NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

REAL-TIME WIND ESTIMATION AND DISPLAY FOR CHEM/BIO ATTACK RESPONSE USING UAV DATA

by

Cristián Sir

June 2003

Thesis Advisor: Isaac Kaminer
Co-Advisor: Vladimir Dobrokhodov

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** June 2003	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE: Real-Time Wind Estimation and Display for Chem/Bio Attack Response using UAV Data.	**5. FUNDING NUMBERS**
6. AUTHOR(S) Cristián Sir	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(maximum 200 words)***

The defense response to a Chemical and Biological attack would be importantly based on predicting the dispersion of a toxic cloud. Considering that an Unmanned Air Vehicle would provide the capability for embedding and positioning inertial and air data sensors geographically as required, real-time wind estimation can be performed for every actual position of the flying device in order to predict the plume moving direction. The efforts in this thesis concentrate on the demonstration and validation of procedures for obtaining Wind Estimation close to real-time and its instantaneous display. The presented work is based on a particular UAV platform available at the NPS Aeronautical Department and it aims to establish a general methodology, which may be used on other flying devices with similar available sensors. An accurate estimation of real wind for a particular combat scenario will enable operational units to have a near real-time decision aid. This final result could be integrated into a Command and Control net, to assist in a focused way the response to a Chemical and Biological attack and to map the source or the region to be affected.

14. SUBJECT TERMS Unmanned Air Vehicle, UAV, Wind Estimation, Real-Time Workshop, xPC Target, SIMULINK, Inertial Measurement Unit, IMU, and Coordinate Systems.	**15. NUMBER OF PAGES** 88
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

REAL-TIME WIND ESTIMATION AND DISPLAY FOR CHEM/BIO ATTACK RESPONSE USING UAV DATA

Cristián Sir
Lieutenant Commander, Chilean Navy
B.Eng. (Electronics), Educational Headquarters, 1992

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN AERONAUTICAL ENGINEERING WITH AVIONICS SUBSPECIALTY

from the

NAVAL POSTGRADUATE SCHOOL
June 2003

Author: Cristián Sir

Approved by: Isaac Kaminer
Thesis Advisor

Vladimir Dobrokhodov
Co-Advisor

Max F. Platzer
Chairman
Department of Aeronautics and Astronautics

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The defense response to a Chemical and Biological attack would be importantly based on predicting the dispersion of a toxic cloud. Considering that an Unmanned Air Vehicle would provide the capability for embedding and positioning inertial and air data sensors geographically as required, real-time wind estimation can be performed for every actual position of the flying device in order to predict the plume moving direction. The efforts in this thesis concentrate on the demonstration and validation of procedures for obtaining Wind Estimation close to real-time and its instantaneous monitoring. The presented work is based on a particular UAV platform available at the NPS Aeronautical Department and it aims to establish a general methodology, which may be used on other flying devices with similar available sensors. An accurate estimation of real wind for a particular combat scenario will enable operational units to have a near real-time decision aid. This final result could be integrated into a Command and Control net, to assist in a focused way the response to a Chemical and Biological attack and to map the source or the region to be affected.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

First and foremost, I would like to thank Dr. Vladimir Dobrokhodov for guiding me through most of the work presented on this thesis. His patience and understanding gave me the motivation to keep moving forward and accomplishing the numerous challenges involved. It was a true pleasure to have applied the engineering tools learned at NPS under Dr. Dobrokhodov's working methodology and logical supervision.

Secondly, many thanks go to Mr. Jerry Lentz for making every technical aspect involved in this work possible and for his many insights during my research. His breadth of knowledge never ceased to amaze me.

Finally, I would like to express my gratitude to Dr. Isaac Kaminer for his academic advise and for giving me the chance to join his great Controls Systems team where I had the opportunity to work with Dr. Oleg Yakimenko and Dr. Dobrokhodov. I learned a lot and really enjoyed working with them.

There's also another great team that put a lot of work in this thesis and during my rewarding stay at Monterey: Ana María, Cristián Jr. and María José. Everything is dedicated to you guys.

THIS PAGE INTENTIONALLY LEFT BLANK

DISCLAIMER

The mention of commercial products in this thesis does not imply endorsement by the United States of America Government.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Chemical and biological (Chem/Bio) weapons have posed a defense response security concern for some time and have gained a renewed focus due to actual tactical scenario threats. In order to successfully respond to attacks by such weapons the dispersion of a toxic cloud is important to measure and predict. Work in this thesis supported a Chem/Bio response project, which was a joint effort between the NPS Aeronautics and Astronautics Department and Meteorology Department to demonstrate and validate a method for the synthesis of measurements and predictions to aid in the response to a chemical and biological attack.

One of the key features of the Chem/Bio response project was the prediction of a toxic plume movement, which strongly depends on the actual wind velocity in the combat scenario. Using an Unmanned Air Vehicle (UAV) for geographically positioning inertial and air data sensors, obtained measurements and sensed data was used as the input for a Real-Time Wind Estimation solution.

This thesis research concentrates on the demonstration and validation of procedures for obtaining Wind Estimation close to real-time and its instantaneous monitoring. The presented work is based on a particular UAV platform available at NPS Aeronautical Department and it aims to establish a general methodology, which may be used on other flying devices with similar available sensors.

An accurate estimation of real-time wind for a particular combat scenario will enable Operational Units to have an important decision aid which can be later integrated into a Command and Control net to assist in a

focused way the response to a Chem/Bio attack. Furthermore, this real wind estimation can be used to map the source of the plume as well as the region to be affected.

A. OBJECTIVES

The starting point for this thesis research is the identification and understanding of available data that will be generated on an UAV platform and that will be available on a ground station as it is transmitted via Data Link.

