velocity compensator in figure 4.3 based on equations 4.9, 4.10 and 4.11 used in table 4.3.

From figure 4.3, the integrators $\{1, 2, 3, 4, 5, 6\}$ and 19 have the following relations

$$\begin{aligned} K_{z1} &= K_{z2} = K_{z3} = K_{z4} = K_{z5} = K_{z6} = K_{y19} \\ K_{x1} &= K_{x2} = K_{x3} \\ K_{x4} &= K_{x5} = K_{x6} \end{aligned} \tag{4.13}$$

Based on these relations and defining:

- M_{kw} as the supremum of $\left\{ K_{\omega_{xi}}, K_{\omega_{yi}}, K_{\omega_{zi}} \right\}$ from table 4.2, where $i = x, y, z$;
- M_{ka} as the supremum of $\left\{ K_{a_{xi}}, K_{a_{yi}}, K_{a_{zi}} \right\}$ from table 4.2, where $i = x, y, z$;
- $K_{\omega r}$ as the scale factor of the output of the gyros, $K_{\omega r} = \frac{\omega_{rmax}}{N_w}$;
- K_a as the scale factor of the output from the acceleration compensator, from equation 4.5 $K_a = \frac{M_k \cdot a_{rmax}}{N_a}$;
- N_w as the size of the integrators' registers Y and R ;
- K_t as the time base scale factor. It relates the time of the integrators and the time base of the system.

The table 4.3 has been built with the integrator scale factors of figure 4.3.

From equation 4.13

$$K_w = M_{ka} \cdot K_a = M_{kw} \cdot K_{\omega r} \tag{4.14}$$

Integrator	Y Register	ΔX	ΔY	M	K_x	K_z	K_y
1	$K_{\omega_x x}$	$\delta\omega_{xt}$	-	M_{kw}	$K_{\omega t}$	$M_{kw} \cdot K_{\omega t}$	M_{kw}/N_ω
2	$K_{\omega_y x}$	$\delta\omega_{yt}$	-	M_{kw}	$K_{\omega t}$	$M_{kw} \cdot K_{\omega t}$	M_{kw}/N_ω
3	$K_{\omega_z x}$	$\delta\omega_{zt}$	-	M_{kw}	$K_{\omega t}$	$M_{kw} \cdot K_{\omega t}$	M_{kw}/N_ω
4	$K_{a_x x}$	δa_x	-	M_{ka}	K_a	$M_{ka} \cdot K_a$	M_{ka}/N_ω
5	$K_{a_y x}$	δa_y	-	M_{ka}	K_a	$M_{ka} \cdot K_a$	M_{ka}/N_ω
6	$K_{a_z x}$	δa_z	-	M_{ka}	K_a	$M_{ka} \cdot K_a$	M_{ka}/N_ω
7	$K_{\omega_x y}$	$\delta\omega_{xt}$	-	M_{kw}	$K_{\omega t}$	$M_{kw} \cdot K_{\omega t}$	M_{kw}/N_ω
8	$K_{\omega_y y}$	$\delta\omega_{yt}$	-	M_{kw}	$K_{\omega t}$	$M_{kw} \cdot K_{\omega t}$	M_{kw}/N_ω
9	$K_{\omega_z y}$	$\delta\omega_{zt}$	-	M_{kw}	$K_{\omega t}$	$M_{kw} \cdot K_{\omega t}$	M_{kw}/N_ω
10	$K_{a_x y}$	δa_x	-	M_{ka}	K_a	$M_{ka} \cdot K_a$	M_{ka}/N_ω
11	$K_{a_y y}$	δa_y	-	M_{ka}	K_a	$M_{ka} \cdot K_a$	M_{ka}/N_ω
12	$K_{a_z y}$	δa_z	-	M_{ka}	K_a	$M_{ka} \cdot K_a$	M_{ka}/N_ω
13	$K_{\omega_x z}$	$\delta\omega_{xt}$	-	M_{kw}	$K_{\omega t}$	$M_{kw} \cdot K_{\omega t}$	M_{kw}/N_ω
14	$K_{\omega_y z}$	$\delta\omega_{yt}$	-	M_{kw}	$K_{\omega t}$	$M_{kw} \cdot K_{\omega t}$	M_{kw}/N_ω
15	$K_{\omega_z z}$	$\delta\omega_{zt}$	-	M_{kw}	$K_{\omega t}$	$M_{kw} \cdot K_{\omega t}$	M_{kw}/N_ω
16	$K_{a_x z}$	δa_x	-	M_{ka}	K_a	$M_{ka} \cdot K_a$	M_{ka}/N_ω
17	$K_{a_y z}$	δa_y	-	M_{ka}	K_a	$M_{ka} \cdot K_a$	M_{ka}/N_ω
18	$K_{a_z z}$	δa_z	-	M_{ka}	K_a	$M_{ka} \cdot K_a$	M_{ka}/N_ω
19	ω_x	δt	$\delta\omega_x$	ω_{max}	K_t	$K_\omega \cdot K_t \cdot N_\omega$	K_ω
20	ω_y	δt	$\delta\omega_y$	ω_{max}	K_t	$K_\omega \cdot K_t \cdot N_\omega$	K_ω
21	ω_z	δt	$\delta\omega_z$	ω_{max}	K_t	$K_\omega \cdot K_t \cdot N_\omega$	K_ω

Table 4.3: Scale factors of the integrators in figure 4.3, the DDA implementation of the angular velocity compensator. The parameter M_{ka} is the supremum of the terms $K_{a,j}$ presented in table 4.2, and M_{kw} is the supremum of the terms $K_{\omega i,j}$ also presented in table 4.2, where $i, j = x, y, z$.

Defining ω_{max} as the maximum value for angular velocity, it can be found from 4.14 to be

$$\omega_{max} = M_{ka} \cdot a_{max} \cdot \frac{N_{\omega}}{N_a} = M_{k\omega} \cdot \omega_{rmax} \quad (4.15)$$

4.3 Navigation Algorithm

The navigation algorithm is used to integrate the position and attitude of the HRU system. It is not the scope of this work to deduce this algorithm, which is well presented in the references [Britting 71], [Bolduc 95] and [Siouris 93]. It will be presented in this section the principles and equations needed to implement the navigation algorithm in the DDA approach.

This process is concurrent to the compensation process presented in section 4.2. The alignment mechanization is implemented as part of the navigation algorithm through the damped spatial rates, which implements the modified version of the Schuler filter, as presented in references [Bolduc 95] and [Staley 84].

4.3.1 Damped Spatial Rates

Reference [Bolduc 95] develops an algorithm for fine-alignment, similar to [Staley 84] approach, where a damping ratio is applied to the North and East axes, the level axes.

$$\begin{aligned} \dot{\nu}_x + \frac{1}{\tau} \cdot \nu_x &= \dot{V}_{cx} \\ \dot{\nu}_y + \frac{1}{\tau} \cdot \nu_y &= \dot{V}_{cy} \end{aligned} \quad (4.16)$$

To find these results, it has been considered that the roll and pitch error angles are independent from the heading angle error, and that all bias instability is negligible, since they are transformed by the small angle misalignment.

All the hypotheses assumed, their consequences, stability and error analysis are presented in reference [Bolduc 95]. From those results, the optimum value for the R and τ parameters are found to be $R = 3000$ km and $\tau = 1080$ s.

The heading errors are also damped, but, unlike the level axes, a precession rate is not applied to the azimuth channel to correct for the vehicle's motion over the spherical earth. To ensure that heading errors are stable a rate correction is applied to the azimuth channel. [Bolduc 95] presents the stability analysis of the azimuth channel which leads to the result presented in equation 4.16, and the threshold value for ' a ' as 0.003 s/m, below which the system would become unstable for the case study of 45° latitude position. From experimental and simulation work the optimum value for ' a ' is presented to be $a = 0.43$ s/m.

Following the approach in [Bolduc 95] and from equation 4.16, the damped spatial angular rate between the computational frame of reference and the body frame of reference is

$$\begin{aligned}\rho_x &= -\frac{\nu_y}{R} \\ \rho_y &= \frac{\nu_x}{R} \\ \rho_z &= a \cdot \dot{V}_y\end{aligned}\tag{4.17}$$

The following relations are found from integrators in figure 4.4 (a) and (b).

Figure 4.4: DDA implementation of equations 4.16 and 4.17, the damped velocities and angular rates. This is the implementation of the self-alignment mechanization as proposed in references [Bolduc 95] and [Staley 84]

- from integrator 1:

$$K_{x_1} = K_t$$

$$K_{y_1} = K_{Vc}$$

$$= K_{z_2}$$

$$K_{z_1} = K_t \cdot K_{Vc} \cdot N_\rho$$

- from integrator 2:

$$K_{x_2} = K_{z_1} = K_t \cdot K_{Vc} \cdot N_\rho$$

$$K_{y_2} = \frac{M_\tau}{N_\rho}$$

$$K_{z_2} = K_t \cdot K_{Vc} \cdot N_\rho \cdot M_\tau$$

- from integrator 3:

$$K_{x_3} = K_{y_1} = K_{Vc}$$

$$K_{y_3} = \frac{M_R}{N_\rho}$$

$$K_{z_3} = K_{Vc} \cdot M_R$$

From the relations of the integrators 1, 2 and 3 in figure 4.4, the following constraints

are found

$$\nu_{max} = K_{Vc} \cdot N_\rho \quad (4.18)$$

$$N_\rho = \frac{1}{K_t \cdot M_\tau}, \quad M_\tau \geq \frac{1}{\tau} \quad (4.19)$$

$$K_\rho = K_{Vc} \cdot M_R \quad (4.20)$$

The integrator scale factors of the integrators in figure 4.4 are shown in table 4.4 where:

- K_{Vc} is the integrator scale factor of the true velocity, that will be addressed in section 4.3.7;
- ν_{max} is the maximum value that ν can assume ;
- M_τ is the supremum of $\frac{1}{\tau}$;
- M_R is the supremum of $\frac{1}{R}$;
- N_ρ is the size of the integrators' registers of the damped rate DDA implementation;
- K_t is the time base scale factor, it relates the time of the integrators and the time base of the system.

With respect to the integrator in figure 4.4 (c) its scale factor is

$$K_y = \frac{M_{act}}{N_\rho} \quad (4.21)$$

$$K_x = K_{Ac} \quad (4.22)$$

$$K_z = K_{Ac} \cdot M_{act} \quad (4.23)$$

Integrator	Y Register	ΔX	ΔY	M	K_x	K_z	K_y
1	ν_i	δt	$\delta \nu_i = \delta V_j + \frac{1}{\tau} \nu_i \cdot \delta t$	ν_{max}	K_t	$K_t \cdot K_{Vc} \cdot N_\rho$	K_{Vc}
2	$\frac{1}{\tau}$	$\nu_i \cdot \delta t$	-	M_τ	$K_t \cdot K_{Vc} \cdot N_\rho$	$K_t \cdot M_\tau \cdot K_{Vc} \cdot N_\rho$	M_τ / N_ρ
3	$\frac{1}{R}$	$\delta \nu_i$	-	M_R	K_{Vc}	$K_{Vc} \cdot M_R$	M_R / N_ρ

Table 4.4: The integrator scale factors of the DDA implementation of the damped velocities and angular rates in figure 4.4, where $i = \{x, y\}$ and $j = \{y, x\}$. These relations can be found from the corresponding integrators as presented in section 3.4.3. The basic relations between the inputs and output for each integrators are $K_{x_i} = \frac{M_{x_i}}{N_i}$, $K_{y_i} = \frac{M_{y_i}}{N_i}$ and $K_{z_i} = K_{x_i} \cdot K_{y_i} \cdot N_i = K_{x_i} \cdot M_{y_i}$, where M is the supremum of the corresponding input.

where K_{Ac} is the true acceleration integrator scale factor, which will be presented in section 4.3.5.

As K_z in figure 4.4 (c) should be equal to $K_{z_3} = K_\rho$ in figure 4.4 (a) and (b) the following relation is found

$$M_{act} = \frac{K_{Vc} \cdot M_R}{K_{Ac}} \geq a \quad (4.24)$$

where M_{act} is the supremum of the coefficient 'a' of equation 4.17.

4.3.2 The Direction Cosine Matrix Q

The DCM $\mathbf{Q}$ is the direction cosine matrix that transforms a vector in the system frame of reference to the computational frame of reference. It corresponds to a sequence of roll rotation (ϕ), followed by pitch rotation (θ) and yaw rotation (ψ). It can be found from reference [Staley 98] to be the following matrices as functions of the quaternions,

equation 4.25, and of the Euler angles, equation 4.26

$$\mathbf{Q} = \begin{bmatrix} (q_4^2 + q_1^2 - q_2^2 - q_3^2) & 2.(q_1.q_2 + q_3.q_4) & 2.(q_1.q_3 - q_2.q_4) \\ 2.(q_1.q_2 - q_3.q_4) & (q_4^2 - q_1^2 + q_2^2 - q_3^2) & 2.(q_2.q_3 + q_1.q_4) \\ 2.(q_1.q_3 + q_2.q_4) & 2.(q_2.q_3 - q_1.q_4) & (q_4^2 - q_1^2 - q_2^2 + q_3^2) \end{bmatrix} \quad (4.25)$$

$$\mathbf{C} = \begin{bmatrix} C_\theta.C_\psi & (-C_\phi.S_\psi + S_\phi.S_\theta.C_\psi) & (S_\phi.S_\psi + C_\phi.S_\theta.C_\psi) \\ C_\theta.S_\psi & (C_\phi.C_\psi + S_\phi.S_\theta.S_\psi) & (-S_\phi.C_\psi + C_\phi.S_\theta.S_\psi) \\ -S_\theta & S_\phi.C_\theta & C_\phi.C_\theta \end{bmatrix} \quad (4.26)$$

The $\mathbf{Q}$ matrix elements are implemented in DDA in figure 4.5 from the equation 4.25, where the inputs are the quaternions δq_i increments.

The integrators in figure 4.5 (b), (c) and (d) have the same topology and can be described through the same table of integrator scale factors. Table 4.5 represents the integrators in 4.5 (b).

The integrators in 4.5 (a) correspond to the diagonal elements of $\mathbf{Q}$ and their integrator scale factors are presented in table 4.6, where:

- K_q is the integrator scale factor of the DDA implementation of the quaternions to be presented in section 4.3.6;
- N_q is the size of the integrators of the quaternions, it is also the size of the integrators 1 to 12 in figure 4.5 (a), and 1 to 8 in figures 4.5 (b), (c) and (d);
- N_Q is the size of the integrators' registers 13 to 15 in figure 4.5 (a) and 9 and 10 in figures 4.5 (b), (c) and (d).

Figure 4.5: DDA implementation of the DCM Q that transforms from the system frame of reference to the computational frame of reference.

From section 4.4, the supremum of the quaternions M_q must be equal to 1. From the integrators 1 to 12 in figure 4.5 (a) and 1 to 8 in figures 4.5 (b), (c) and (d), the size of the registers must satisfy the condition $N \geq N_q$, in addition a constant multiplier of 2 must also be applied to δQ resulting in the condition $N_Q \geq N_q$.

From these tables K_Q , the integrator scale factors for the increments output of figure 4.5, needs to correct the times 2 factor. Defining $M_2 \geq 2$ the value of K_Q is

$$K_Q = M_2 \cdot K_q \cdot M_q \quad (4.27)$$

4.3.3 The Direction Cosine Matrix **D**

The direction cosine matrix (DCM) **D** transforms coordinates from the earth-fixed frame of reference to the computational frame of reference. It is used to find the components of Ω_{ie} , the earth angular rate, in the computational frame of reference and compensate it from the angular rates measured by the gyros.