This available data includes a Differential Global Position System (DGPS), an Inertial Measurement Unit (IMU), an Air Data System and a Data Link assembly; all of them embedded onboard the NPS FROG UAV. All the necessary background to understand and identify the generation of the inputs that will be later used for Wind Estimation is included in Chapter Two (II.). It also includes a brief description of the UAV and the Data Link system used.

Chapter Three (III.) will focus on the development of the proposed Wind Estimation Model. This work used SIMULINK for solving necessary applied mathematics in the Wind Estimation calculation and xPC Target software package to carry out the required real-time signal processing and results presentation.

Chapter Four (IV.) will cover the flight test setup and data acquisition that lead to the model validation. Flight test results will be discussed and the model will be evaluated against the meteorological wind measurements. Finally, on Chapter Five (V.) recommendations and conclusions will be presented.

II. BACKGROUND

A. THE UAV PLATFORM (NPS FROG)

The FROG UAV is a small high wing monoplane used for Digital Control System research by the Department of Aeronautics of the Naval Postgraduate School. The airplane is manufactured by BAI Aerosystems Inc., as the “BAI TERN” (Tactically Expendable Remote Navigator), and was formerly designated the FOG-R by the U.S. Army. In the FOG-R configuration the airplane has been flown non-line-of-sight using a fiber optic Data Link for command uplink and video downlink. The wing is fitted with flaps that can be trimmed to provide slow flight speeds for surveillance and extended to facilitate tight landings.

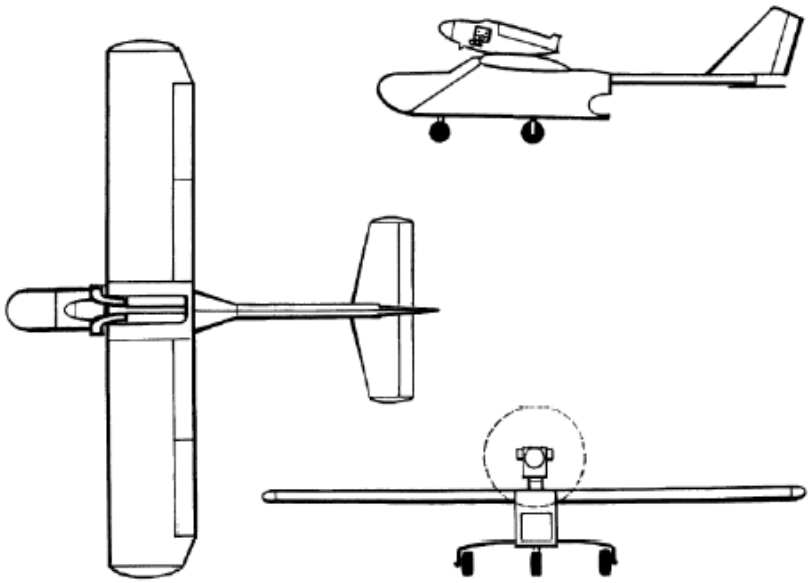


Figure II.1 FROG UAV Three View Drawing.

FROG is equipped with conventional elevator, rudder, ailerons and flaps. Small servomotors, designed for use in radio-controlled airplanes, actuate the control surfaces.

This UAV was designed to carry up to twenty-two pounds of payload for periods of up to four hours. It is actually equipped with a Model BA64 6.4 cubic inch, horizontally opposed, piston engine, manufactured by Brinson Aircraft Company. The 2-cylinder engine developed 9.3 Hp and is equipped with a two bladed propeller mounted in a tractor orientation in a nacelle atop the wing, as depicted in Figure (II.2). The FROG has fixed tricycle landing gear with a steer able nose wheel. The empennage is connected to the body of the airplane by a 1.75 inch diameter aluminum tube.

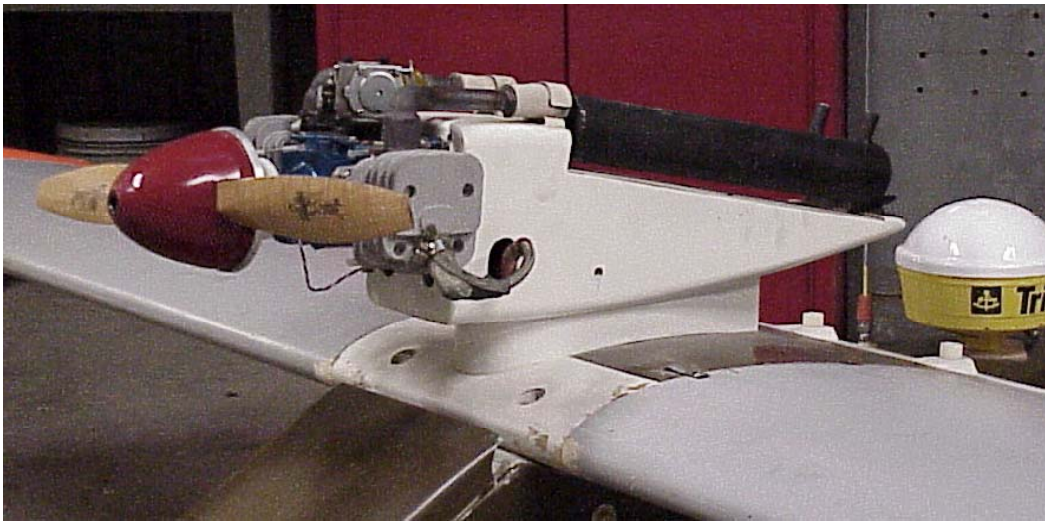


Figure II.2 FROG UAV Engine Configuration.

FROG is currently in use as a test bed for airborne sensor systems and advance control projects for the Aeronautics and Astronautics Department at NPS. In the past, the NPS's FROG had been configured with a variety of sensors including an onboard autopilot, various inertial measurement units, GPS receivers, an instrumented nose boom and a digital camera.

The FROG's significant physical characteristics are presented in Table II.1.

PARAMETER	MEASUREMENT
Length	8.125 ft
Height	1.75 ft
Weight	67.7 lbs
Power Plant	9.3 Hp / 2 Cycle
Wing Airfoil	NACA 2415
Horizontal Stabilizer Airfoil	NACA 0006 (Approx.)
Wing Span (b)	126.5 in
Tail Span (b_w)	39.75 in
Vertical Tail Span (b_v)	15.0 in
Aspect Ratio (AR)	6.32

Table II.1 FROG UAV Physical Characteristics

More details on the FROG characteristics and its engine are documented in Appendix A [Ref. 1].

Figure II.3 FROG UAV Actual Configuration

B. DIFFERENTIAL GLOBAL POSITIONING SYSTEM (DGPS)

The GPS receiver used on the FROG is Trimble Ag132 DGPS shown in Figure (II.4). The Ag132 DGPS is a 12 channel L-band differential correction receiver that provides sub-meter accuracy. It combines a GPS receiver, a beacon differential receiver, and a satellite differential receiver in the same housing. These receivers use a combined antenna with a single antenna cable. The Ag132 is configured with two programmable RS-232 serial ports and outputs GPS data at 1, 5 or 10 Hz with latency of 10 msec in RS-232 serial ASCII format at baud rates up to 38,400 bps. All outputs conform to the National Marine Electronics Association (NMEA)-0183 data protocol.

Figure II.4 DGPS Trimble Ag132 Main Components and Mounting Locations.

Among the various sentences in the GPS data stream shown in Table II.2, only information in the $GPGGA (GGA) and the $GPRMC (RMC) sentences is relevant to this application.

MESSAGE	CONTENTS
GGA	Time, position and fix related data.
GLL	Position fix, time of position fix and status.
GRS	GPS Range Residuals.
GSA	GPS position fix mode, SVs used for navigation and DOP values.
GST	GPS Pseudorange Noise Statistics.
GSV	Number of SVs visible, PRN numbers, elevation, azimuth and SNR values.
MSS	Signal strength, signal-to-noise ratio, beacon frequency and beacon bit rate.
RMC	UTC time, status, latitude, longitude, speed over ground (SOG), date and magnetic variation of the position fix.
VTG	Actual track made good and speed over ground.
XTE Message	Cross-track error.
ZDA	UTC time, day, month, year, local zone number and local zone minutes.
PTNLDG Proprietary	Beacon channel strength, channel SNR, channel frequency, channel bit rate, channel number, channel tracking status, RTCM source and channel performance indicator.
PTNLEV Proprietary	Time, event number, and event line state for time-tagging change of state on a event input line.
PTNL, GGK	Time, Position, Position Type and DOP Values.
PTNLID Proprietary	Receiver machine ID, product ID, major and minor release numbers and firmware release date.
PTNLSM	Reference Station Number ID and the contents of the Special Message included in valid RTCM Type 16 records.

Table II.2 DGPS Trimble Ag132 Messages

C. INERTIAL MEASUREMENT UNIT (IMU)

1. IMU Overview

The Inertial Measurement Unit (IMU) that was built for the FROG to be used for this research work consists of four separate sensors and two microprocessors that are used to integrate and transmit sensor measurements. The sensors

consist of the Trimble Ag132 DGPS, an Air Data assembly, a Microstrain 3DM accelerometer and magnetometer, and a Systrom Donner AQRS 104 rate gyros set. The microprocessor units are Tattletale8 Data Logger from Onset Corporation. The two microprocessors are programmed in Tattletale C (a variation of ANSI standard C) and control the timing, measurement and transmission of all sensors. The combined sensor output provides measurements of the complete state vector with the exception of heading angle ψ, which can be computed from the other components of the state vector. Table (II.3) shows the complete measurement values, associated sensors and the data rate at which the measurements are available.

STATE VECTOR	NOTATION	RATE	SENSOR	NOTES
Roll Rate	p	40 Hz	AQRS 104	4.096 Volt A to D sample
Pitch Rate	q	40 Hz	AQRS 104	4.096 Volt A to D sample
Yaw Rate	r	40 Hz	AQRS 104	4.096 Volt A to D sample
Temperature	*Temp*	40 Hz	OAT	4.096 Volt A to D sample
Alpha	α	20 Hz	AQRS 104	4.096 Volt A to D sample
Beta	β	20 Hz	AQRS 104	4.096 Volt A to D sample
Airspeed	*Vair*	40 Hz	Air Data	4.096 Volt A to D sample
Accel. X component	*Ax*	20 Hz	3DM	RS-232 9,600 baud
Accel. Y component	*Ay*	20 Hz	3DM	RS-232 9,600 baud
Accel. Z component	*Az*	20 Hz	3DM	RS-232 9,600 baud
Mag. Vect. X comp.	*Hx*	20 Hz	3DM	RS-232 9,600 baud
Mag. Vect. Y comp.	*Hy*	20 Hz	3DM	RS-232 9,600 baud
Mag. Vect. Z comp.	*Hz*	20 Hz	3DM	RS-232 9,600 baud
GPS Time	*t*	10 Hz	DGPS	RS-232 38,400 baud
GPS Latitude	*Lat*	10 Hz	DGPS	RS-232 38,400 baud
GPS Longitude	*Long*	10 Hz	DGPS	RS-232 38,400 baud
GPS Altitude	*Alt*	10 Hz	DGPS	RS-232 38,400 baud
GPS Ground Speed Knt	*GrndSpd*	10 Hz	DGPS	RS-232 38,400 baud
GPS Ground Track Deg	*GrndTrk*	10 Hz	DGPS	RS-232 38,400 baud
GPS Mag.Var.Deg	*MagVar*	10 Hz	DGPS	RS-232 38,400 baud

Table II.3 IMU State Vector Components

2. IMU Sensors Summary

Sensor data rate is an important element when handling data obtained by different components. Data rates of 20 Hz for the Euler Angles rates (*p*, *q*, & *r*) are known to be acceptable. Realistically a data rate of 30 to 40 Hz is required for the optimal digital signal processing. Each sensor has its own inherent limitations that restrict the maximum allowable data rate. Measurements from each sensor are taken at various data rates and merged together in order to achieve a high data rate yet maintain an easily decoded and error free data stream.