[Bolduc 95] and [Siouris 93] present how one can find the following equation of **D**.

$$\mathbf{D} = \begin{bmatrix} C_\alpha \cdot C_\lambda - S_\alpha \cdot S_{\phi_L} \cdot S_\lambda & S_\alpha \cdot C_{\phi_L} & -C_\alpha \cdot S_\lambda - S_\alpha \cdot S_{\phi_L} \cdot C_\lambda \\ -S_\alpha \cdot C_\lambda - C_\alpha \cdot S_{\phi_L} \cdot S_\lambda & C_\alpha \cdot C_{\phi_L} & S_\alpha \cdot S_\lambda - C_\alpha \cdot S_{\phi_L} \cdot C_\lambda \\ C_{\phi_L} \cdot S_\lambda & S_{\phi_L} & C_{\phi_L} \cdot C_\lambda \end{bmatrix} \quad (4.28)$$

Defining $\hat{\rho}$ as the angular rate of the computational frame of reference with respect

Integrators	Y Registers	ΔX	ΔY	M	K_x	K_z	K_y
1	q_1	δq_1	δq_1	M_q	K_q	$K_q.M_q$	K_q
2	q_2	δq_2	δq_2	M_q	K_q	$K_q.M_q$	K_q
3	q_3	δq_3	δq_3	M_q	K_q	$K_q.M_q$	K_q
4	q_4	δq_4	δq_4	M_q	K_q	$K_q.M_q$	K_q
5	q_1	δq_1	δq_1	M_q	K_q	$K_q.M_q$	K_q
6	q_2	δq_2	δq_2	M_q	K_q	$K_q.M_q$	K_q
7	q_3	δq_3	δq_3	M_q	K_q	$K_q.M_q$	K_q
8	q_4	δq_4	δq_4	M_q	K_q	$K_q.M_q$	K_q
9	q_1	δq_1	δq_1	M_q	K_q	$K_q.M_q$	K_q
10	q_2	δq_2	δq_2	M_q	K_q	$K_q.M_q$	K_q
11	q_3	δq_3	δq_3	M_q	K_q	$K_q.M_q$	K_q
12	q_4	δq_4	δq_4	M_q	K_q	$K_q.M_q$	K_q
13	2	$\delta Q_{11}/2$	-	M_2	$K_q.M_q$	$M_2.K_q.M_q$	M_2/N_Q
14	2	$\delta Q_{22}/2$	-	M_2	$K_q.M_q$	$M_2.K_q.M_q$	M_2/N_Q
15	2	$\delta Q_{33}/2$	-	M_2	$K_q.M_q$	$M_2.K_q.M_q$	M_2/N_Q

Table 4.5: Integrator scale factors of the DDA implementation of the diagonal elements of figure 4.5 (a), the DCM matrix that transforms from the system frame of reference to the computational frame o reference.

Integrators	Y Registers	ΔX	ΔY	M	K_x	K_z	K_z
1	q_2	δq_1	δq_2	M_q	K_q	$K_q.M_q$	K_q
2	q_1	δq_2	δq_1	M_q	K_q	$K_q.M_q$	K_q
3	q_4	δq_3	δq_4	M_q	K_q	$K_q.M_q$	K_q
4	q_3	δq_4	δq_3	M_q	K_q	$K_q.M_q$	K_q
5	q_3	δq_1	δq_3	M_q	K_q	$K_q.M_q$	K_q
6	q_1	δq_3	δq_1	M_q	K_q	$K_q.M_q$	K_q
7	q_4	δq_2	δq_4	M_q	K_q	$K_q.M_q$	K_q
8	q_2	δq_4	δq_2	M_q	K_q	$K_q.M_q$	K_q
9	2	δQ_{12}	-	M_2	$K_q.M_q$	$M_2.K_q.M_q$	M_2/N_Q
10	2	δQ_{21}	-	M_2	$K_q.M_q$	$M_2.K_q.M_q$	M_2/N_Q

Table 4.6: Integrator scale factors of the DDA implementation of the non-diagonal elements of $\mathbf{Q}$ presented in figure 4.5 (b), the DCM matrix that transforms from the system frame of reference to the computational frame o reference. These relations can be found from the corresponding integrators as presented in section 3.4.3.

The basic relations between the inputs and output for each integrators are $K_{x_i} = \frac{M_{x_i}}{N_i}$, $K_{y_i} = \frac{M_{y_i}}{N_i}$ and $K_{z_i} = K_{x_i}.K_{y_i}.N_i = K_{x_i}.M_{y_i}$, where M is the supremum of the corresponding input.

to the earth-centered earth-fixed frame of reference, is shown in [Siouris 93] that

$$\begin{aligned}\hat{\rho}_x &= -\frac{V_{cy}}{R_o} \\ \hat{\rho}_y &= \frac{V_{cx}}{R_o}\end{aligned}$$

where $\mathbf{V}_c$ is the true velocity of the vehicle and R_o is the earth radius. As the wander-azimuth is the mechanization used, it is assumed that $\hat{\rho}_z = 0$.

From the definition of $\hat{\rho}$, the rate of change of the direction cosines for the wander-azimuth mechanization, where $\hat{\rho}_z = 0$, is [Siouris 93]

$$\begin{aligned}\dot{D}_{11} &= -\hat{\rho}_y \cdot D_{31} \\ \dot{D}_{21} &= \hat{\rho}_x \cdot D_{31} \\ \dot{D}_{31} &= \hat{\rho}_y \cdot D_{11} - \hat{\rho}_x \cdot D_{21} \\ \dot{D}_{12} &= -\hat{\rho}_y \cdot D_{32} \dot{D}_{22} = \hat{\rho}_x \cdot D_{32} \\ \dot{D}_{32} &= \hat{\rho}_y \cdot D_{12} - \hat{\rho}_x \cdot D_{22} \\ \dot{D}_{13} &= -\hat{\rho}_y \cdot D_{33} \\ \dot{D}_{23} &= \hat{\rho}_x \cdot D_{33} \\ \dot{D}_{33} &= \hat{\rho}_y \cdot D_{13} - \hat{\rho}_x \cdot D_{23}\end{aligned}\tag{4.29}$$

The sets of integrators $\{3, \dots, 10\}$, $\{11, \dots, 18\}$ and $\{19, \dots, 26\}$ have the same topology, as a result, the table and relations of integrator scale factors are the same. With this in mind only the table for the integrators 1 to 10 is presented and the results can be directly applied to the other integrator sets.

Figure 4.6: The DDA implementation of the DCM D that transforms from earth-centered earth-fixed frame of reference to the computational frame of reference.

Assuming K_{zi} is equal to $i = \{3, \dots, 10\}$, the following relations are found

$$K_{y3} = K_{x4} = K_{x7} = K_{y8} = K_{z1}$$

$$K_{x5} = K_{y6} = K_{x9} = K_{y10} = K_{z2}$$

$$K_{x3} = K_{y4} = K_{y5} = K_{x6} = K_{z7} = K_{z8}$$

$$K_{y7} = K_{x8} = K_{z3}$$

$$K_{y9} = K_{x10} = K_{z5}$$

The integrator scale factors for the **D** elements can be found and are presented in table 4.7.

Integrators	Y Registers	ΔX	ΔY	M	K_z	K_z	K_y
1	$\frac{1}{R_z}$	δV_z	-	M_{R_o}	K_{Vc}	$K_{Vc} \cdot M_{R_o}$	M_{R_o}/N_D
2	$\frac{1}{R_o}$	δV_y	-	M_{R_o}	K_{Vc}	$K_{Vc} \cdot M_{R_o}$	M_{R_o}/N_D
3	$\hat{\rho}_y$	δD_{31}	$\delta \hat{\rho}_y$	$K_{Vc} \cdot M_{R_o} \cdot N_D$	M_D/N_D	$K_{Vc} \cdot M_{R_o} \cdot M_D$	$K_{Vc} \cdot M_{R_o}$
4	D_{31}	$\delta \hat{\rho}_y$	δD_{31}	M_D	$K_{Vc} \cdot M_{R_o}$	$K_{Vc} \cdot M_{R_o} \cdot M_D$	M_D/N_D
5	D_{31}	$-\delta \hat{\rho}_x$	δD_{31}	M_D	$K_{Vc} \cdot M_{R_o}$	$K_{Vc} \cdot M_{R_o} \cdot M_D$	M_D/N_D
6	$-\hat{\rho}_x$	δD_{31}	$-\delta \hat{\rho}_x$	$K_{Vc} \cdot M_{R_o} \cdot N_D$	M_D/N_D	$K_{Vc} \cdot M_{R_o} \cdot M_D$	$K_{Vc} \cdot M_{R_o}$
7	D_{11}	$\delta \hat{\rho}_y$	δD_{11}	M_D	$K_{Vc} \cdot M_{R_o}$	$K_{Vc} \cdot M_{R_o} \cdot M_D$	M_D/N_D
8	$\hat{\rho}_y$	δD_{11}	$\delta \hat{\rho}_y$	$K_{Vc} \cdot M_{R_o} \cdot N_D$	M_D/N_D	$K_{Vc} \cdot M_{R_o} \cdot M_D$	$K_{Vc} \cdot M_{R_o}$
9	D_{21}	$-\delta \hat{\rho}_x$	δD_{21}	M_D	$K_{Vc} \cdot M_{R_o}$	$K_{Vc} \cdot M_{R_o} \cdot M_D$	M_D/N_D
10	$-\hat{\rho}_x$	δD_{21}	$-\delta \hat{\rho}_x$	$K_{Vc} \cdot M_{R_o} \cdot N_D$	M_D/N_D	$K_{Vc} \cdot M_{R_o} \cdot M_D$	$K_{Vc} \cdot M_{R_o}$

Table 4.7: Integrator scale factors of the DDA implementation of the DCM **D** terms, the DCM that transforms from earth-centered earth-fixed frame of reference to the computational frame of reference. These relations are found from the corresponding integrators as presented in section 3.4.3.

The basic relations between the inputs and output for each integrators are $K_{z_i} = \frac{M_{z_i}}{N_i}$, $K_{y_i} = \frac{M_{y_i}}{N_i}$ and $K_{z_i} = K_{z_i} \cdot K_{y_i} \cdot N_i = K_{z_i} \cdot M_{y_i}$, where M is the supremum of the corresponding input.

From the relations presented in table 4.7, the scale factor for the D_{ij} terms is

$$K_D = K_{Vc} \cdot M_{R_o} \cdot M_D = \frac{M_D}{N_D} \quad (4.30)$$

therefore

$$K_{V_c} M_{R_o} = \frac{1}{N_D} \quad (4.31)$$

where M_D is the supremum of the D_{ij} terms of the DCM $\mathbf{D}$, and $M_{R_o} \geq \frac{1}{R_o}$, where R_o is the earth radius.

In section 4.4 will be shown that $M_D = 1$ as the supremum of the DCM $\mathbf{Q}$ terms.

4.3.4 True Angular Velocity

The true angular velocity relative to the computational reference frame is found from the DCM $\mathbf{Q}$, the angular rates from the gyros in the body frame of reference, the damped angular rates and the earth rate in the computational frame Ω_c

$$\mu_c = \mathbf{Q} \cdot \omega - (\rho_c + \Omega_c) \quad (4.32)$$

where:

- ρ_c is the damped angular rate of the computational frame of reference with respect to the earth-centered earth-fixed frame of reference. In reference [Siouris 93] the not damped rate is also called *transport rate*, $\hat{\rho}_x = -\frac{V_y}{R_o}$, $\hat{\rho}_y = \frac{V_x}{R_o}$ and $\hat{\rho}_z = 0$, as presented in section 4.3.3;
- Ω_c is the earth rate relative to the earth-centered inertial frame of reference transformed to the computational frame of reference, $\Omega_c = \mathbf{D} \cdot \Omega_{ie}$, where $\mathbf{D}$ is the DCM that transforms from the earth-fixed frame of reference to the computational frame of reference, as presented in section 4.3.3.

So the true angular velocity components relative to the computational frame of reference are

$$\begin{aligned}
 \mu_x &= Q_{11}.\omega_x + Q_{12}.\omega_y + Q_{13}.\omega_z - (\rho_x + D_{12}.\Omega_o) \\
 \mu_y &= Q_{21}.\omega_x + Q_{22}.\omega_y + Q_{23}.\omega_z - (\rho_y + D_{22}.\Omega_o) \\
 \mu_z &= Q_{31}.\omega_x + Q_{32}.\omega_y + Q_{33}.\omega_z - (\rho_z + D_{32}.\Omega_o)
 \end{aligned} \tag{4.33}$$

Figure 4.7 presents the implementation of the true angular velocity components relative to the computational frame of reference in the DDA approach.

Figure 4.7: DDA implementation of equations 4.33, the true angular velocity components relative to the computational frame of reference.

The topology of the integrators in figure 4.7 (a), (b) and (c) are very similar to each other, so the results presented in table 4.8, the integrator scale factors of figure 4.7 (a), can also be applied to the integrators in figures 4.7 (b) and (c).

Integrators	Y Registers	ΔX	ΔY	M	K_x	K_z	K_y
1	ω_x	δQ_{11}	$\delta \omega_x$	$K_\omega \cdot N_\mu$	K_Q	$K_Q \cdot K_\omega \cdot N_\mu$	K_ω
2	Q_{11}	$\delta \omega_x$	δQ_{11}	$K_Q \cdot N_\mu$	K_ω	$K_Q \cdot K_\omega \cdot N_\mu$	K_Q
3	ω_y	δQ_{12}	$\delta \omega_y$	$K_\omega \cdot N_\mu$	K_Q	$K_Q \cdot K_\omega \cdot N_\mu$	K_ω
4	Q_{12}	$\delta \omega_y$	δQ_{12}	$K_Q \cdot N_\mu$	K_ω	$K_Q \cdot K_\omega \cdot N_\mu$	K_Q
5	ω_z	δQ_{13}	$\delta \omega_z$	$K_\omega \cdot N_\mu$	K_Q	$K_Q \cdot K_\omega \cdot N_\mu$	K_ω
6	Q_{13}	$\delta \omega_z$	δQ_{13}	$K_Q \cdot N_\mu$	K_ω	$K_Q \cdot K_\omega \cdot N_\mu$	K_Q
7	Ω_o	δD_{12}	-	M_Ω	K_D	$M_\Omega \cdot K_D$	M_Ω / N_D

Table 4.8: Integrator scale factors of the DDA implementation of equations 4.33, the true angular velocity component μ_{cz} referenced to the computational frame of reference. These relations can be found from the corresponding integrators as presented in section 3.4.3.

The basic relations between the inputs and output for each integrators are $K_{x_i} = \frac{M_{x_i}}{N_i}$, $K_{y_i} = \frac{M_{y_i}}{N_i}$ and $K_{z_i} = K_{x_i} \cdot K_{y_i} \cdot N_i = K_{x_i} \cdot M_{y_i}$, where M is the supremum of the corresponding input.

From the relation between the integrators

$$K_{z1} = K_{z2} = K_{z3} = K_{z4} = K_{z5} = K_{z6} = K_{z7}$$

the following relations are found

$$K_\mu = K_Q \cdot K_\omega \cdot N_\mu = \frac{M_\mu}{N_\mu} \quad (4.34)$$

$$M_\Omega = \frac{K_\omega \cdot K_Q \cdot N_\mu}{K_D}, \text{ and } M_\Omega \geq \Omega_o \quad (4.35)$$

$$K_\mu = K_\rho \quad (4.36)$$

where:

- K_Q is the integrator scale factor of the DCM Q terms presented in section 4.3.2;

- K_ω is the integrator scale factor of the compensated angular velocity presented in section 4.2.2;
- K_ρ is the integrator scale factor of the damped angular rate presented in section 4.3.1.

4.3.5 True Acceleration

This section will focus on the DDA integrator topology for the transformation of the acceleration compensated $\mathbf{A}^T = \begin{bmatrix} a_x & a_y & a_z \end{bmatrix}$ from the body frame of reference to the computational frame of reference.

From section 4.3.2, $\mathbf{Q}$ is the DCM that transforms coordinates from body frame of reference to computational frame of reference, so

$$\mathbf{A}_c = \mathbf{Q} \cdot \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} \quad (4.37)$$

Figure 4.8 shows the topology for the transformation. Table 4.9 presents the scale factors for the integrators of the component increment δA_{cx} . As all three axes are analogous, the results in table 4.9 can also be applied to the components δA_{cy} and δA_{cz} .

From table 4.9 the relation between the integrator scale factors of the true acceleration components and the compensated acceleration components is

$$K_{Ac} = \frac{M_{Ac}}{N_{Ac}} = K_Q \cdot K_a \cdot N_{Ac} \quad (4.38)$$

Figure 4.8: DDA implementation of equation 4.37 the true acceleration components in the computational frame of reference.

Integrators	Y Registers	ΔX	ΔY	M	K_x	K_z	K_y
1	a_x	δQ_{11}	δa_x	$K_a \cdot N_{Ac}$	K_Q	$K_Q \cdot K_a \cdot N_{Ac}$	K_a
2	Q_{11}	δa_x	δQ_{11}	$K_Q \cdot N_{Ac}$	K_a	$K_Q \cdot K_a \cdot N_{Ac}$	K_Q
3	a_y	δQ_{12}	δa_y	$K_a \cdot N_{Ac}$	K_Q	$K_Q \cdot K_a \cdot N_{Ac}$	K_a
4	Q_{12}	δa_y	δQ_{12}	$K_Q \cdot N_{Ac}$	K_a	$K_Q \cdot K_a \cdot N_{Ac}$	K_Q
5	a_z	δQ_{13}	δa_z	$K_a \cdot N_{Ac}$	K_Q	$K_Q \cdot K_a \cdot N_{Ac}$	K_a
6	Q_{13}	δa_z	δQ_{13}	$K_Q \cdot N_{Ac}$	K_a	$K_Q \cdot K_a \cdot N_{Ac}$	K_Q

Table 4.9: True acceleration integrator scale factor relations for the x-axis component. From figure 4.8 these relations can be found from the corresponding integrators as presented in section 3.4.3.