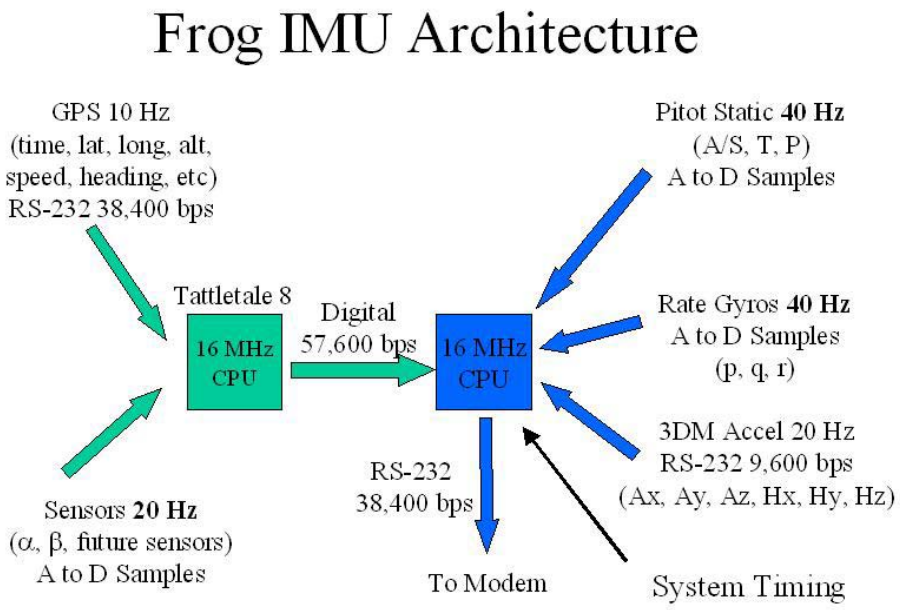


Figure II.5 IMU Architecture and Data Rates

a. Onset Tattletale 8 Microprocessor

The Tattletale8 Microprocessor from Onset Computer Corporation is designed to be a data logger that can either be programmed in Tx Basic or C Programming. The Tattletale8 has a Motorola 68332 microprocessor, 8 channels to make analog to digital conversion (A to D) samples using

a 4.096 volt reference, two RS-232 serial communications ports, a system clock adjustable from 160 KHz to 16 MHz, and 15 digital (0 to 5 volts) lines that can be used to transmit serial data at up to 500 Kbps.

As seen on previous Figure (II.5), two Tattletale8 are used to process the data and measurements made by the IMU. One Tattletale8 is dedicated to the processing of serial GPS data and taking A to D samples from the Air Data system. The second Tattletale8 takes the serial output of the 3DM, A to D samples for the IMU sensors (Rate Gyros and Air Speed), and controls all of the measurement and transmission timing. The GPS Tattletale8 transmits its data to the 3DM Tattletale8 via the digital lines.

The communication link between the UAV and the ground station computer is implemented with wireless modems. Each modem uses frequency hopping spread spectrum technology and has a power output of 1/3 Watts. They are capable of communicating over a line of sight with range of up to 20 miles, and to support data transmission at baud rates from 1200 bps to 115.2 Kbps. The ground station computer performs the real-time data logging and processes the transmitted data according to the Wind Estimation algorithm.

Figure II.6 Data Link Modems

b. DGPS Trimble Ag132 ASCII Format

The Trimble Ag132 GPS is capable of outputting data at 1, 5 or 10 Hz utilizing an RS-232 serial ASCII data format at baud rates up to 38,400 bps. All output conforms to the National Marine Electronics Association (NMEA) GPS data protocol used by GPS receivers to transmit data. Different lines of NMEA data can be selected to be included in the GPS data stream. The lines that contain relevant GPS information for this thesis are the $GPGGA and $GPRMC sentences. The setting of the GPS output used for these two sentences was 10 Hz at a baud rate of 38,400 bps. Tables (II.4) and (II.5) show the breakdown of the ASCII text output and an example conversion. Shaded values indicate information not included in final IMU data stream.

$GPGGA Sentence: Global Positioning System Fix Data			
EXAMPLE: $GPGGA, 001218.80, 3635.761652, N, 12152.607700, W, 2, 08, 1.1, 15.14, M, -27.47, M, 5.4, 0565*45			
Field	**Description**	**Data**	**Notes**
1	UTC of Position	001218.80	Fix at 00:12:18.80 UTC
2	Latitude	3635.761652	LAT 36° 35.761652′
3	N or S	N	North Latitude
4	Longitude	12152.60770	LONG 121° 52.6077′
5	E or W	W	West Longitude
6	GPS quality indicator 0=invalid 1=GPS fix 2=DGPS fix	2	DGPS Mode
7	Number of satellites in use	08	8 Sats. in use
8	Horizontal dilution of position	1.1	HDOP 1.1
9	Altitude +/- mean sea level (geoid)	15.14	15.14 Meters above S.L.
10	Meters (Antenna height unit)	M	Meters
11	Geoidal separation (Diff. between WGS-84 earth ellipsoid and mean sea level - = geoid is below ellipsoid)	–27.47	-27.47 below
12	Meters (Units of Geoidal sep.)	M	Meters
13	Age in seconds since last update from Diff. reference station	5.4	5.4 seconds old
14	Diff. reference station ID#	0565	Station #
15	Checksum	*45	Checksum

Table II.4 $GPGGA Sentence Structure

$GPRMC Sentence: Recommended Min. Spec. GPS/TRANSIT Data			
EXAMPLE: $GPRMC, 001218, A, 3635.761652, N, 12152.607700, W, 000.00, 0.0, 170401, 15.3, E, D*03			
Field	**Description**	**Data**	**Notes**
1	UTC of position fix	001218	Fix at 00:12:18.80 UTC
2	Data status (V navigation receiver warning)	A	Receiver Status Valid
3	Latitude of fix	3635.761652	LAT 36° 35.761652′
4	N or S	N	North Latitude
5	Longitude of fix	12152.60770	LONG 121° 52.6077′
6	E or W	W	West Longitude
7	Speed over ground in Knots (0-3 decimal places)	000.00	0.0 Knots ground speed
8	True Ground Track in degrees True	0.0	0.0° ground track
9	UTC date	170401	April 17, 2001
10	Magnetic Variation degrees (Easterly Var. subtracts from true course)	15.3	Magnetic Variation 15.3°
11	E or W	E	East
12	Checksum	D*03	Checksum

Table II.5 $GPRMC Sentence Structure

c. DGPS Trimble Ag132 Data Conversion

The $GPGGA and $GPRMC sentences combine 161 ASCII text characters. This data is received approximately 10 times per second and would take up about one third of the available bandwidth of the final IMU data stream at 38,400 bps, if all of the text values were transmitted. Therefore, it is necessary to sort out and transmit only the information required by the application. Transmitting the critical data in a binary format allows the total data stream to be reduced to 29 bytes of binary data. Each GPS data package includes a two-byte header to indicate the start of the GPS message, bringing the total GPS packet size to 31 bytes.

Data is transmitted in a binary format as a series of 8 bit bytes. The serial driver that is receiving the data stream must decode it before it can be used. Values that range between 0 and 255 are represented by a

single byte of data and no conversion is necessary. Values that range between 0 and 65535 are represented by a two byte integer with the most significant byte (MSB) transmitted first, followed by the least significant byte (LSB). To decode the two-byte integer the MSB must be multiplied by the placeholder value of 256 (2^8) and added to the LSB. When decoding entire bytes the placeholder values increase by factors of 2^8 (Ex: 2^0, 2^8, 2^{16}, 2^{24}, etc). This is a simple conversion that can be implemented in most programming languages. It is worth noting, however, that this only works for unsigned values. If signed values are used, the bits of the MSB must be shifted by the appropriate amount to preserve the integrity of whatever signing convention is used. Numbers greater than 65535 are represented by a four-byte long integer with the MSB transmitted first. The four byte sequence is decoded using the following equation: long integer = 2^{24}xByte1 + 2^{16}xByte2 + 2^8xByte3 + 2^0xByte4. Table (II.6) shows the format of the DGPS data and method required to convert the data to a useable number format.

DGPS Data Package Format

EXAMPLE: $GPGGA, 001218.80, 3635.761652, N, 12152.607700, W, 2, 08, 1.1, 15.14, M, -27.47, M, 5.4,0565*45
$GPRMC, 001218, A, 3635.761652, N, 12152.607700, W, 000.00, 0.0, 170401, 15.3, E ,D*03

Data Description	Notation	Bytes	Text	Binary	Decoding Method
GPS Data Package Header	GPS Header	2		238 238	Hex values EE EE
Hours (00 – 23)	timeHH	1	00	00	N/A
Minutes (00 – 59)	timeMM	1	12	12	N/A
Seconds (00 – 59)	timeSS	1	18	18	N/A
Decimal Seconds (0 – 9)	timeDecSS	1	8	08	N/A
Degrees LAT (00 – 90)	latDeg	1	36	36	N/A
Minutes LAT (00 – 60)	latMin	1	35	35	N/A

DGPS Data Package Format					
EXAMPLE: $GPGGA, 001218.80, 3635.761652, N, 12152.607700, W, 2, 08, 1.1, 15.14, M, -27.47, M, 5.4,0565*45 $GPRMC, 001218, A, 3635.761652, N, 12152.607700, W, 000.00, 0.0, 170401, 15.3, E ,D*03					
Data Description	**Notation**	**Bytes**	**Text**	**Binary**	**Decoding Method**
Decimal Minutes LAT (0 –7 decimal places)	latDecMin	4	761652	00 11 159 52	MSB 1st 2^{24}xByte1+2^{16}xByte2+ 2^{8}xByte3+2^{0}xByte4
Degrees LONG (00 – 180)	longDeg	1	121	121	N/A
Minutes LONG (00 – 60)	longMin	1	52	52	N/A
Decimal Minutes LONG (0 –7 decimal places)	longDecMin	4	607700	00 09 69 212	MSB 1st 2^{24}xByte1+2^{16}xByte2+ 2^{8}xByte3+2^{0}xByte4
DGPS status	diffGPS	1	2	02	N/A
Altitude in meters	altMeters	2	15	00 15	MSB 1st LSB 2nd 2^{8}xByte1+2^{0}xByte2
Altitude in tenths of meters	altDecMts	1	14	14	N/A
Groundspeed in knots	grndSpeed	2	000	00 00	MSB 1st LSB 2nd 2^{8}xByte1+2^{0}xByte2
Groundspeed in tenths of knots	grndSpeedDecimal	1	00	00	N/A
Ground track in degrees (0-359)	grndTrack	2	0	00 00	MSB 1st LSB 2nd 2^{8}xByte1+2^{0}xByte2
Ground track in tenths of degrees	grndTrackDecimal	1	0	00	N/A
Magnetic variation in degrees	magVar	2	15	00 15	MSB 1st LSB 2nd 2^{8}xByte1+2^{0}xByte2
Magnetic variation in tenths of degrees	magVarDec	1	3	3	N/A