The basic relations between the inputs and output for each integrators are $K_{x_i} = \frac{M_{x_i}}{N_i}$, $K_{y_i} = \frac{M_{y_i}}{N_i}$ and $K_{z_i} = K_{x_i} \cdot K_{y_i} \cdot N_i = K_{x_i} \cdot M_{y_i}$, where M is the supremum of the corresponding input.

where:

- K_Q is the integrator scale factor of the DCM Q terms presented in section 4.3.2;
- K_a is the integrator scale factor of the compensated acceleration presented in section 4.2.1.

4.3.6 Quaternions

Reference [Staley 98] shows that the values of the quaternion derivatives can be obtained from the 4×4 skew-symmetric matrix, as in equation 4.39

$$\begin{bmatrix} \frac{d}{dt}(q_1) \\ \frac{d}{dt}(q_2) \\ \frac{d}{dt}(q_3) \\ \frac{d}{dt}(q_4) \end{bmatrix} = \frac{1}{2} \cdot \begin{bmatrix} 0 & \mu_z & -\mu_y & \mu_x \\ -\mu_z & 0 & \mu_x & \mu_y \\ \mu_y & -\mu_x & 0 & \mu_z \\ -\mu_x & -\mu_y & -\mu_z & 0 \end{bmatrix} \cdot \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} \quad (4.39)$$

Taking the term $\frac{d}{dt}(q_1)$

$$\begin{aligned} 2 \cdot \frac{d}{dt}(q_1) &= \mu_z \cdot q_2 - \mu_y \cdot q_3 + \mu_x \cdot q_4 \\ \therefore 2 \cdot dq_1 &= \mu_z \cdot (q_2 \cdot dt) - \mu_y \cdot (q_3 \cdot dt) + \mu_x \cdot (q_4 \cdot dt) \end{aligned} \quad (4.40)$$

Approximating the differentials dq_i and dt to δq_i and δt and applying it to a DDA integrator as in figure 4.9(a), the term $q_i \cdot \delta t$ will be the output δz . With this in mind, using $q_i \delta t$ as the δx input and $\delta \mu_j$ as the δy input, the terms $\mu_j \cdot (q_i \cdot \delta t)$ are obtained as the output of the DDA integrator presented in figure 4.40(b), where $j = x, y, z$ and $i = 1, 2, 3, 4, .$

Figure 4.9: Simple DDA integrator of a term with output equal to $q_i \cdot \delta t$ as in equation 4.40.

Applying the integrators in figure 4.9, the DDA implementation of equation 4.39 is obtained as shown in figure 4.10, where:

- M_q is the supremum of q_i the quaternions, with $i = 1, 2$ and 3 ;
- M_{2q} is the supremum of the constant multiplier factor $\frac{1}{2}$: $M_{2q} \geq \frac{1}{2}$;
- N_q is the size of the registers Y and R of the integrators;
- from section 4.3.4 equations 4.34 and 4.36 $K_\mu = K_\omega \cdot K_Q \cdot N_\mu = K_\rho$.

The four topologies corresponding to q_1 , q_2 , q_3 and q_4 are very similar, varying just in the signal of some multipliers output, which can be achieved changing the signal of the increments. With this in mind, table 4.10 is presented with the integrator scale factors for the q_1 implementation. Figure 4.10 (a), and its results can be applied to all the others quaternions implementations.

Integrators	Y Registers	ΔX	ΔY	M	K_x	K_z	K_y
1	q_2	δt	δq_2	M_q	K_t	$K_t \cdot M_q$	M_q / N_q
2	μ_z	$q_2 \cdot \delta t$	$\delta \mu_z$	$K_\mu \cdot N_q$	$K_t \cdot M_q$	$K_t \cdot K_\mu \cdot M_q \cdot N_q$	K_μ
3	q_3	δt	δq_3	M_q	K_μ	$K_t \cdot M_q$	M_q / N_q
4	μ_y	$q_3 \cdot \delta t$	$\delta \mu_y$	$K_\mu \cdot N_q$	$K_t \cdot M_q$	$K_t \cdot K_\mu \cdot M_q \cdot N_q$	K_μ
5	q_4	δt	δq_4	M_q	K_μ	$K_t \cdot M_q$	M_q / N_q
6	μ_x	$q_4 \cdot \delta t$	$\delta \mu_x$	$K_\mu \cdot N_q$	$K_t \cdot M_q$	$K_t \cdot K_\mu \cdot M_q \cdot N_q$	K_μ
7	$1/2$	$2 \cdot \delta q_1$	-	M_{2q}	$K_t \cdot K_\mu \cdot M_q \cdot N_q$	$M_{2q} \cdot K_t \cdot K_\mu \cdot M_q \cdot N_q$	M_{2q} / N_q

Table 4.10: Integrator scale factors of the DDA implementation of the quaternions that relates the computational frame of reference and the system frame of reference presented in equation 4.39.

Figure 4.10: Quaternions' differential equation 4.39 implemented in a DDA form using the integrators presented in figure 4.9. These are the quaternions that relates the computational frame of reference and the system frame of reference.

From the relations between the integrators

$$K_q = M_{2q} \cdot K_t \cdot K_\mu \cdot M_q \cdot N_q \quad (4.41)$$

4.3.7 True Velocity

The following equations of the true velocities relative to the computational frame of reference can be found in [Bolduc 95] and [Siouris 93].

$$\begin{aligned} \dot{V}_x &= A_{cx} - (\rho_y + 2\Omega \cdot D_{22}) \cdot V_z + (2\Omega \cdot D_{32}) \cdot V_y + \delta(R)_x \\ \dot{V}_y &= A_{cy} - (2\Omega \cdot D_{32}) \cdot V_x + (\rho_x + 2\Omega \cdot D_{12}) \cdot V_z + \delta(R)_y \\ \dot{V}_z &= A_{cz} - (\rho_x + 2\Omega \cdot D_{12}) \cdot V_y + (\rho_y + 2\Omega \cdot D_{22}) \cdot V_x + \delta(R)_z \end{aligned} \quad (4.42)$$

Assuming the simple gravitational model presented in [Siouris 93], equation 4.42 is simplified to

$$\begin{aligned} \dot{V}_x &= A_{cx} - (\rho_y + 2\Omega \cdot D_{22}) \cdot V_z + (2\Omega \cdot D_{32}) \cdot V_y \\ \dot{V}_y &= A_{cy} - (2\Omega \cdot D_{32}) \cdot V_x + (\rho_x + 2\Omega \cdot D_{12}) \cdot V_z \\ \dot{V}_z &= A_{cz} - (\rho_x + 2\Omega \cdot D_{12}) \cdot V_y + (\rho_y + 2\Omega \cdot D_{22}) \cdot V_x - g_z \end{aligned} \quad (4.43)$$

and is implemented in a DDA form as in figure 4.11.

Since the topologies of the DDA implementations of the components of the true velocity $\mathbf{V}_e$ are very similar, the results found for the component δV_{cx} in table 4.11 can also be applied to the components δV_{cy} and δV_{cz} , where:

- $M_{2\Omega} \geq 2\Omega_{ie}$, where Ω_{ie} is the earth rate;

Figure 4.11: True velocity differential equations implementation in a DDA form.

- M_{V_c} is the supremum of the true velocities that the system can measure;
- $M_{\dot{V}_c}$ is the supremum of the derivative of the true velocities;
- $M_{\rho D}$ is the supremum of the terms $(2.\Omega.D_{ij} + \rho_k)$, $k = \{x, y\}$ and $i, j = \{1, 2, 3\}$.

From table 4.3 $M_{\rho D} = K_D.M_{2\Omega}.N_{V_c} = K_{\rho}.N_{V_c}$;

- N_{V_c} is the size of the registers Y and R of the integrators.

Int.	Y Registers	ΔX	ΔY	M	K_z	K_z	K_y
1	$2.\Omega$	δD_{22}	-	$M_{2\Omega}$	K_D	$K_D.M_{2\Omega}$	$M_{2\Omega}/N_{V_c}$
2	$2.\Omega$	δD_{32}	-	$M_{2\Omega}$	K_D	$K_D.M_{2\Omega}$	$M_{2\Omega}/N_{V_c}$
3	V_z	$2.\Omega.\delta D_{22} + \delta\rho_y$	δV_z	M_{V_c}	$K_D.M_{2\Omega}$	$K_D.M_{2\Omega}.M_{V_c}$	K_{V_c}
4	$2.\Omega.\delta D_{22} + \delta\rho_y$	δV_z	$2.\Omega.\delta D_{22} + \delta\rho_y$	$M_{\rho D}$	K_{V_c}	$K_D.M_{2\Omega}.M_{V_c}$	$K_D.M_{2\Omega}$
5	V_y	$2.\Omega.\delta D_{32}$	δV_y	M_{V_c}	$K_D.M_{2\Omega}$	$K_D.M_{2\Omega}.M_{V_c}$	K_{V_c}
6	$2.\Omega.D_{32}$	δV_y	$2.\Omega.\delta D_{32}$	$M_{\rho D}$	K_{V_c}	$K_D.M_{2\Omega}.M_{V_c}$	$K_D.M_{2\Omega}$
7	A_{cz}	δt	δA_{cz}	$K_{A_c}.N_{V_c}$	K_t	$K_t.K_{A_c}.N_{V_c}$	K_{A_c}
8	$(2.\Omega.\delta D_{22} + \delta\rho_y).V_z$	δt	$\delta [(2.\Omega.\delta D_{22} + \delta\rho_y).V_z]$	$K_D.M_{2\Omega}.M_{V_c}.N_{V_c}$	K_t	$K_t.K_D.M_{2\Omega}.M_{V_c}.N_{V_c}$	$K_D.M_{2\Omega}.M_{V_c}$
9	$(2.\Omega.D_{32}).V_y$	δt	$\delta [(2.\Omega.D_{32}).V_y]$	$K_D.M_{2\Omega}.M_{V_c}.N_{V_c}$	K_t	$K_t.K_D.M_{2\Omega}.M_{V_c}.N_{V_c}$	$K_D.M_{2\Omega}.M_{V_c}$

Table 4.11: Integrator scale factors of the DDA implementation of the component δV_{cx} of the true velocity , figure 4.11 (a).

The following relations between the integrator scale factors can be found from figure 4.11(a).

- from integrator 1:

$$\left. \begin{array}{l} K_{x_1} = K_D \\ K_{y_1} = M_{2\Omega}/N_{V_c} \end{array} \right| \Rightarrow K_{x_1} = K_D.M_{2\Omega} = K_{\rho} \quad (4.44)$$

- from integrator 2:

$$\left. \begin{array}{l} K_{x_2} = K_D \\ K_{y_2} = M_{2\Omega}/N_{Vc} \end{array} \right| \Rightarrow K_{z_2} = K_D \cdot M_{2\Omega} \quad (4.45)$$

- from integrators 3 and 4:

$$\left. \begin{array}{l} K_{x_3} = K_{y_4} = K_{z_1} = K_D \cdot M_{2\Omega} = K_\rho \\ K_{y_3} = K_{x_4} = K_{Vc} \end{array} \right| \Rightarrow \begin{array}{l} K_{z_3} = K_{z_4} = K_\rho \cdot M_{Vc} \\ = K_D \cdot M_{2\Omega} \cdot M_{Vc} \end{array} \quad (4.46)$$

- from integrators 5 and 6:

$$\left. \begin{array}{l} K_{x_5} = K_{y_6} = K_{z_2} = K_D \cdot M_{2\Omega} \\ K_{y_5} = K_{x_6} = K_{Vc} \end{array} \right| \Rightarrow \begin{array}{l} K_{z_5} = K_{z_6} = K_D \cdot M_{2\Omega} \cdot M_{Vc} \end{array} \quad (4.47)$$

- from integrator 7:

$$\left. \begin{array}{l} K_{x_7} = K_t \\ K_{y_7} = K_{Ac} \end{array} \right| \Rightarrow K_{z_7} = K_t \cdot K_{Ac} \cdot N_{Vc} \quad (4.48)$$

- from integrator 8:

$$\left. \begin{array}{l} K_{x_8} = K_t \\ K_{y_8} = K_{z_3} = K_D \cdot M_{2\Omega} \cdot M_{Vc} \end{array} \right| \Rightarrow K_{z_8} = K_t \cdot K_D \cdot M_{2\Omega} \cdot M_{Vc} \cdot N_{Vc} \quad (4.49)$$

- from integrator 9:

$$\left. \begin{aligned} K_{z_9} &= K_t \\ K_{y_9} = K_{z_3} &= K_D \cdot M_{2\Omega} \cdot M_{Vc} \end{aligned} \right| \Rightarrow K_{z_9} = K_t \cdot K_D \cdot M_{2\Omega} \cdot M_{Vc} \cdot N_{Vc} \quad (4.50)$$

from equations 4.44, 4.45, 4.46, 4.47, 4.48, 4.49, 4.50 and table 4.11

$$K_\rho = K_D \cdot M_{2\Omega} \quad (4.51)$$

$$K_{Ac} = K_D \cdot M_{2\Omega} \cdot M_{Vc} = K_\rho \cdot M_{Vc} \quad (4.52)$$

$$M_{Vc} = \frac{K_{Ac}}{K_D \cdot M_{2\Omega}} = K_t \cdot K_{Ac} \cdot N_{Vc}^2 \quad (4.53)$$

$$N_{Vc} = \sqrt{\frac{M_{Vc}}{K_t \cdot K_{Ac}}} = \sqrt{\frac{1}{K_t \cdot K_D \cdot M_{2\Omega}}} \quad (4.54)$$

Following the approach presented in [Bolduc 95], the simple gravitational model has been applied which resulted in the initial value of integrator 7 to be $g_m = -9.806$ scaled by $K_{y_7} = \frac{K_{Vc}}{N_{Vc}}$ from figure 4.11 (c).

4.4 Orthogonalization

In sections 4.3.6, 4.3.2 and 4.3.3 the results must be orthogonalized in order to minimize error propagation. As has been done in the previous sections, the same approach that was applied in [Bolduc 95] will be implemented in a DDA form in the following sections.

4.4.1 Quaternions

To orthogonalize the quaternions let

$$q_1^2 + q_2^2 + q_3^2 + q_4^2 = 1 + \epsilon$$

so the orthogonalized quaternion is

$$\bar{q}_i = \frac{q_i}{\sqrt{1 + \epsilon}}$$

Assuming $\epsilon \ll 1$ and taking the first term of the Taylor expansion

$$\bar{q}_i \approx q_i \cdot \left(1 - \frac{\epsilon}{2}\right) \quad (4.55)$$

finding the differential of $\bar{q}_i$

$$\delta \bar{q}_i \approx \delta q_i - \delta q_i \cdot \frac{\epsilon}{2} - q_i \cdot \frac{\delta \epsilon}{2} \quad (4.56)$$

and the differential of ϵ

$$\delta \epsilon = 2 \cdot q_1 \cdot \delta q_1 + 2 \cdot q_2 \cdot \delta q_2 + 2 \cdot q_3 \cdot \delta q_3 + 2 \cdot q_4 \cdot \delta q_4 \quad (4.57)$$

With the equations 4.56 and 4.57, the DDA approach for the orthogonalization of the quaternions is presented in figure 4.12.

From the relations between the integrator scale factors shown in table 4.12 the

Figure 4.12: DDA implementation of the orthogonalization of the quaternions based on the Taylor expansion of the error ϵ of the orthogonalization, as presented in equation 4.57.