Table II.6 DGPS Binary Data Package Format

d. DGPS Time Issues

At 10 Hz the DGPS should transmit the $GPGGA and $GPRMC sentences every 100 ms. For the DGPS signal the ASCII text data is reliably received at 10 samples per second without any dropouts on lost sample times. The exact timing of the arrival of the GPS data has a huge amount of variability. On average, data is received every 100 ms. A particular sample, however, may arrive early or it may be as late as 100 ms (i.e. 200 ms between samples). This creates great problems when regularly sampled inputs are

required. When the DGPS data does not arrive in time, the DGPS parsing routine waits for all of the data to arrive. Therefore, it is important that critical timing of measurements not to be based on or delayed by the arrival of the DGPS signal. Specific programming implementation regarding this problem must be addressed.

e. 3DM Accelerometer and Magnetometer

The 3DM module has an orthogonal array of DC accelerometers and magnetometers that measure all three components of the local acceleration and magnetic vectors. The unit is capable of transmitting computed roll, pitch, and yaw angles or raw acceleration and magnetic vectors. The 3DM uses the Earth’s gravity vector to compute the orientation of the sensor in pitch and roll direction. Using the magnetic vector and the pitch and roll angles, the 3DM can compute the yaw angle and therefore is able to determine all three Euler Angles. This method works well when the unit is in a static position; however, it will not work during accelerated flight onboard the UAV. When the UAV is in maneuvering flight (Ex: 60° AOB level turn at 2 Gs) the unit will incorrectly measure the Earth’s gravity vector and consequently generate erroneous Euler angle measurements. Therefore, only the raw acceleration and magnetic vectors are output from the 3DM and valid results for Wind Estimation are only expected for a close to straight and level flight condition.

The 3DM transmits RS-232 serial data at 9,600 bps. The device can be put into one of two communication modes: polled or continuous. When the device is in continuous mode it sends packets of data to the computer continuously. In polled mode the unit only sends data if a

controlling computer prompts it. This is a more robust communications mode that requires less error checking. The polled mode is normally selected by sending the 3DM the ASCII character “t”. The 3DM installed in the Frog IMU, however, has a special EEPROM that allows only the polled mode. This was done to solve a problem with the device unpredictable switching communication modes. Thus, the 3DM can only be operated in the polled mode.

When operating in polled mode it is necessary to send 3DM a specific one-byte command word before it will transmit its data. To receive the complete acceleration and magnetic vectors the binary value 10010000 (decimal number 144) must be sent to 3DM. After receiving the appropriate command word, 3DM will transmit the data shown in Table (II.7). This data is stored in a buffer in the Tattletale8 microprocessor and is transmitted at the appropriate time. The 3DM is polled and the sensor’s data is updated at a rate of 20 Hz. The remaining parameters of the IMU data package are measured at 40 Hz.

To simplify the decoding of the IMU data a standard IMU package is transmitted at 40 Hz. The 20 Hz 3DM data is only updated every other frame even though the 12 bytes of data is transmitted every single frame. A close examination of the 3DM output will reveal that the data occurs in repeated pairs. This may create some problems if the acceleration or magnetic data needs to be analyzed in the frequency domain. The data is transmitted MSB first and LSB second for each measured value. The bytes can be decoded as Value = 2^8xByte1 + 2^0xByte2. Table (II.7) shows the order of data.

Data	Description
Diagnostic Byte	0x41h if Valid; 0x6Xh if error (“X” is an error code)
H_{X-m}	X Axis Magnetometer Data MSB
H_{X-l}	X Axis Magnetometer Data LSB
H_{Y-m}	Y Axis Magnetometer Data MSB
H_{Y-l}	Y Axis Magnetometer Data LSB
H_{Z-m}	Z Axis Magnetometer Data MSB
H_{Z-l}	Z Axis Magnetometer Data LSB
A_{X-m}	X Axis Accelerometer Data MSB
A_{X-l}	X Axis Accelerometer Data LSB
A_{Y-m}	Y Axis Accelerometer Data MSB
A_{Y-l}	Y Axis Accelerometer Data LSB
A_{Z-m}	Z Axis Accelerometer Data MSB
A_{Z-l}	Z Axis Accelerometer Data LSB

Table II.7 3DM Magnetometer and Accelerometer Data Format

f. AQRS 104 Rate Gyros

The Systrom Donner AQRS 104 rate gyros measure the angle rates of change (*p*, *q*, & *r*) about the Body-Fixed axis. The angle rates are measured by an Analog to Digital converter sampled from the Tattletale8. Since measurements only require an A to D sample, the only timing issue involved in determining the maximum sample rate is the time required for the Tattletale8 to make an A to D sample. Therefore the maximum sample rate is only limited by the rate at which the Tattletale8 can make A to D samples and transmit them to the modem. Current software is set to sample at 40 Hz. The three sensors are actually wired to the Tattletale8 in the order (*p*, *r*, & *q*), so the samples

have been reversed in software to provide the conventional order (*p*, *q*, & *r*) in the output data stream.

g. IMU Data Format

In order to distinguish each package of data a two-byte header is added to the beginning of each data package as shown on Table (II.8):

Data Header Formats				
Header (Hex)	**Header (Decimal)**	**Data Type**	**Data Size**	**Order**
FF FF	255 255	IMU TT A to D Data	28 Bytes	MSB, LSB
EE EE	238 238	GPS Data	29 Bytes	MSB, LSB
DD DD	221 221	GPS TT A to D Data	16 Bytes	MSB, LSB

Table II.8 IMU Data Headers

D. AIR DATA

1. Air Speed

Two basic pressures are used for measurement of airspeed, static and total pressure. The static pressure is the atmospheric pressure at the flight level of the aircraft, while the total pressure is the sum of the static and the impact pressure, which is the pressure developed by the forward speed of the aircraft. The relation of the three pressures can thus be expressed by the following equation:

$$p_t = p + q_c \quad (1)$$

Where p_t is the total pressure, p is the static pressure, and q_c the impact pressure.