Integrator	Y Register	ΔX	ΔY	M	K_x	K_z	K_y
1	q_1	δq_1	δq_1	M_q	M_q/N_o	M_q^2/N_o	M_q/N_o
2	q_2	δq_2	δq_2	M_q	M_q/N_o	M_q^2/N_o	M_q/N_o
3	q_3	δq_3	δq_3	M_q	M_q/N_o	M_q^2/N_o	M_q/N_o
4	q_4	δq_4	δq_4	M_q	M_q/N_o	M_q^2/N_o	M_q/N_o
5	$\frac{\epsilon}{2}$	δq_1	$\frac{\delta \epsilon}{2}$	$M_{\frac{\epsilon}{2}}$	M_q/N_o	$M_q \cdot M_{\frac{\epsilon}{2}}/N_o$	$M_{\frac{\epsilon}{2}}/N_o$
6	q_1	$\frac{\delta \epsilon}{2}$	δq_1	M_q	$M_{\frac{\epsilon}{2}}/N_o$	$M_q \cdot M_{\frac{\epsilon}{2}}/N_o$	M_q/N_o
7	$\frac{\epsilon}{2}$	δq_2	$\frac{\delta \epsilon}{2}$	$M_{\frac{\epsilon}{2}}$	M_q/N_o	$M_q \cdot M_{\frac{\epsilon}{2}}/N_o$	$M_{\frac{\epsilon}{2}}/N_o$
8	q_2	$\frac{\delta \epsilon}{2}$	δq_2	M_q	$M_{\frac{\epsilon}{2}}/N_o$	$M_q \cdot M_{\frac{\epsilon}{2}}/N_o$	M_q/N_o
9	$\frac{\epsilon}{2}$	δq_3	$\frac{\delta \epsilon}{2}$	$M_{\frac{\epsilon}{2}}$	M_q/N_o	$M_q \cdot M_{\frac{\epsilon}{2}}/N_o$	$M_{\frac{\epsilon}{2}}/N_o$
10	q_3	$\frac{\delta \epsilon}{2}$	δq_3	M_q	$M_{\frac{\epsilon}{2}}/N_o$	$M_q \cdot M_{\frac{\epsilon}{2}}/N_o$	M_q/N_o
11	q_4	$\frac{\delta \epsilon}{2}$	δq_4	M_q	$M_{\frac{\epsilon}{2}}/N_o$	$M_q \cdot M_{\frac{\epsilon}{2}}/N_o$	M_q/N_o
12	$\frac{\epsilon}{2}$	δq_4	$\frac{\delta \epsilon}{2}$	$M_{\frac{\epsilon}{2}}$	M_q/N_o	$M_q \cdot M_{\frac{\epsilon}{2}}/N_o$	$M_{\frac{\epsilon}{2}}/N_o$

Table 4.12: Integrator scale factors of the DDA implementation of the orthogonalization of the quaternions, as presented in figure 4.12, based on the Taylor expansion presented in equations 4.56 and 4.57.

following relation must be satisfied

$$M_q = M_q^2 = M_q \cdot M_{\frac{\pi}{2}}$$

the constraints that follow are found to be

$$M_q = 1 \quad (4.58)$$

$$M_{\frac{\pi}{2}} = 1 \quad (4.59)$$

4.4.2 Direction Cosine Matrices

D and **Q** are DCM that transform coordinates from the earth-centered earth-fixed frame of reference to the computational frame of reference and from the system frame of reference to the navigational frame of reference, respectively. All DCM matrices are orthogonal and so their determinants must be equal to 1. The determinant of the DCM **D** can be approximated using the same approach presented in the previous section:

$$\begin{aligned} & D_{11} \cdot D_{22} \cdot D_{33} + D_{12} \cdot D_{23} \cdot D_{31} + D_{21} \cdot D_{32} \cdot D_{13} \\ & - D_{13} \cdot D_{22} \cdot D_{31} - D_{12} \cdot D_{21} \cdot D_{33} - D_{11} \cdot D_{23} \cdot D_{32} = 1 + \epsilon \end{aligned}$$

So the normalized element of the DCM **D** is

$$\bar{D}_{ij} = \frac{D_{ij}}{1 + \epsilon}$$

Assuming $\epsilon \ll 1$ and taking the first term of the Taylor expansion, the normalized

element of DCM $\mathbf{D}$ can be approximated to

$$\bar{D}_{ij} \approx D_{ij} \cdot (1 - \epsilon)$$

With the same approach presented to the quaternions, but now for each element of the DCM

$$\delta \bar{D}_{ij} \approx \delta [D_{ij} \cdot (1 - \epsilon)] \quad (4.60)$$

where $i, j = 1, 2, 3$.

As was presented in chapter 3.4 for the DDA implementation of the product of two functions, the differential of each function is needed. In equation 4.60 δD_{ij} is known, so $\delta(1 - \epsilon)$ is

$$\begin{aligned} \delta(1 - \epsilon) &= -\delta\epsilon \\ \delta\epsilon &= \delta(-D_{11} \cdot D_{22} \cdot D_{33} - D_{12} \cdot D_{23} \cdot D_{31} - D_{21} \cdot D_{32} \cdot D_{13} \\ &\quad + D_{13} \cdot D_{22} \cdot D_{31} + D_{12} \cdot D_{21} \cdot D_{33} + D_{11} \cdot D_{23} \cdot D_{32}) \end{aligned} \quad (4.61)$$

Figure 4.13 (a) is the implementation of $\delta\epsilon$. The normalized terms of the DCM are found by applying the multiplier DDA presented in figure 4.13 (b) where $i, j = 1, 2, 3$.

The same results are valid for the Q DCM matrix, so they are generically applied to every DCM.

All the integrators are products of 3 differential terms. Table 4.13 presents the integrator scale factors for the first set of multipliers, the result is applicable to all the others since they have the same topology.

Figure 4.13: DDA implementation of the error term $\delta\epsilon$ of the orthogonalization of the DCM elements as presented in equation 4.61. In this implementation only the first order term of the Taylor expansion has been taken into account.

Integrator	Y Register	ΔX	ΔY	M	K_x	K_z	K_y
1	D_{22}	δD_{11}	δD_{22}	M_D	M_D/N_o	M_D^2/N_o	M_D/N_o
2	D_{11}	δD_{22}	δD_{11}	M_D	M_D/N_o	M_D^2/N_o	M_D/N_o
13	D_{33}	$\delta D_{11} \cdot \delta D_{22}$	δD_{31}	M_D	M_D^2/N_o	M_D^3/N_o	M_D/N_o
14	$D_{11} \cdot D_{22}$	δD_{33}	$\delta D_{11} \cdot \delta D_{22}$	M_D	M_D/N_o	M_D^3/N_o	M_D^2/N_o

Table 4.13: Integrator scale factors of the DDA implementation of the orthogonalization of the elements of the DCM as presented in figure 4.13 based on the Taylor expansion presented in equation 4.60.

In figure 4.13 (b) the integrator scale factor is

$$K_z = K_\epsilon \cdot K_D \cdot N_o$$

The scale factor of $\delta \bar{D}_{ij}$ must be equal to the scale factor of δD_{ij} which implies

$$M_D = K_\epsilon \cdot K_D \cdot N_o^2 = M_D^3 \cdot M_D$$

$$\therefore M_D = M_D^4$$

the constraint that follows are found to be

$$M_D = 1 \quad (4.62)$$

4.5 Constraints

Sections 4.2.1 to 4.4 presented the DDA implementations for the compensators, alignment and navigation algorithms and in each one some constraints were found. Now, in this section, all these constraints are put together in order to reach more general constraints applied to the system as a whole. All constraints found are listed in table

4.14.

Section	Constraints
4.2.1	$a_{max} = M_k \cdot a_{rmax}$
4.2.2	$M_{kw} \cdot K_w = M_{ka} \cdot K_a$ $\omega_{max} = M_{ka} \cdot a_{max} \cdot \frac{N_\mu}{N_a} = M_{kw} \cdot \omega_{rmax}$
4.3.1	$M_\tau = \frac{1}{K_t \cdot N_p} \geq \frac{1}{\tau}$ $K_\rho = K_{Vc} \cdot M_R$ $M_{act} = \frac{K_{Vc} \cdot M_R}{K_{Ac}} \geq a$
4.3.2	$K_Q = M_2 \cdot K_q \cdot M_q$
4.3.3	$K_D = K_{Vc} \cdot M_{Ro} \cdot M_D$ $N_D = \frac{N_{Vc}}{M_{Vc} \cdot M_{Ro}}$
4.3.4	$K_\mu = K_Q \cdot K_w \cdot N_\mu = K_\rho$ $M_\Omega = \frac{K_w \cdot K_Q \cdot N_\mu}{K_D}$, and $M_\Omega \geq \Omega_o$
4.3.5	$K_{Ac} = K_Q \cdot K_a \cdot N_{Ac}$
4.3.6	$K_q = M_{2q} \cdot K_t \cdot K_\mu \cdot M_q \cdot N_q$
4.3.7	$K_{Ac} = K_D \cdot M_{2\Omega} \cdot M_{Vc} = K_\rho \cdot M_{Vc}$ $M_{Vc} = \frac{K_{Ac}}{K_D \cdot M_{2\Omega}} = K_t \cdot K_{Ac} \cdot N_{Vc}^2$ $N_{Vc} = \sqrt{\frac{M_{Vc}}{K_t \cdot K_{Ac}}} = \sqrt{\frac{1}{K_t \cdot K_D \cdot M_{2\Omega}}}$
4.4	$M_q = 1$ $M_D = M_Q = 1$

Table 4.14: This table summarizes all the constraints found from the relations between the integrators presented in the previous sections. They have been implemented in the software as part of the initialization process or as a constraint verification for each integration step.

From sections 4.3.2, 4.3.4 and 4.3.6

$$\begin{aligned}
 \frac{M_q}{N_q} &= M_{2q} \cdot K_t \cdot K_\mu \cdot M_q \cdot N_q = M_{2q} \cdot K_t \cdot K_Q \cdot K_w \cdot N_\mu \cdot M_q \cdot N_q \\
 \frac{1}{N_q} &= M_{2q} \cdot K_t \cdot K_Q \cdot K_w \cdot N_\mu \cdot N_q \\
 \therefore N_\mu &= \frac{N_w}{M_{2q} \cdot K_t \cdot K_Q \cdot \omega_{max} \cdot N_q^2}
 \end{aligned} \tag{4.63}$$

From section 4.3.2

$$K_Q = M_2 \cdot K_q \cdot M_q \Leftrightarrow \frac{M_Q}{N_Q} = M_2 \cdot \frac{M_q}{N_q} \cdot M_q$$

but from section 4.4

$$M_Q = 1$$

$$M_q = 1$$

so

$$\frac{1}{N_Q} = M_2 \cdot \frac{1}{N_q} \Rightarrow N_q = M_2 \cdot N_Q \quad (4.64)$$

As N_Q and N_q are powers of 2, M_2 is chosen to be $M_2 = 2 \geq 2$, which results in $N_q = 2 \cdot N_Q$.

With equations 4.63 and 4.64

$$\begin{aligned} N_\mu &= \frac{N_Q \cdot N_\omega}{M_{2q} \cdot K_t \cdot \omega_{max} \cdot N_q^2} = \frac{N_Q \cdot N_\omega}{M_{2q} \cdot K_t \cdot \omega_{max} \cdot M_2^2 \cdot N_Q^2} \\ \therefore N_\mu &= \frac{N_\omega}{M_{2q} \cdot K_t \cdot \omega_{max} \cdot M_2^2 \cdot N_Q} \end{aligned} \quad (4.65)$$

From section 4.3.7 and 4.3.4

$$\begin{aligned} K_\rho &= K_D \cdot M_{2\Omega} = K_Q \cdot K_\omega \cdot N_\mu \\ \therefore K_D &= \frac{K_Q \cdot K_\omega \cdot N_\mu}{M_{2\Omega}} \end{aligned}$$

but from section 4.4

$$M_Q = 1$$

$$M_D = 1$$

so

$$\frac{1}{N_D} = \frac{\omega_{max} \cdot N_\mu}{M_{2\Omega} \cdot N_Q \cdot N_w} \Leftrightarrow N_D = \frac{M_{2\Omega} \cdot N_Q \cdot N_w}{\omega_{max} \cdot N_\mu}$$

with equations 4.63, 4.64 and section 4.4

$$\begin{aligned} N_D &= \frac{M_{2\Omega} \cdot N_Q \cdot N_w}{\omega_{max}} \cdot \frac{M_{2q} \cdot K_t \cdot \omega_{max} \cdot N_q^2}{N_Q \cdot N_w} \\ &= M_{2\Omega} \cdot M_{2q} \cdot K_t \cdot N_q^2 \\ \therefore N_D &= M_{2\Omega} \cdot M_{2q} \cdot K_t \cdot M_2^2 \cdot N_Q^2 \end{aligned} \quad (4.66)$$

From sections 4.3.7, 4.3.5 and 4.4

$$\begin{aligned} M_{Vc} &= \frac{K_{Ac}}{K_D \cdot M_{2\Omega}} = \frac{K_Q \cdot K_a \cdot N_{Ac}}{K_D \cdot M_{2\Omega}} \\ \therefore M_{Vc} &= \frac{N_D \cdot a_{max} \cdot N_{Ac}}{M_{2\Omega} \cdot N_Q \cdot N_a} \end{aligned} \quad (4.67)$$

From sections 4.3.5, 4.3.1, 4.4 and equations 4.66, 4.67

$$\begin{aligned}
K_{Ac} &= \frac{1}{N_Q} \cdot \frac{a_{max}}{N_a} \cdot N_{Ac} = \frac{M_{Vc} \cdot M_R}{N_{Vc} \cdot M_{act}} \\
N_{Ac} &= \frac{M_R \cdot N_a \cdot N_Q}{M_{act} \cdot a_{max}} \cdot \frac{M_{Vc}}{N_{Vc}} = \frac{M_R \cdot N_a \cdot N_Q}{M_{act} \cdot a_{max}} \cdot \frac{N_D \cdot a_{max} \cdot N_{Ac}}{M_{2\Omega} \cdot N_Q \cdot N_a} \cdot \frac{1}{N_{Vc}} \\
N_{Vc} &= \frac{M_R \cdot N_D}{M_{act} \cdot M_{2\Omega}} = \frac{M_R}{M_{act} \cdot M_{2\Omega}} \cdot M_{2\Omega} \cdot M_{2q} \cdot K_t \cdot M_2^2 \cdot N_Q^2 \\
\therefore N_{Vc} &= \frac{M_R \cdot M_{2q} \cdot K_t \cdot M_2^2 \cdot N_Q^2}{M_{act}} \tag{4.68}
\end{aligned}$$

From sections 4.3.4, 4.3.1, 4.4

$$N_\mu = \frac{K_D \cdot M_\Omega}{K_\omega \cdot K_Q} = \frac{K_\rho}{K_Q \cdot K_\omega}$$

and from section 4.3.7 $K_\rho = K_D \cdot M_{2\Omega}$ so

$$K_D \cdot M_\Omega = K_D \cdot M_{2\Omega} \Rightarrow M_\Omega = M_{2\Omega} \tag{4.69}$$

From sections 4.3.4, 4.3.1, 4.4 and equations 4.63, 4.66, 4.67, 4.69

$$\begin{aligned}
N_\mu &= \frac{M_\Omega}{N_D \cdot K_\omega \cdot K_Q} = \frac{K_\rho}{K_Q \cdot K_\omega} \\
\frac{M_\Omega}{N_D} &= \frac{M_R \cdot M_{Vc}}{N_{Vc}} = M_R \cdot \frac{N_D \cdot a_{max} \cdot N_{Ac}}{M_{2\Omega} \cdot N_Q \cdot N_a} \cdot \frac{M_{act} \cdot M_{2\Omega}}{M_R \cdot N_D} \\
\frac{M_\Omega}{N_D} &= \frac{a_{max} \cdot N_{Ac} \cdot M_{act}}{N_Q \cdot N_a} \\
N_{Ac} &= \frac{M_\Omega}{M_{2\Omega} \cdot M_{2q} \cdot K_t \cdot M_2^2 \cdot N_Q^2} \cdot \frac{N_Q \cdot N_a}{a_{max} \cdot M_{act}} \\
\therefore N_{Ac} &= \frac{N_a}{M_{2q} \cdot K_t \cdot M_2^2 \cdot M_{act} \cdot a_{max} \cdot N_Q} \tag{4.70}
\end{aligned}$$

From equations 4.66, 4.67 and 4.70

$$M_{Vc} = \frac{a_{max}}{M_{2\Omega} \cdot N_Q \cdot N_a} \cdot M_2^2 \cdot M_{2\Omega} \cdot M_{2q} \cdot K_t \cdot N_Q^2 \cdot \frac{N_a}{M_{2q} \cdot K_t \cdot M_2^2 \cdot M_{act} \cdot a_{max} \cdot N_Q}$$

$$\therefore M_{Vc} = \frac{1}{M_{act}} \quad (4.71)$$

From section 4.3.1 and equations 4.68 and 4.71

$$K_\rho = M_R \cdot \frac{M_{Vc}}{N_{Vc}} = M_R \cdot \frac{1}{M_{act}} \cdot \frac{M_{act}}{M_2^2 \cdot M_R \cdot M_{2q} \cdot K_t \cdot N_Q^2}$$

$$\therefore K_\rho = \frac{1}{M_2^2 \cdot M_{2q} \cdot K_t \cdot N_Q^2} \quad (4.72)$$

but $N_\rho = \frac{1}{K_t \cdot M_\tau}$ so

$$M_\rho \cdot K_t \cdot M_\tau = \frac{1}{M_2^2 \cdot M_{2q} \cdot K_t \cdot N_Q^2}$$

$$\therefore M_\rho = \frac{1}{M_2^2 \cdot M_{2q} \cdot K_t^2 \cdot M_\tau \cdot N_Q^2} \quad (4.73)$$

From sections 4.3.4, 4.3.7 and equations 4.63, 4.66 and 4.69

$$K_\mu = K_Q \cdot K_\omega \cdot \frac{K_D \cdot M_\Omega}{K_\omega \cdot K_Q} = K_D \cdot M_\Omega$$

$$M_\mu = M_\Omega \cdot \frac{1}{M_{2\Omega} \cdot M_{2q} \cdot K_t \cdot M_2^2 \cdot N_Q^2} \cdot \frac{N_\omega}{M_{2q} \cdot K_t \cdot \omega_{max} \cdot M_2^2 \cdot N_Q}$$

$$\therefore M_\mu = \frac{N_\omega}{M_2^4 \cdot M_{2q}^2 \cdot K_t^2 \cdot \omega_{max} \cdot N_Q^3} \quad (4.74)$$

Also from section 4.3.4 and equations 4.72, 4.65

$$\begin{aligned}
 K_\mu &= K_\rho = \frac{1}{M_2^2 \cdot M_{2q} \cdot K_t \cdot N_Q^2} \\
 M_\mu &= \frac{N_\mu}{M_2^2 \cdot M_{2q} \cdot K_t \cdot N_Q^2} = \frac{1}{M_2^2 \cdot M_{2q} \cdot K_t \cdot N_Q^2} \cdot \frac{N_\omega}{M_{2q} \cdot K_t \cdot \omega_{max} \cdot M_2^2 \cdot N_Q} \\
 \therefore M_\mu &= \frac{N_\omega}{M_2^4 \cdot M_{2q}^2 \cdot K_t^2 \cdot \omega_{max} \cdot N_Q^3}
 \end{aligned}$$

which gives the same result.

From section 4.3.7 and equations 4.66 ,4.70, 4.71

$$\begin{aligned}
 M_{Ac} &= K_D \cdot M_{2\Omega} \cdot M_{Vc} \cdot N_{Ac} \\
 &= M_{2\Omega} \cdot \frac{1}{M_{2\Omega} \cdot M_{2q} \cdot K_t \cdot M_2^2 \cdot N_Q^2} \cdot \frac{1}{M_{act}} \cdot \frac{N_a}{M_{2q} \cdot K_t \cdot M_2^2 \cdot M_{act} \cdot a_{max} \cdot N_Q} \\
 \therefore M_{Ac} &= \frac{N_a}{M_2^4 \cdot M_{2q}^2 \cdot K_t^2 \cdot M_{act}^2 \cdot N_Q^3 \cdot a_{max}} \quad (4.75)
 \end{aligned}$$

but also from section 4.3.5 and equation 4.70

$$\begin{aligned}
 M_{Ac} &= K_Q \cdot K_a \cdot N_{Ac}^2 = \frac{a_{max}}{N_Q \cdot N_a} \cdot \left(\frac{N_a}{M_{2q} \cdot K_t \cdot M_2^2 \cdot M_{act} \cdot a_{max} \cdot N_Q} \right)^2 \\
 &= \frac{N_a}{M_2^4 \cdot M_{2q}^2 \cdot K_t^2 \cdot M_{act}^2 \cdot N_Q^3 \cdot a_{max}}
 \end{aligned}$$

which gives the same result.

To summarize the relations between the size of the integrators and the supremum values of the integrators found in this section, the table 4.15 has been generated.

All these relations are implemented in the DDA software as part of the initialization

N	Value	M	Value
N_q	$M_2 \cdot N_Q$	M_2	2
N_D	$M_{2\Omega} \cdot M_{2q} \cdot K_t \cdot M_2^2 \cdot N_Q^2$	M_D	1
N_μ	$\frac{N_\omega}{M_{2q} \cdot K_t \cdot M_2^2 \cdot \omega_{max} \cdot N_Q}$	M_μ	$\frac{N_\omega}{M_2^4 \cdot M_{2q}^2 \cdot K_t^2 \cdot \omega_{max} \cdot N_Q^3}$
N_{Vc}	$\frac{M_R \cdot M_{2q} \cdot K_t \cdot M_2^2 \cdot N_Q^2}{M_{act}}$	M_{Vc}	$\frac{1}{N_a}$
N_{Ac}	$\frac{M_{act} \cdot a_{max} \cdot N_Q}{M_{2q} \cdot K_t \cdot M_2^2 \cdot M_{act} \cdot a_{max} \cdot N_Q}$	M_{Ac}	$\frac{1}{M_2^4 \cdot M_{2q}^2 \cdot K_t^2 \cdot M_{act}^2 \cdot N_Q^3 \cdot a_{max}}$
N_ρ	$\frac{1}{M_\tau \cdot K_t}$	M_ρ	$\frac{1}{M_2^2 \cdot M_{2q} \cdot K_t^2 \cdot M_\tau \cdot N_Q^2}$
N_ω	defined	ω_{max}	$M_{k\omega} \cdot \omega_{rmax}$
N_a	defined	a_{max}	$M_k a_{rmax}$
N_Q	defined	M_Q	1
K_t	defined	$M_{2\Omega}$	M_Ω
		M_{ka}	$\frac{\omega_{max}}{a_{max}}$
		M_{act}	defined
		ω_{rmax}	defined
		a_{rmax}	defined

Table 4.15: This table summarizes the relations presented in section 4.5, the relations between the size of the registers and the supremum of the integrators in each of the algorithms' steps. These relations are implemented in the software as constraints.

process of each class and/or as a constraint verification step through the integrations.

Chapter 5

Software Implementation

The principal idea behind the use of a DDA is to implement the algorithms developed for a particular problem in hardware, in an FPGA or EPLD for example. The algorithms can then be run in parallel, improving performance. Since the hardware to implement the DDA proposed in chapter 4 was not available, all the algorithms presented in chapter 4 were implemented in a sequential approach using C++.

This chapter will present the architecture of this implementation, the various classes, attributes, methods, and interconnections.

In a hardware implementation, the overflow and the size of the registers are implemented in the hardware. For the sequential approach, they are implemented as comparisons and subtractions to generate the same effect. As a consequence of this implementation the performance is drastically reduced. Each step in a parallel implementation is equivalent to hundreds of comparisons, additions or subtractions in a sequential implementation. Also there are the overheads to call each integrator method.

With all these commands the one step in the parallel implementation is translated

into thousands of instructions in a sequential form. Since the frequency needed is approximately 580 kHz, the software approach (sequential organization) could not be implemented in the HRU hardware.

As a consequence, all results were found through the simulation software using the same data files that Carole Bolduc used in [Bolduc 95].

The software has been developed under Linux kernel version 2.2.14, using the GNU C++ compiler gcc 2.95.2, Python version 1.5.2 and Tk version 8.0. The Linux OS has been chosen because of its stability and the tools available through the internet at very low cost.

All C++ code is C++ ANSI and can be ported to other operational systems with no additional effort using a C++ ANSI compliant compiler.

The graphical user interface (GUI) was developed using Python, an object oriented script language, which made it possible to develop the stripcharts more easily than under C++. It is a very good script language for fast prototyping, for laboratory interface or GUI development. It is also portable to other operational systems, such as Windows 9x™.

To develop the stripchart program, the libraries BLT version 2.4 and PMW version 0.8.3 were used under Python. BLT is a graphic library to Tk and PMW (Python Megawidgets) consists of a set of base classes that makes the Tk programming under Python easier.

5.1 Software Implementation

This section presents the description of the classes, their attributes, methods and how they relate to each other.

Since the number of integrators needed to implement the navigation algorithm is too big, the size of the software made it impractical to be put as an appendix to this work, since it has almost 10,000 lines adding all files together.

The source files are:

```
integr.cpp
dda_alg.cpp
comp.cpp
nav1.cpp
nav2.cpp
navigat.cpp
ortho.cpp
simul1.cpp
interpol.cpp
tspline.cpp
navsimul.cpp
mainnav.cpp
datagen.cpp
stripchart.py
```

Each file has the corresponding header file with the description, attributes, methods and constructors for each class.

All algorithms have been implemented as object oriented classes that work as collectors of integrators and inherit the common behavior from the base class `TDDA_Alg`. The navigation and compensation algorithm classes are implemented in the files `comp.cpp`, `nav1.cpp`, `nav2.cpp` and `navigat.cpp`. The orthogonalization algorithms are implemented in the file `ortho.cpp`.

The class `TSimul1` which is responsible for reading the configuration files, the raw data and interface the navigation class `TNavigation` is implemented in the file `simul1.cpp`.

Since the data files had been sampled with a 1 Hz sampling frequency, it was necessary to interpolate the data points in order to generate the sampling frequency of 580,922 Hz. To interpolate the data points the `Tinterpol` class was used, which implements a cubic B-Spline interpolation routine based in the algorithm proposed in [William 98]. It is implemented in the file `tspline.cpp`.

`stripchart.py` is the python script that generates the stripchart of the navigation program as the simulation is run, enabling the user to see the behavior of the navigation variables quaternions, DCM D and Q , true angular velocity μ_c , damped angular rates ρ , true acceleration A_c and true velocity V_c .

This script file has two configuration files `stripgraph.conf` and `stripchart.conf`. The `stripgraph.conf` defines the size of the graphs and font to be used, enabling `stripchart.py` to run in different resolutions. The `stripchart.conf` has information about the variables being plotted, such as the titles, the range of the x-axis to be displayed and the amount of it to be shifted.

Appendix B shows the description of the files that are part of the simulation software `navsimul`.

5.2 Configuration Files

This section presents all the configuration files used by the simulation programs. The content of each one is presented in appendix C.

`hru.cfg` this file has all the parameters for the navigation and is loaded by the `TSimul` instance.

`inout.cfg` this file has the directories and names of the input and output files.

`datagen.cfg` this file defines the parameters for the `datagen` program.

`stripchart.conf` this file has all the parameters of the python script `stripchart.py`, such as the title of the graphs, number of iterations per point, etc.

`stripgraph.conf` this file has the information needed by `stripchart.py` to create the output windows at different resolutions. There are two versions of this file one for a 640x480 and an other for a 1200x768 graphical resolutions.

`optionDB` this file has the font to be used by `stripchart.py`.

Chapter 6

Results

The DDA approach is intended to be implemented in hardware allowing the integrations to be done in parallel through all the stages of the algorithm. Since the hardware implementation was not possible, the sequential implementation using C++ has been done, as presented in chapter 5.

As a result of this approach, the software has 304 integrators implemented in 11 classes in the simulation software.

In reference [Bolduc 95] there are several simulation data files, and several test runs developed with the same algorithm that is implemented in this work, but the sampling frequency was 1Hz and the frequency needed for a resolution of 21bits in the DDA implementation is 580,923Hz, as presented in table 6.1, the parameters file of `navsimul`.

So it was necessary to interpolate 580,923 points for each point in the data files, which was done using the classes `TSpline`, `TIntAcc` and `TIntGyr` presented in chapter 5.

The use of the B-Spline interpolation plus the number of integrators have generated a great overhead for each iteration, and because of this the simulation time has become very long. To simulate a file with 510 points it takes approximately 5 hours with an AMD K6-II at 300MHz with 64Mbytes of RAM, and 510 points means 8.5 minutes with a sampling frequency of 1Hz.

According to reference [Bolduc 95] the time needed to damp half of the drift in pitch, for example, would be about 10,800 seconds, which corresponds to approximately 106 hours, or 4.4 days, of simulation with this sequential implementation.

So because of the really long simulation time, the results presented in this chapter do not have all sets of simulations as presented in reference [Bolduc 95]. The two that were considered relevant to verify the results with this implementation are shown bellow.

6.1 Simulation Results

This section will present the results of two simulations. The first one is the simple simulation of an ideal stationary vehicle, to verify that the output is stationary as it should be.

The second simulation is with a real stationary input from a sample file from the HRU used in [Bolduc 95] for the coarse alignment test and for the analysis of the damping parameters τ and R . The use of the real sampled data with its inherent noise and bias drift is important to verify the stability of the system.

6.1.1 Ideal Sample Data

The ideal sample data file was generated with the `datagen` program. This program generates a file with the same format of HRU's, with the corresponding bias, quadrature and mass unbalance terms of the sensors, applied for a chosen position and attitude.

The sample data was generated for:

- Latitude 48°
- Longitude -123°
- wander 0°
- roll 0°
- pitch 0°
- yaw 0°

The configuration file for the `navsimul` program is presented in table 6.1.

Since for 200 points the generated interpolated data would have 116,184,600 points, which corresponds to a file with approximately 3 Gbytes, for a double precision information in a binary form, the output of the `navsimul` was generated in the same frequency of the input file, at 1Hz. At every new data point the most recent status of the integrators was transformed into position and attitude information and was written in the output file. During all simulations the mean values of attitude and position were also calculated and written to the mean output file at the end of the simulation.

The mean values of position and attitude from the `navsimul` program are:

Wander	=0.000000
Latitude	=47.999851

Longitude	=57.000075
Yaw	=0.000000
Pitch	=0.000000
Roll	=0.000000

To follow the simulation software behavior the `stripchart.py` script was used. It presents the status of the simulation software parameters corrected by its integrator scale factors in stripcharts. Figure 6.1 presents the plots generated by this program during the simulation. It does not present the stripcharts of all the simulations because of the reasons presented above.

As can be seen in figure 6.1 the expected constant output was generated.

6.1.2 Static Vehicle Sample Data

This simulation used the files `hru_acc3.out` and `hru_gyr3.out`, the same files used in [Bolduc 95] for the coarse alignment test. These files were generated at the DREO facilities with the HRU local-leveled and pointing north with the sensors oriented as follows:

Gyro S/N 19:	X-Case: East	Y-Case: Down
Gyro S/N 24:	X-Case: North	Y-Case: Down
Accelerometers:	AX (S/N 232): East	
	AY (S/N 228): South	
	AZ (S/N 234): Up	

The configuration file and position settings are the same used in the ideal conditions presented in section 6.1.1 and table 6.1.

Parameter	Values
AB (g)	0.00292658083 0.0004389546 0.009026187
AK	1 1 1
A	1 0 0 0 1 0 0 0 1
GB (deg/h)	-7.9482 -2.6698 6.7845 0.84124
GK	1 1 1 1
MU (deg/h/g)	1.191 -3.0981 1.233 -0.5190
Qd (deg/h/g)	7.743 -0.6642 7.680 -2.9654
G	1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0
ω_{0r}	0 0 0 0
a_{0r}	0 0 0
V_{0c}	0 0 0
τ	1080
R (m)	3,000,000 m
R_o (m)	6,378,140 m
act	0.43
Ω_{ie} (deg/h)	15.0417
g (m/s ²)	9.806
freq. (Hz)	580,923

Table 6.1: Configuration file used by `navsimul` program to simulate the ideal input file generated by `datagen` program in order to verify the static vehicle ideal input.

Figure 6.1: Stripcharts of the simulation using the data file generated by `datagen` program as the ideal input file to program `navsimul`, in order to verify the static vehicle ideal input.

The mean output values of position and attitude found through the navigation program from [Bolduc 95] are:

Wander	=0.000000
Latitude	=45.35004
Longitude	=-122.99785
Yaw	=0.00438
Pitch	=0.09175
Roll	=0.10920

Following the same approach presented for the ideal data, the stripcharts in figure 6.2 were generated. In this set of stripcharts it can be seen that the bias in the accelerometers are not exactly equal to the value in `hru.cfg` which generated the velocity component. The same can be verified with respect to the angular rates and the quaternions values.

The mean values are as expected even with the bias drifts and noise in the sampled data file. The results from [Bolduc 95] work are:

Wander	=0.00000
Latitude	=45.34824
Longitude	=-122.99990
Yaw	=0.00211
Pitch	=0.00264
Roll	=0.00209

Figure 6.2: Stripcharts of the simulation using the data file sampled at 1Hz sampling frequency from HRU in DREO facilities with it local-leveled and pointing north.

Chapter 7

Conclusion

The results presented in chapter 6 verified that the system is stable for a static input. It is assumed from the results presented in [Bolduc 95] that it will work for normal conditions, since the parameters used are the same.

As presented in chapter 6, the simulation time is too long and no other simulations have been made, such as the damping of attitude errors or tracking position simulations.

The parameters of the integrators are strongly connected to each other, as presented in section 4.5. The change in the value of one parameter implies changes in the others in order to keep the compatibility between the stages of integration.