In incompressible flow, the pressure developed by the forward motion of a body is called the dynamic pressure q, which is related to the true airspeed V by the equation:

$$q = \frac{1}{2}\rho V^2 \qquad (2)$$

From Equation (2), ρ is the density of the air and V is the speed of the aircraft relative to the air.

For compressible flow, the measured impact pressure q_c is higher than the dynamic pressure and the effects of compressibility must be taken into account. Since the FROG operates in the low subsonic range, compressibility effects are ignored.

The airspeed of the FROG is computed based on dynamic pressure measurement using a pitot-static probe mounted at a wingtip and pressure transducers. The dynamic pressure is “fed” into a pressure transducer that in turn converts it to the analog voltage signal. With proper calibration and application of Equation (2), the airspeed of the UAV can be computed.

The pitot-static probe is a straight 26 inches long conventional type with four static pressure sensing ports located 1.125 inches aft of the total pressure port.

The pressure transducer used is a 0–4 inches H_2O differential pressure transducer that gives an output signal of 0 to 5 volts.

Figure II.7 Pitot-Static Probe Layout

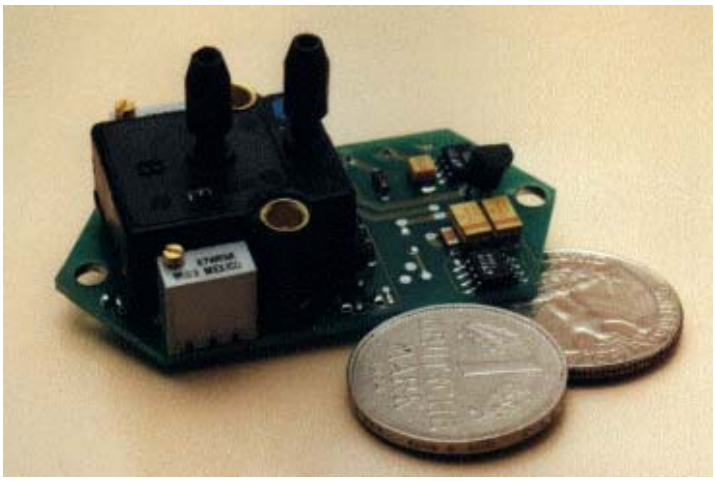
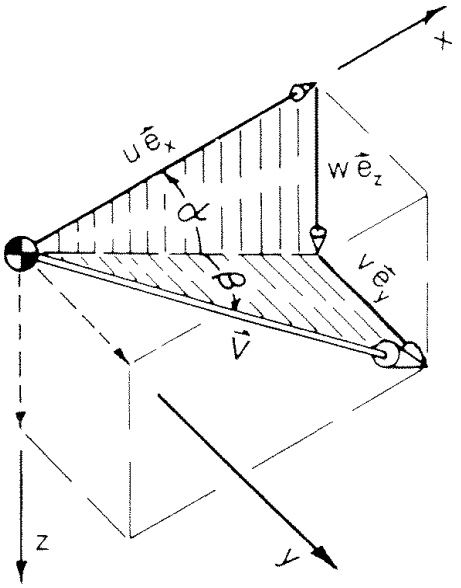


Figure II.8 Pressure Transducer

Figure II.9 Pitot-Static Probe Mounted on FROG

2. Angle of Attack and Sideslip Angle Sensor

Angle of Attack (Alpha, α, AOA) is defined as the angle between the relative wind in the plane of symmetry and the longitudinal axis of the aircraft. Sideslip Angle (Beta, β) is defined as the angle between the wind vector and the plane of symmetry. Figure (II.9) illustrate these definitions:



$$\alpha = \tan^{-1}\frac{w}{u} \qquad \beta = \sin^{-1}\frac{v}{V}$$

$$\alpha_f = \tan^{-1}\frac{v}{u}$$

Figure II.10 α and β Definition

Wind vanes mounted on potentiometers are used to measure α and. β Note that the β vane actually measures "flank angle of attack" but since α is small, true β can be approximated by "flank angle of attack".

The vane-potentiometer assembly is mounted on a probe that is similar to the pitot-static probe at the wing tip. The wind vanes are attached to the shaft of the potentiometer. As the UAV pitches and/or yaws, the vanes rotate and that causes the shaft to rotate. The rotation of the shafts changes the resistance of the potentiometers and

an analog output voltage signal is produced. With proper calibration, the α and β angles of the FROG are obtained.