The values presented in table 6.1 of chapter 6 were implemented with a spreadsheet using StarOffice™ version 5.1. As the changes in one parameter implies changes in others, and with the necessity that N_i be a power of 2, this analysis was lengthy.

The precision of the system is related to the sampling frequency and the maximum values of angular velocity, acceleration and velocity expected. For different applications, different values should be found.

The maximum parameters for the sensors, such as quadrature, mass unbalance and scale factors, also need to be in a range defined by the supremum values of the integrators. If new sensors have their parameters out of the range a new parameter analysis should be done.

Special care should be taken to define the initial values of the integrators and the correct units of the values, since these values are propagated to the other integrators, therefore affecting the behavior of the entire system.

The use of the `stripchart.py` program is a powerful tool to debug and diagnose the system performance, as the saturation of registers and the propagation through the integrators can be seen in real simulation time.

7.1 Future Research

As future research:

1. The error analysis of the DDA implementation of the navigation and compensation algorithms. This analysis has not been developed because of the complexity as a consequence of the large number of integrators, 304 integrators. In section 3.6 the error analysis for a single integrator was presented.
2. The implementation in hardware of the DDA presented in this work using devices such as FPGA's, which can enable the performance analysis that could not be developed using the simulation software.
3. Implementation of an optimization process to define the values of the 34 parameters of the DDA implementation, since these values are dependent on each other.

The use of genetic algorithm seems to be a possibility for the implementation of the optimization process.

4. Design of a less complex problem using the DDA approach, such as the rebalance loop of the GyroWheelTM, which would enable an implementation with fewer integrators, possibly enabling the sequential implementation and consequently a better performance analysis, simulation and test.

Bibliography

- [Bolduc 95] Bolduc, Carole R. M. - "Self-Alignment and Navigation Algorithms for DREO Navigation Laboratory Heading Reference Unit", Master Thesis, Department of Mechanic and Aerospace Engineer, Carleton University, Ottawa, ON, 1995.
- [Britting 71] Britting, Kenneth R. - "Inertial Navigation System Analysis", Wiley & Sons Inc., New York USA, 1971, ISBN 0-471-10485-X, [TL588.5 B75].
- [Chai 94] Chai, O. H.; Wong, Y. S.; Poo, A. N. - "A DDA Parabolic Interpolators for Computer Numerical Control of Machine Tools", Mechatronics Vol. 4, No. 7, pp. 673-692, 1994, Great Britain.
- [Farrell 76] Farrell, James L. - "Integrated Aircraft Navigation", Academic Press Inc., New York , NY, 1973, ISBN 0-12-249750-3, [TL588.5.F34].
- [Fejes 94] Fejes, T. G. - "An Ultra-Precision Digitizer for Inertial Systems" - Master Thesis, Department of Mechanic and Aerospace Engineer, Carleton University, Ottawa, ON, 1996.
- [Friedman 71] Friedman, Avner - "Advanced Calculus" - Holt, Rinehart and Winston Inc., USA, 1971, ISBN 03-083983-1, [QA.303.F831].

- [Grayson 00] Grayson, J.E. - "Python and Tkinter Programming", Manning Publications Co., USA, 2000, ISBN 1-884777-81-3.
- [Hartal 97] Hartl, Derek - "Review of the Status of The TINORU 7716 Digitiser Boards for Use in The HRU ", Carleton University, Ottawa ON, February 1997.
- [Hartal-1 97] Hartl, Derek - "Heading Reference Unit (HRU) Status Review", Carleton University, Ottawa ON, February 1997.
- [Hoeshele 94] Hoeschele, D. F. - "Analog-to-Digital and Digital-to-Analog Conversion Techniques", 2nd edition, John Wiley & Sons, Inc. 1994, New York, NY, ISBN 0-471-57147-4, [TK7887.6.H6.C2].
- [Lawrence 93] Lawrence, A. - "Moderne Inertial Technology: Navigation, Guidance, and Control", Spring-Verlag, New York, NY, 1993, ISBN 0-387-97868-2, [TL588.5.L38].
- [Lutz 99] Lutz, Mark and Ascher, David - "Learning Python", O'Reilly & Associates Inc., USA, 1999, ISBN 1-56592-464-9.
- [Norsworthy 97] Norsworthy, S. R. Schreier, R. Temes, G. C. - "Delta-Sigma Data Converters", IEEE Press, 1997, New York, NY.
- [Park 94] Park, S. W.; Oh, J. H. - "Hardware Realization of Inverse Kinematics for Robot Manipulators", IEEE Transactions on Industrial Electronics Vol. 41, No. 1, pp. 45-50, February 1994.

- [Proakis 96] Proakis, J. G.; Manolakis, D. G. - "Digital Signal Processing: Principles, Algorithms and Applications", Prentice-Hall, New Jersey 1996, ISBN 0-13-373762-4.
- [Siouris 93] Siouris, George M. - "Aerospace Avionics Systems: a modern synthesis" - Academic Press Inc., San Diego CA. USA. 1993. ISBN 0-12-646890-7. [TL588.5 S56].
- [Sizer 68] Sizer, T. R. H. - "The Digital Differential Analyzer", Chapman and Hall Ltd., London, 1968.
- [Staley 84] Staley, D. A. - "ANCON-R.829 Analysis of a Stable Platform Vertical Control Loop", ANCON Space Technology Corporation, Ontario Canada, September 1984.
- [Staley 98] Staley, D. A. - "Orbital Mechanics and Spacecraft Control", Lecture notes Carleton University winter 98/99 course 88.515.
- [William 98] William, H. - "Numerical Recipes in C: The Art of Scientific Computing", 2nd ed., Cambridge, England, 1998, [QA76.73.C15N865 1992].

Appendix A

Analog to Digital Converters

Analog to Digital (A/D) converters and Digital to Analog (D/A) converters provide the interface between the analog signal domain and the binary digital computational domain, making it possible to interface and control sensors and equipment via software, with the advantage of easy upgrade and adjustment. However, for this to be true fast and accurate digitizers and controllers, compatible with the time constant of the controlled system are needed.

Inertial sensors are a mature technology where sensors can reach resolution of 0.001 parts per million (ppm), which is beyond the resolution and dynamic range of most digital converters. From [Fejes 94] the dynamic range necessary for the strap-down HRU is $0.01^\circ/\text{hour}$ to $400^\circ/\text{hour}$ ($1\mu V$ to $10V$), which will give us a 20-bit resolution with an output rate of 1Hz up to 200Hz.

This appendix presents some A/D converters, some of their advantages and disadvantages. The concepts will follow the approach in reference [Hoeshele 94], but it is not the scope of this appendix to make a complete analysis of the A/D converter technolo-

gies, instead the principal technologies available will be presented in order to justify the choice made to apply Digital Differential Analyzers to implement the navigation and compensation algorithms of the HRU.

Basic Concepts

The design of an A/D converter begins with an analysis of the system requirements, where the first two requirements that must be determined are:

- dynamic range;
- speed of conversion.

An important characteristic of analog signals is the noise in the analog value that is either inherent in the transducer, or in the circuit used to amplify the analog signal. The value of the noise at small signal levels or the uncertainty in the analog measurement compared to the full-scale signal value at the upper limit of its range, determines the *dynamic range* of the usable analog signal. In other words, we can define *dynamic range* as the ratio of the largest input that can be converted to the smallest step of the converter. For example, a 10-bit A/D with an input of 0-1V has a quantization step of $\frac{1V - 0V}{2^{10}} = 0.97\text{mV}$, therefore the *dynamic range* ratio is $\frac{1.0V}{0.97\text{mV}} = 1024$, in decibels (dB) $20 \log 1024 = 60\text{dB}$. Table A shows values for different *dynamic ranges* and quantizing accuracy [Hoeshele 94] in $\pm \frac{1}{2}$ bit (%).

The *speed of conversion* is a measure of conversions per second and is associated with the time constant of the system.

The other requirements that will aid the selection of the A/D converter are [Hoeshele 94]:

Number of Binary Bits	Decimal Equivalent Value	Quantizing Accuracy (%)
4	15	± 3.3
6	63	± 1.6
8	255	± 0.4
10	1023	± 0.1
12	4095	± 0.025
14	16,383	± 0.006
16	65,535	± 0.0015
18	262,143	± 0.00038
20	1,048,575	± 0.000095

A/C converter range and resolution in $\pm \frac{1}{2}$ bit (%).

- accuracy;
- operating power characteristics;
- cost;
- size;
- weight;
- reliability/complexity/maintainability;
- operating environment.

Classification

Nyquist Rate versus Oversampled

According to [Fejes 94] we can classify A/D converters in function of the sampling frequency as:

- *Nyquist Rate*;
- *Oversampled*.

The first operates near the Nyquist rate, which is twice the maximum frequency of interest, however because of practical considerations, that is limitation in low pass filter, hardware cutoff characteristics and limitation in digital signal processing, higher sampling rates are often used (4 to 6 times the Nyquist rate). The converters of this type are:

- linear pulse code modulator (PCM);
- differential PCM;
- algorithm converters;
- flash converters;
- pipe-line converters;
- integrating converters.

The *oversampled* converter uses a sampling frequency much higher than the Nyquist rate, typically more than 16 times higher, which allows the elimination of high performance anti-alias filters in design, the noise is spread over the spectrum such that baseband noise is reduced. The converters of this type are:

- Delta Sigma;
- Sigma.

Direct-Comparison versus Integrating

[Hoeshele 94] presents another classifications of A/D where they are divided between *direct-comparison* converters and *integrating* converters. In the *direct-comparison* converters each reference to be used in the comparison is assigned a digital value. A simple example is the 3-bit flash A/D converter.

Other types of direct comparison A/D converters take a longer time to complete a conversion because they determine the equivalent digital value by generating a sequence of reference analog signals by sequentially comparing the analog input against fixed analog reference. Examples are the PCM successive approximation A/D converters and the pipeline A/D converter.

The integrating A/D converters have a digital counter that counts the time it takes a reference signal to integrate a charge on a capacitor to the analog input signal value, as examples we have:

- single-slope A/D converter;
- dual-slope A/D converter;
- voltage-to-frequency (V/F) converter.

Following are some brief descriptions of these A/D converters.

Pulse Code Modulation (PCM)

The most common PCM A/D converter is the successive approximation A/D converter. This converter uses a single comparator, a successive approximation register and a digital to analog (D/A) converter in feed back. This conversion starts with the most

significant bit (MSB) and successively tries a ONE in each bit of the D/A converter. As each bit is tried, the output of the D/A converter is compared against the input analog signal. If the D/A output is larger the ONE is removed and a ONE is tried in the next most significant bit. If the analog input signal is greater than the D/A output the ONE remains.

In each step of the process, sufficient time must be allowed for the D/A converter to reach its final value with its accuracy constraints, then sufficient time must be allowed for the comparator to recover from its previous input and to generate an output dependent upon its new input from the D/A converter.

This A/D converter requires n steps for encoding to an n -bit binary value. The digital output can be taken serially, as each bit is tried, or it can be taken in parallel at the end of the conversion time by reading the flip-flops driving the D/A converter.

Differential PCM

In the Differential PCM the quantizer converts the difference between the current and the previous input sample. It is useful for inputs with a large dynamic range, especially music.

Algorithmic Converters

In an algorithmic successive approximation converter the converter calculates the error signal using a single comparator and cycle it through a loop, the analog input is amplified and acquired by the S/H (sample and hold), then the MSB is determined by comparing the S/H output against V_R . If the signal is greater than V_R it is subtracted from it in the feedback loop. If it is less than V_R then it is passed directly around the

feedback loop. The recirculation process continues for n cycles.

This approach for successive approximation converters eliminates the need for a precision D/A converter, but does require a precision $2X$ amplifier and a similar complexity of control and readout logic circuit.

Pipeline converters

A pipeline converter could be created by cascading a reduced accuracy A/D converter with a D/A converter and finding the error signal from the original analog signal. The error signal is converted and the result of the next conversion will have a greater resolution, in other words, the first stage converts a number of the most significant bits during the first sampled data time and then passes the residue to the input of the next stage in the pipeline for processing. During the second analog-signal sampling period, the first stage converts the MSB's for the next analog sample while the second stage converts the previous sample's residue to the next MSB's. The residue from this second approximation is passed along to the next stage, and so on until all bit values have been determined.

At the beginning of each analog-signal sampling period, the new sample of the analog signal is entered into the first stage of the converter for its first approximation to digital form. The result is that each sample of analog signal taken follows the previous sample through the string of partial converters resulting in a converter rate equal to the sampling rate, but the full digital equivalent value is delayed in the pipeline by the number of sample times it takes to pass through the successive stages of the converter.

This type of converter can be used where the latency of conversion (the delay in having the full digital equivalent value of the analog signal) is of little importance for

the application, and in most cases the analog sampling rate is the important criterion.

Flash Converters

These converters use one analog comparator, with a fixed reference voltage, for every quantization level in the digital number from zero to full scale. The conversion rate possible with this type of converter is extremely fast because the conversion is completed in one step. However, since the number of comparators increases geometrically with resolution, the amount of required circuitry is practically doubled.

The errors in this conversion process are the comparator and the reference voltage errors. The basic problem is typically not the static accuracy, but obtaining fast enough transient response so that the true equivalent digital word is available within the allowed time. Also when many comparators detect at one time, significant transients can be kicked back on the analog signal input.

Integrating A/D Converters

This type of A/D converter is applied where the analog input signal is:

- DC or slowly varying;
- with necessity of a high dynamic range;
- with necessity of high noise immunity.

They are simple and relatively low cost.

Single-Slope or Ramp-Integrating A/D converter

At the beginning of the integration of the input signal, a switch is opened and the output of the integrator is a constant slope voltage ramp. As the integrator output voltage ramps up from zero volts, the accumulator counts the cycles of the reference clock frequency. When the voltage from the integrator equals the input analog voltage, the comparator output changes state, the comparator inhibits the reference frequency from entering the counter accumulator, and the final count in the accumulator is the digital equivalent of the analog input voltage.

The advantages of this type of A/D converter are simplicity, high dynamic range capability, monotonic conversion with no missing codes and high frequency noise rejection due to the relatively long integration of the input signal during the conversion.

The error sources are, errors in values of the resistor and capacitor in the integrator, the reference voltage drift, clock frequency error, analog reset switching errors as well as comparator detection point error.

These errors make the use of this converter limited to relatively low accuracy applications.

Dual-Slope Integrating A/D Converter

To eliminate the errors inherent in the single-slope A/D converters the dual-slope A/D converter is used. The basic idea is to generate a pulse width proportional to the analog input voltage by making a time comparison between two integrators. In this way, many of the absolute errors in generating the integrated ramp voltage are canceled. The first integration is on the input analog signal. It proceeds for a fixed interval of time, t_1 . The input to the integrating circuit is then switched to a known reference voltage. The

time from this switching until the integrator output reaches some fixed reference point gives a measure of the analog input voltage.

The conversion begins with the reset and a convert pulse; the counters are reset to zero. Switch S_1 is closed and S_2 is opened so that the time integrator begins. Since the integrator output exceeds *zero volts* the comparator output changes state allowing the counter to receive clock pulses and count until an all ONE condition occurs. At the next clock signal the counter changes to all ZERO causing the input to switch between the reference voltage. Then the integrator output voltage decreases until it crosses zero, when the clock signal is inhibited. Reference [Hoeshele 94] proves mathematically that this conversion process eliminates some of the worst error sources in an integrating-type converter.

Voltage-to-Frequency (V/F) Converter

These A/D converters generate digital output pulses at a repetition rate that is linearly proportional to the analog input voltage. The pulses are counted for a fixed period of time, the *conversion time period*. The digital value is the average value since the pulse rate will increase or decrease as the analog signal V_{IA} increases or decreases during each conversion time period.

For a perfect V/F converter, the theoretical frequency of the output pulse trains f_0 is:

$$f_0 = \frac{V_{IA}}{V_{FS}} \cdot f_{FS}$$

where:

- V_{IA} is the analog input voltage;
- V_{FS} is the maximum full-scale input voltage; and
- f_{FS} is the maximum frequency for full-scale input voltage.

Charge Reset V/F Converter

Figure A.1: Charge reset V/F converter [Hoeshele 94].

The operation consists of the capacitor C_1 charging until the voltage on A is equal to V_{REF} the reference voltage. The comparator triggers the one-shot multivibrator which

closes the switch S_1 discharging the capacitor C_1 . This sequence sets the period for one cycle of voltage-to-frequency conversion.

The sources of error are the resistor R_1 , the capacitor C_1 , the reference voltage V_{REF} and the comparator detection error.

Charge Balancing V/F Converter

This type of V/F converter provides a more accurate V/F converter because it does not depend on the capacitor C_1 value.

The integrator is an operational amplifier with its negative input held at virtual ground by the negative feedback from C_1 . The input current I_{IA} is then directly proportional to V_{IA} , where $I_{IA} = \frac{V_{IA}}{R_1}$. As the integrating capacitor C_1 discharge, the integrator output voltage on A ramps linearly in a negative direction. When it reaches the zero-volt reference on the comparator, the comparator detects and triggers the one-shot multivibrator. During the one-shot's on time, t_o , the reset current, I_R , pulls charge from capacitor C_1 as the output voltage on A linearly ramps in the positive voltage direction. At the end of the one-shot on time the one-shot's turns off, thereby opening S_1 , the end of the rest period. The value of I_R is larger than I_{IA} since during reset it must be equal to I_{IA} plus the current necessary to reset the capacitor for the next discharge integration.

This circuit, as the charge reset V/F converter, is a free-running circuit, which means that once it is powered the circuit will start itself and generate an output frequency proportional to the input voltage.

Summarizing, it is an integration of a constant current source, with an analog comparator to detect when the ramp has reached a reference value.

Figure A.2: Charge balancing V/F converter [Hoeshele 94].

The value of f_0 using the circuit elements shown in figure A.2 is [Hoeshele 94]:

$$f_0 = V_{IA} \left(\frac{1}{R_1 t_R I_R} \right)$$

From this expression we have the scale factor for converting V_{IA} to f_0 and shows us the principal sources of errors.

Effect of C_1 Although the value of C_1 is not the determinate of f_0 , C_1 must be large enough to absorb the charge during the reset time without saturating the integrator amplifier and not too large so that a reasonable $\frac{\Delta V}{\Delta t}$ slope occurs at the comparator input for timely comparator detection. It is assumed that the value of C_1 is constant for the value of f_0 shown above. If dielectric absorption is significant the charge absorbed due to dielectric absorption is released slowly from C_1 and may not be discharged completely during the discharge time. This causes an effective variation in C_1 and results in a linearity error, since V_{IA} values would have a faster than predicted discharge time and therefore a higher f_0 than predicted for a linear conversion. Polystyrene or NPO ceramic capacitors are preferred for temperature coefficient and low dielectric absorption; polycarbonate and mica may be acceptable [Hoeshele 94].

Oversampled Converters

Delta Modulation

The principle of the delta modulator is to quantize the change in the signal from sample to sample rather than quantize the absolute value of the signal at each sample. The integrator in the feedback acts as a predictor, trying to minimize the error at each

sample. the error term is used to make the next prediction.

Delta-Sigma ($\Delta\Sigma$) Modulation

$\Delta\Sigma$ oversampled A/D converters have in recent years become more prevalent for high-accuracy, 12-bit to beyond 20-bit, A/D conversion of DC through moderately high (hundreds of kilohertz) AC signals. Their greatest advantage is that they trade greatly reduced analog circuit accuracy requirements for increased digital circuit complexity [Hoeshele 94]. The result is high precision at low cost.

The basic oversampled delta-sigma A/D converter is an integrating type A/D converter. The single-bit feedback D/A converter output is subtracted from the analog input signal, V_{IA} , in the summing amplifier. The resultant error signal from the summing amplifier output is low-pass filtered by the integrator and the integrated error signal polarity is detected by the single comparator. This comparator is a one-bit A/D converter. The output of the comparator drives the 1 bit D/A to a 1 or 0; a 1 if during the previous sample time the integrator output was detected by the comparator as being below $0V$; a 0 if the difference detected during the previous sample time was above the $0V$ reference of the comparator. The 1-bit D/A converter, as in successive approximation A/D converter, provides the negative feedback. Then for small values of V_{IA} the integrator would take many clock pulses to cross $0V$, after a single 1 was generated, during which time the comparator is sending 0's to the digital filter. If V_{IA} were at full scale, the integrator would cross zero every clock time and the comparator output would be a string of alternate 1's and 0's. The digital filter's function is to determine a digital number at its output that is proportional to the number of 1's in the previous bit stream from the comparator. Various types of digital filters are used to

perform this computational function, which is the most complex function in this type of A/D converter.

Appendix B

Description of the Simulation Software Files

`integr.cpp`

The DDA integrator class `TIntegrator` and two collector classes the `TMultiplyDDA`, which implements the DDA multiplier, and `TSquareDDA`, the square DDA implementation are implemented in `integr.cpp`, as presented in section 3.5.

```
class TIntegrator
{
protected:
    long Y;           // register Y
    long R;           // register R
    long limit;       // maximum value that Y and R can assume
    int N;            // size of Y and R in bits
    char flgY;        // flag for no-linear answer of
                     // integrator if 1
```

```

        char flag;          // flag of overflow - 1 when it occurs
public:
        char dz;            // output of the integrator
        TIntegrator(long Y0, long R0,
                     int N0, char flgY0);
        ~TIntegrator();
        void setY(long Y0);
        void setR(long R0);
        long getY();
        long getR();
        char getflag();
        char integrate(char dxi, char dyi);
}

```

```

class TSquareDDA
{
private:
        TIntegrator *integr;
        int N;              // size in bits of the registers
        char flag;          // overflow flag
public:
        TSquareDDA(long Y0,int N);
        TSquareDDA(long Y0,int N,char flagY0);
        ~TSquareDDA();
        long getY();
        char getflag();
        char square(char dx);
}

```

```

class TMultiplyDDA
{
private:

```

```

    TIntegrator *integr1;
    TIntegrator *integr2;
    int N;          // size in bits of the registers
    long *Y;        // Y registers
    char flag;      // overflow flag
public:
    TMultiplyDDA(long Y1,long Y2,int N);
    TMultiplyDDA(long Y1,long Y2,int N,char flagY0);
    ~TMultiplyDDA();
    long *getY();
    char getflag();
    char multiply(char du,char dv);
}

```

dda_alg.cpp

TDDA_Algo is the base class for all the algorithm classes, it has the basic attributes common to all the navigation classes and is implemented in this file.

```

class TDDA_Algo
{
protected:
    TIntegrator **integr;    // integrators
    TMultiplyDDA **multpl;   // multiply integrators
    TSquareDDA **square;     // square integrators
    int N;                  // size of the integrators registers
    int Nint;               // number of integrators
    int Nmlt;               // number of multiply integrators
    int Nsqr;               // number of square integrators
    long *Yi0;              // initial value of the integrators

```



```

    long *Ym0;          // initial value of the multipliers
    long *Ys0;          // initial value of the square
    char flag;          // flag the error if different of zero

    void Initialize();
    void Initialize(char flagY0);
public:
    TDDA_Alg(int Nint0,
              int Nmlt0,int Nsqr0,int N0);
    ~TDDA_Alg();

    char getflag();
    long *getintY();
    long *getmltY();
    long *getsqrY();
    int Constraint(); // verify the constraints of each
}

```

comp.cpp

This is the file that implements the compensators for the angular rates and accelerations sensed through the gyroscopes and accelerometers.

TAccelComp

This class implements the DDA compensator for the accelerometers data as presented in section 4.2.1 figure 4.2.

```

class TAccelComp:public TDDA_Alg
{

```

```

    double armax; // maximum acceleration raw value
    double Mk;    // supremum of AK
    double Kar;   // accel integrator scale factor
    double Ky;    // Ky for the integrators
    char *adt;    // output of the integrators
    char *da;     // output of the integrators
public:
    TAccelComp(double *AB0, double *AK0,
               double Mk0, double A0[3][3],
               double armax0, double *a0,
               int N);
    ~TAccelComp();

    long *getA();
    char *getadt();
    char *getda();
    double getMk();
    int Constraint(); // verify the constraint
    char *integrate(char *dar, char dt);
}

```

TW_Comp

This class implements the compensation algorithm for the gyros data. It follows the algorithm and diagram of figure 4.3 in section 4.2.2.

```

class TW_Comp:public TDDA_Alg
{
    double wrmax; // maximum angular velocity raw value
    double Mkw;   // supremum of Kwij terms

```

```

    double Mka;    // maximum value of Kaij terms
    double Kwr;    // scale factor for integrator
    double Kyw;    // Ky for the integrators
    double Kya;    // Ky for the integrators
    double Kw;     // scale factor for integrator
    char *wdt;     // output of the integrators
    char *dw;      // output of the integrators
    char flgConstraint; // Constraint flag 1=Ok; 0=fail

public:
    TW_Comp(double GK0[4], double G0[4][4],
            double Kaij[9], double wrmax0,
            double w0[3], double Mkw0,
            double Mka0, int N);
    ~TW_Comp();

    char *getdw();
    long *getW();
    double getMkw();
    int Constraint(); // verify the constraint
    char *integrate(char *da,char *dwr, char dt);
}

```

nav1.cpp

This file implements the classes that integrate the DCM elements and the quaternions.

TQ, **TD** and **Tqi** have each an instance of the orthogonalization class.

TQii, TQij and TQ

The DCM **Q** is the direction cosine matrix that transforms a vector in the system frame of reference to the computational frame of reference as a function of the quaternions. It has two major classes, the one which implements the diagonal elements called **TQii**, as in figure 4.5 (a), and **TQij** which implements the non-diagonal elements, as presented in figures 4.5 (b), (c) and (d). **TQ** is the collector of **TQii** and **TQij**.

```
class TQii: public TDDA_Algo
{
    double q[4];    // initial value of the quaternions
    double Kq;      // quaternions scale factor
    int Nq;         // size of the DCM Q integrators
    char *dQii;     // output of the integrators

    void Initialize(char flagY0);
    // override the TDDA_Algo one
    // allow the different size of
    // integrators

public:
    TQii(double *q0, double M2,
          double Kq0,int Nq,int NQ0);
    ~TQii();

    char *getdQii();
    int Constraint(); // verify the constraints

    char *integrate(char *dq);
}
```

```

class TQij: public TDDA_Algo
{
    double q[4];    // initial value of the quaternions
    double Kq;      // quaternions scale factor
    int NQ;
    char *dQij;     // output of the integrators

    void Initialize(char flagY0);
    // override the TDDA_Algo one to
    // allow the different size of
    // integrators
public:
    TQij(double *q0, double M2,
          double Kq0, int Nq, int NQ0);
    ~TQij();

    char *getdQij();
    int Constraint(); // verify all the constraints together
    char *integrate(char *dq);
}

class TQ: public TDDA_Algo
{
    TQii *Qii;      // diagonal elements
    TQij *Qij;      // non-diagonal elements
    double KQ;      // Q terms scale factor
    int NQ;
    char **dQ;      // increments
    double **Q;      // Q elements
    TDCM_Ortho *Q_Ortho; // Q terms orthogonalization
public:
    TQ(double *q, double Q0[3][3],

```

```

        double M2, double Kq0,
        int Nq, int NQ);
~TQ();

char **getdQ();
double **getQ();
char getflag();
int Constraint(); // verify all the constraints together
char **integrate(char *dq,char dt);
}

```

TD

TD is the implementation of the integrators of the elements of **D**, the DCM that transforms coordinates from the earth-centered earth-fixed frame of reference to the computational frame of reference, as presented in figure 4.6 section 4.3.3.

```

class TD: public TDDA_Alg
{
    double rho[2]; // value of rho
    double **D;    // values of D elements
    TDCM_Ortho *D_Ortho; // Q terms orthogonalization
    double Kvc;    // True velocity scale factor
    double KD;     // D terms scale factor
    double MRo;    // supremum of Ro
    double MD;     // supremum of D elements
    double Ro;     // constant Ro earth radius
    char **dD;     // output of the multipliers
    char drho[2];  // output of the integrators
public:
    TD(double *rho0, double D0[3][3],

```

```

        double MRo0, double Ro0,
        double KVc0, int N);
~TD();

char getflag();
char **getdD();
double **getD();
int Constraint(); // verify all the constraints together
char **integrate(char *dVc,char dt);
}

```

Tqi

This class implements the quaternions differential equation as presented in section 4.3.6 figure 4.10.

```

class Tqi: public TDDA_Alg
{
    double mu[3]; // initial values of mu
    double q[4];  // initial values of q
    double Kmu;
    double Kq;
    double K2q;   // scale factor of the 1/2
                  // constant multiplier
    char *dz;     // output of integrators
    char *dq;     // quaternions increments
    TOrtho *q_Ortho; // quaternions orthogonalization
public:
    Tqi(double *mu0,double *q0,double *dqdt0,
        double Kmu0, double Kq0, double M2q0,          int N);
    ~Tqi();
}

```

```

    char getflag();
    char *getdq();
    int Constraint(); // verify all the constraints together
    char *integrate(char *dmu, char dt);
}

```

nav2.cpp

This file implements the classes that are responsible for the angular velocities, acceleration and velocity components.

TDamped_rates

This class implements the damped spatial rate presented in [Bolduc 95] as in section 4.3.1 figure 4.4.

```

class TDamped_rate:public TDDA_Alg
{
    double nu[2]; // initial values of nu
    double KVc;
    double Mrho;
    double Krho;
    double MTau; // supremum
    double MR;
    double Mact;
    double Tau;
    double R;
    double a; // constant of the z damping component
    char *dz; // integrators output
}

```



```

    char *drho;    // output of the integrators
public:
    TDamped_rate(
        double Tau0, double R0,
        double a0, double *nu0,
        double *rho0, double Krho0,
        double Mrho0, double KVc0,
        double MTau0, double MR0,
        double Mact0, int N);
    ~TDamped_rate();

    char *getdrho();
    double *getV();
    int Constraint();
    char *integrate(char *dV,char d2Vy,char dt);
}

```

TMuc

This class implements the true angular velocity relative to the computational frame of reference μ_c as presented in section 4.3.4 figure 4.7.

```

class TMuc:public TDDA_Alg
{
    double MOmega; // supremum of omega
    double Kw;
    double KQ;
    double Kmu;
    double Omega;
    char *dmu;    // output of the integrator
public:

```

```

    TMuc(double Q0[3][3], double *w0,
          double Kw0, double KQ0, double M0omega0,
          double Omega0, int N);
    ~TMuc();

    char *getdmu();
    int Constraint();
    char *integrate(char **dQ, char *dw,
                    char *drho, char **dD);
}

```

TAc

This class implements the true acceleration relative to the computational frame of reference as presented in section 4.3.5 figure 4.8.

```

class TAc:public TDDA_Alg
{
    double Q[3][3]; // initial value of the Q matrix elements
    double a[3];    // initial value of the acceleration components
    double Ka;
    double KQ;
    char *dAc;      // output of the integrator
public:
    TAc(double Q0[3][3], double *a0,
          double Ka0, double KQ0, int N);
    ~TAc();

    char *getdAc();
    int Constraint();
    char *integrate(char **dQ, char *da);
}

```

```

}

```

TVc

This class implements the true velocity relative to the computational frame of reference as presented in section 4.3.7 figure 4.11. The $\dot{d2Vc}$ is the derivative of $\dot{V}_c$, it is implemented since it is needed as input to calculate ρ_z .

```

class TVc:public TDDA_Alg
{
    double Krho;
    double KD;
    double KAc;
    double KVc;
    double Kgz;
    double B[3];
    double MB;
    double MVc;
    double Omega;
    double M2Omega; // supremum of Omega (earth rate)
    char *dVc;      // output of the integrator
    char *d2Vc;     // output for the damped rates
public:
    TVc(double D[3][3],double *rho,
        double *Vc0, double *Ac0,
        double KVc0, double Krho0, double KD0,
        double KAc0, double Kt,
        double M2Omega0, double Omega0,
        double *B0, double gz0, int N);
    ~TVc();

```

```

    char *getd2Vc();
    char *getdVc();
    int Constraint();
    char *integrate(char **dD, char *drho,
                    char *dVc, char *dAc, char dt);
}

```

navigat.cpp

This file contains the navigation class TNavigation which uses all the others classes defined in nav1.cpp, nav2.cpp and comp.cpp. It also defines the TNavInitdt, the configuration data structure used by TSimull to configure the navigation algorithm.

```

struct TNavInitdt {
    double AB[3];    // accelerometers bias
    double AK[3];    // accelerometers scale factors
    double A[3][3];  // A DCM matrix of misalignment
    double GB[4];    // angular rate bias
    double GK[4];    // gyros scale factors
    double MU[4];    // the mass unbalance parameters
    double Qd[4];    // the quadrature parameters
    double G[4][4];  // G DCM matrix of misalignment
    double Kaij[9];  // compensator parameter
    double a[3];     // initial value of the acceleration components
    double w[4];     // initial value of the angular rate components

    double Ac[3];    // initial value of the true acceleration components
    double Vc[3];    // initial value of the true velocity components
    double q[4];     // initial value of the quaternions
    double rho[3];   // initial value of rho
}

```

```

double mu[3];    // initial values of mu true angular rate
double nu[3];    // initial values of nu dumped velocity
double dqdt[4];
double Q[3][3]; // initial value of DCM Q
double D[3][3]; // initial value of DCM D

double Tau;      // damping filter constant
double act;      // damping constant of z axis
double Ro;      // constant R earth radius
double R;        // constant R of damping
double Omega;    // earth rate
double B[3];     // damping constants
double gm;       // gravity acceleration

int Nar;         // size of Accel_Comp registers in bits
int Nwr;         // size of W_Comp registers in bits
int Nrho;        // size of Damped_Rate in bits
int NQ;          // size of DCM Q terms in bits
int ND;          // size of DCM D terms in bits
int Nmu;         // size of True angular velocity in bits
int NAc;         // size of true acceleration in bits
int Nq;          // size of quaternions in bits
int NVc;         // size of true velocity in bits

double armax;    // maximum acceleration raw value
double Mk;       // supremum of AK (Accel_Comp)
double wrmax;    // maximum angular velocity raw value
double Mkw;      // supremum of Kw terms in W_Comp
double Mka;      // supremum of Ka terms in W_Comp
double Mrho;     // supremum of damped rate
double Mmu;      // supremum of true angular velocity
double MTau;     // supremum of Tau (damping parameter)

```

```

double Mact;                // supremum for act (damping param
double MR;                  // supremum of R (damping constant)
double MRo;                  // supremum of Ro (earth radius)
double MOmega;              // supremum of omega
double M2Omega;             // supremum of 2*omega
double MAc;                  // supremum of true acceleration
double MVc;                  // supremum of true velocity
double M2;                   // supremum of the 2 multiplier in DCM Q
double M2q;                  // supremum of the 1/2 multiplier in quaternions
double Kt;                   // time integrators scale factor
double Kar;                  // acceleration raw scale factor
double Kwr;                  // angular velocity raw scale factor
double Ka;                   // acceleration scale factor
double Kw;                   // angular velocity scale factor
double KQ;                   // DCM Q terms scale factor
double KD;                   // DCM D terms scale factor
double Krho;                 // damped rate scale factor
double Kmu;                  // true angular velocity scale factor
double KAc;                  // true acceleration scale factor
double Kq;                   // quaternions scale factor
double Kvc;                  // true velocity scale factor
}

```

```

class TNavigation

```

```

{
    TNavInitdt *NavInitdt;
    TAccelComp *AccelComp; // accelerometer compensator
    TW_Comp *W_Comp; // angular velocity compensator
    TQ *Q;              // integrates the DCM Q components
    TD *D;               // integrates the DCM D components
    Tqi *qi;             // integrates the quaternions
    TDamped_rate *Damped_rate; // integrates the damped rates rho
}

```

```

    TMuc *Muc;          // integrates the true angular velocity
    TAc *Ac;            // integrates the true acceleration
    TVc *Vc;            // integrates the true velocity
    char *adt;          // compensated acceleration
    char *da;           // compensated acceleration
    char *dw;           // compensated angular velocity
    char *drho;         // dumped angular rate
    char *dmu;          // true angular rate increment
    char *dAc;          // true acceleration increment
    char *dq;           // quaternions increments
    char *d2Vc;         // true velocity derivative increments
    char *dVc;          // true velocity increments
    char **dQ;          // DCM Q components increments
    char **dD;          // DCM D components increments
    double **DCM[2];    // pointer to the DCM Q and D
    double *attitude;   // attitude parameters
    double *position;   // position parameters
    long count;         // iterations count for python pipe

    // python pipe parameters
    long iterations;    // iterations limit for python pipe output
    int flgPr;          // print to stdin flag for python pipe
    long itcount;       // iterations counter

public:
    TNavigation(TNavInitdt *NavInit,
               long multi0, int flgPr,
               long iterations0);
    ~TNavigation();

    double ***getDCM();

```

```

    double *getAttitude();
    double *getPosition();
    int Constraint(); // verify all the constraints together
    char integrate(char *dar,char *dwr,char dt);
    void SetInitialCondition(TNavInitdt *NavInit);
}

```

ortho.cpp

This file is the implementation of the orthogonalization algorithms for the DCM terms and the quaternions, as presented in section 4.4.

The class TOrtho is the implementation of the quaternion's orthogonalization as in figure 4.12, and the class TDCM_Ortho is the implementation of the DCM orthogonalization algorithm as in figure 4.13.

```

class TOrtho:public TDDA_Alg
{
    int nterms;      // number of terms 4 for
                    // quaternions 3 for DCM
    double Mv;       // supremum of the terms
    double Me;       // supremum of epsilon
    double epsilon;
    char *dv_;       // integrators output
public:
    TOrtho(double *v0,int nterms0,int N);
    ~TOrtho();

    char *getd();
    char *integrate(char *dv0);
}

```



```

}

class TDCM_Ortho:public TDDA_Alg
{
    double MD;          // DCM D matrix supremum
    double Me;          // supremum of epsilon
    double epsilon;
    char **dD_;         // output of the integrator
    char *dMlt;         // output of the multipliers
public:
    TDCM_Ortho(double **D0,int N);
    ~TDCM_Ortho();

    char **getdD();
    char **integrate(char **dD);
}

```

simul1.cpp

This file is responsible to read the configuration file `hru.cfg`, fill the data structure `TNavInitdt` with all the parameters for the integrators, and to create the instance of the `TNavigation` class that will be responsible for running the navigation and compensation algorithms.

```

class TSimul1
{
    TNavInitdt *NavInit;
    char filecfg[MAXPATH]; // Config file name
    char fileinacc[MAXPATH]; // Input data file name

```

```

char fileingyr[MAXPATH]; // Input data file name
char fileout[MAXPATH];  // Output file name
char fileQout[MAXPATH]; // Output file name
char fileDout[MAXPATH]; // Output file name
FILE *fcfg;             // pointer to cfg input file
FILE *finacc;           // pointer to data input file
FILE *fingyr;           // pointer to data input file
FILE *fout;             // pointer to data output file
FILE *fQout;            // pointer to data output file
FILE *fDout;            // pointer to data output file
int *dar;               // raw increments
int *dwr;               // raw increments
double *accr;           // raw accelerometer data
double *wr;             // raw gyro data
double Mkread,Mkwread;  // values from the cfg file
double pos[3];          // latitude, longitude, wander
double att[3];          // roll,pitch,yaw
double Qmean[3][3];     // mean values of Q
double Dmean[3][3];     // mean value of D
long N;                 // counter of elements to mean calculation
char flag;
int flgPr;              // python flag 0->Python output

void Initialize(char *inacc, char *ingyr,
               char *Qout, char *Dout,
               char *out, int flgPr0,
               long iterations0);

void InputCfg();
void ReadCfg();
void WriteCfg();
void FillNavInit();

```

```

        int ReadData();
        void WriteData();
public:
    TNavigation *Navigation;// the navigation algorithm

    TSimul1(char *inacc,char *ingyr,
            char *Qout,char *Dout,char *out,
            int flgPr0, long iterations0);
    TSimul1(char *cfg,char *inacc,
            char *ingyr, char *Qout,
            char *Dout,char *out, int flgPr0,
            long iterations0);
    TSimul1(char *cfg, double *accr0,
            double *wr0, int flgPr0,
            long iterations0);
    ~TSimul1();

    TNavInitdt *getNavInit();
    void setInitials(double *w0,double *a0);
    void run();
}

```

interpol.cpp

The classes TIntAcc and TIntGyr uses TSpline to implement the cubic spline interpolation of the accelerometer and gyroscope data, respectively, from the input files. The interpolation algorithm implemented in TSpline is based on the algorithm presented in [William 98].

```

class TIntAcc
{
protected:
    FILE *fin;
    FILE *fout;
    int res;
    int mult;
    long N;           // number of points
    long n0,nf;
    long size;        // size of the buffer
    long newN;
    double *t,**acc;
    TSpline **spline;

    int ReadData();
public:
    TIntAcc(char *FileIn, char *FileOut,
            long NO, long Nf, int Mult);
    TIntAcc(char *FileIn, long NO, long Nf);
    ~TIntAcc();

    double Interpolate(int l, double x);
    void getAcc(long n, double *resacc);
    double getT(long n);
    int Run();
}

```

```

class TIntGyr
{
protected:
    FILE *fin;
    FILE *fout;

```

```

    int res;
    int mult;
    long N;          // number of points
    long n0,nf;
    long size;       // size of the buffer
    long newN;
    double *t,**gyr;
    TSpline **spline;

    int ReadData();
public:
    TIntGyr(char *FileIn, char *FileOut,
             long N0, long Nf, int Mult);
    TIntGyr(char *FileIn, long N0, long Nf);
    ~TIntGyr();

    double Interpolate(int l, double x);
    void getGyr(long n, double *resgyr);
    double getT(long n);
    int Run();
}

```

tspline.cpp

In this file is implemented the class TSpline which is responsible for the cubic B-spline interpolation used by TIntAcc and TIntGyr. The interpolation algorithm is based on the algorithm presented in [William 98].

```

class TSpline
{

```

```

protected:
    double *x;
    double *y;
    double *coef;      // coefficients
    long N;            // number of points
    double deriv0;     // boundary
    double derivN;     // conditions

    void Sort();
    void Spline();
public:
    TSpline(double *x, double *y, long N,
            double ypn, double yp1);
    TSpline(double *x, double *y, long N);
    ~TSpline();

    double splint(double y);
    double *getCoefs();
}

```

navsimul.cpp

This is one of the main files of the navigation/simulation implementation. It calls the spline routines through instances of TIntAcc and TIntGyr, finds the coefficients of the B-splines, and then, for each new point interpolated, finds the increments and runs the navigation algorithm through the instance of TSimul1.

```

class TNavSimul
{

```

```

// configuration information
FILE *finout;      // pointer to inout.cfg file
FILE *fhru;        // pointer to hru.cfg file
char inDir[MAXPATH]; // input directory
char outDir[MAXPATH]; // output directory
char inAcc[MAXPATH]; // input acceleration file
FILE *finAcc;      // pointer to input acceleration file
char inGyr[MAXPATH]; // input gyros data file
FILE *finGyr;      // pointer to input gyros data file
char outAtt[MAXPATH]; // output attitude file
FILE *foutAtt;      // pointer to output attitude file
char outPos[MAXPATH]; // output position file
FILE *foutPos;      // pointer to output position file
char outMean[MAXPATH]; // output mean file
FILE *foutMean;      // pointer to output mean file
int flgPr;           // output flag 1->file 0->stdout

TIntAcc *intacc;      // B-spline interpolates acceleration
TIntGyr *intgyr;      // B-spline interpolates for gyros data
TSimul1 *simul;

char *dar;            // raw increments
char *dwr;            // raw increments
long NO,Nf;           // initial and final points of the sample data
long N;               // mean values counter
long freq;            // sampling frequency
double att[3];        // attitude
double pos[3];        // position
double meanatt[3];    // mean attitude
double meanpos[3];    // mean position
char flag;            // error flag

void WriteData();      // write the results to the output files

```

```

public:
    TNavSimul(int flg,long iterations0);
    ~TNavSimul();

    void run();
}

```

mainnav.cpp

This is the file where the main function is implemented. It creates the instance of the TNavSimul and verifies the command line input, to generate the output to be piped through python to be plotted by the stripchart.py script.

```

int main(int argv, char *argc[])
{
    int flg = 1;          // not python interface
    long iterations = 10;

    // read the possible flg as input command
    //(0->output to stdout (Python pipe))
    if(argv > 2)
    {
        flg = atoi(argc[1]);
        iterations = atoi(argc[2]);
    }
    else if(argv > 1)
        flg = atoi(argc[1]);

    TNavSimul *NavSimul = new TNavSimul(flg,iterations);
}

```



```

    NavSimul->run();

    delete NavSimul;

    return 0;
}

```

datagen.cpp

This is the generator of data to the `navsimul` program. It was first used to generate piped data through python for the first tests of `stripchart.py`, but it was also used to generate the data files used for the static vehicle test of the `navsimul` program with the simulation files from Carole Bolduc's work [Bolduc 95].

The data output format is the same as the DREO files. It generates data points for a static vehicle in a defined position and attitude, with the corresponding bias and drift terms of the accelerometers and gyros. To define these parameters it uses the configuration file `datagen.cfg`.

```

class TDataGen
{
    double wander;
    double latitude;
    double longitude;
    double roll;
    double pitch;
    double yaw;
    double g;
    double omega;
}

```

```

    double GK;
    double *w;
    double *GB;
    double *Qd;
    double *MU;
    double *a;
    double *AB;
    double **Q;
    double **D;
    long count;
    FILE *faccddata;
    FILE *fgyrdata;

    void InitMatrices();
    void GenerateDt();
public:
    TDataGen(double wan, double lat,
              double lon, double ro,
              double pit, double ya, double g0,
              double omega0, double *AB0,
              double GK0, double *GB0,
              double *MU0, double *Qd0,
              FILE *faccdt, FILE *fgyrdt);
    ~TDataGen();

    void NewOrientation(double wan,
                       double lat, double lon,
                       double ro, double pit, double ya);
    void OutputDt();
}

```

Appendix C

Configuration Files

hru.cfg

AB	0.00292658083	0.0004389546	0.009026187	
AK	1	1	1	
A	1	0	0	
A	0	1	0	
A	0	0	1	
GB	-7.9482	-2.6698	6.7845	0.84124
GK	1	1	1	1
MU	1.191	-3.0981	1.233	-0.5190
Qd	7.743	-0.6642	7.680	-2.9654
G	1.0	0.0	0.0	0.0
G	0.0	1.0	0.0	0.0
G	0.0	0.0	1.0	0.0

G	0.0 0.0 0.0 0.0
wr0	0 0 0
ar0	0 0 0
Vc0	0 0 0
Tau	1080
act	0.43
Ro	6378140
R	3000000
Omega	15.0417
B	0 0 0
gm	9.806
lat	48
long	-123
wander	0
roll	0
pitch	0.001
yaw	0
Nar	20
Nwr	20
NQ	16
armax	5
Mk	3.92
wrmax	1.74533E-3
Mkw	401.32

MTau	8.75536E-2
Mact	3.57143E-2
MR	0.1
MOmega	5.60343E+0
M2	2
M2q	0.5
Kt	1.08925E-5

inout.cfg

InputDir=	./
OutputDir=	./
InputAcc=	hru_acc.dat
InputGyr=	hru_gyr.dat
ResultAtt=	res_att.2nd
ResultPos=	res_pos.2nd
ResMean=	mean.2nd
N0=	1
Nf=	1461

datagen.cfg

wander	0
lat	48
long	-123

roll	0
pitch	0
yaw	0
Omega	15.0417
gm	9.806
AB	0.002927 0.000439 0.009026
GK	1
GB	-7.948200 -2.669800 6.784500 0.841240
MU	1.191 -3.0981 1.233 -0.5190
Qd	7.743 -0.6642 7.680 -2.9654
AccData	acc_data.dat
GyrData	gyr_data.dat
Npts	200

stripchart.conf

0.autorange	10000.0		
1.iterations	1000		
2.shiftby	1000		
3.numgraphs	9	4.ynumpts	[4,3,3,3,6,3,3,3,6]
5.y2axishide	yes		
6.graphtitles1	Quaternions		
7.graphtitles2	True_Angular_Rates		
8.graphtitles3	Damped_Rates		

```
9.graphtitles4  Dii
10.graphtitles5 Dij
11.graphtitles6 Velocity
12.graphtitles7 Acceleration
13.graphtitles8 Qii
14.graphtitles9 Qij
15.linetimes1   ["q1","q2","q3","q4"]
16.linetimes2   ["mux","muy","muz"]
17.linetimes3   ["rhox","rhoy","rhoz"]
18.linetimes4   ["11","22","33"]
19.linetimes5   ["12","13","21","23","31","32"]
20.linetimes6   ["Vx","Vy","Vz"]
21.linetimes7   ["Ax","Ay","Az"]
22.linetimes8   ["11","22","33"]
23.linetimes9   ["12","13","21","23","31","32"]
24.psfiles1     q.ps
25.psfiles2     mu.ps
26.psfiles3     rho.ps
27.psfiles4     pos.ps
28.psfiles5     att.ps
29.psfiles6     vc.ps
30.psfiles7     ac.ps
31.psfiles8     Qii.ps
32.psfiles9     Qij.ps
```

stripgraph.conf (640x480)

```
0.tickfont      -Clean-Bold-R-Normal-*7-140-*7
1.width          640
2.height         430
3.graphwidth     220
4.graphheight    140
```

optionDB (640x480)

```
*font:          clean 7
```