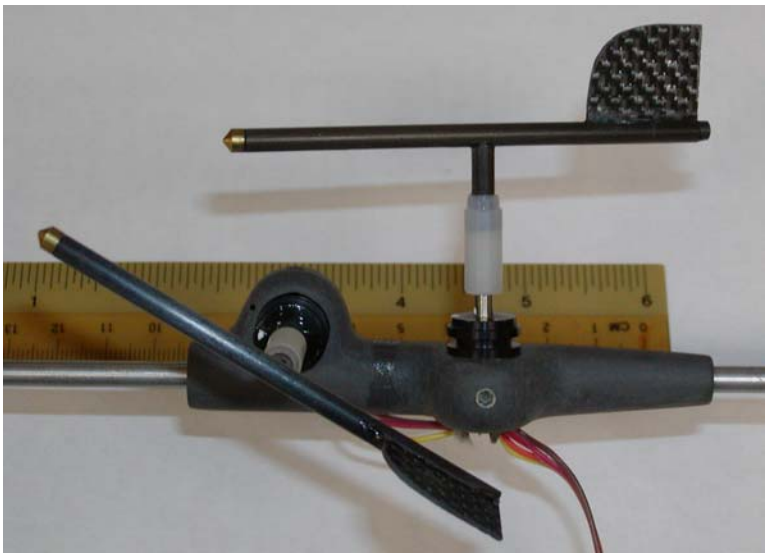


Figure II.11 α and β Vanes Mounted on Potentiometers

Further details about air data capture, calibration and processing can be found in Chapter three (III.) and [Ref. 2].

E. COORDINATE SYSTEMS

To develop the relationship between the GPS and the IMU that is needed for attempting a Wind Estimation solution, an understanding of coordinate systems involved is essential.

Four different coordinate systems are used in this thesis:

- True Inertial Coordinate System {*I*}
- Local Tangent Plane Coordinate System {*LTP*}
- Body-Fixed Coordinate System {*b*}
- Wind or Flight Path Coordinate System {*w*}

For the purposes of this thesis, the rotation of the earth and its associated Coriolis' forces can be ignored

and the Local Tangent Plane Coordinate System can be considered to be a True Inertial Coordinate System. Further information about Coordinate Systems can be found in [Ref. 3].

1. True Inertial Coordinate System {*I*}

The True Inertial Coordinate System is a set of mutually perpendicular axes that neither accelerate nor rotate with respect to some fixed point in space. Newton assumed there was a reference frame whose absolute motion was zero, fixed relative to the stars, and it is in this reference frame where Newton's laws are valid. However, Newton's laws of motion can also be applied to any reference frame as long as the proper coordinate transformations are used.

2. Local Tangent Plane Coordinate System {*LTP*}

This coordinate system is defined by extending a ray from the center of the earth to its surface. A plane is attached tangent to the point of intersection of the ray with the Earth's surface and this point becomes the origin of the system. While it is somewhat arbitrary, for our purposes it is defined the positive *x-axis* direction as pointing true east, the positive *y-axis* direction as pointing true north, and the positive *z-axis* direction as pointing up (away from the center of the earth). This is depicted in Figure (II.12).

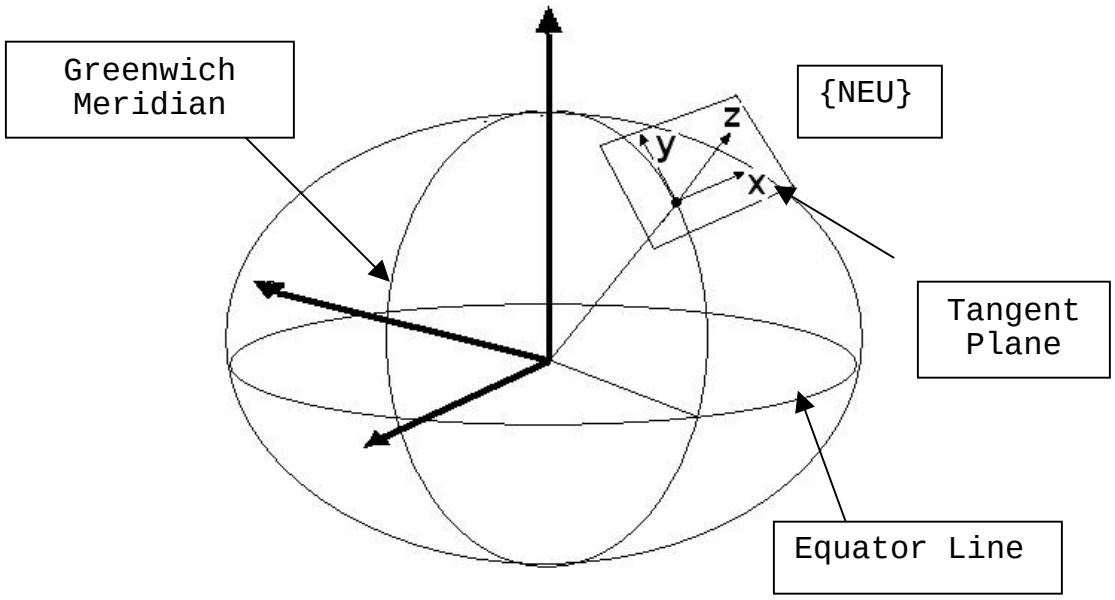


Figure II.12 Local Tangent Plane Coordinate System

3. Body-Fixed Coordinate System {*b*}

The Body-Fixed Coordinate System is a right hand orthogonal system with the origin at the center of gravity of the air vehicle. The positive *x-axis* direction points toward the nose. The positive *y-axis* direction points out the right wing and the positive *z-axis* direction points towards the bottom of the air vehicle. The velocity of the air vehicle with respect to the Inertial Coordinate System, resolved along the *x*, *y*, and *z* axes of the Body-Fixed Coordinate System, are termed *u*, *v*, and *w*, respectively. The angular rate of rotation of the air vehicle with respect to the Inertial Coordinate System, resolved in the Body-Fixed Coordinate System, are called *p*, *q*, and *r*, respectively. Positive values for angular rates in the Body-Fixed frame are shown in Figure (II.12).

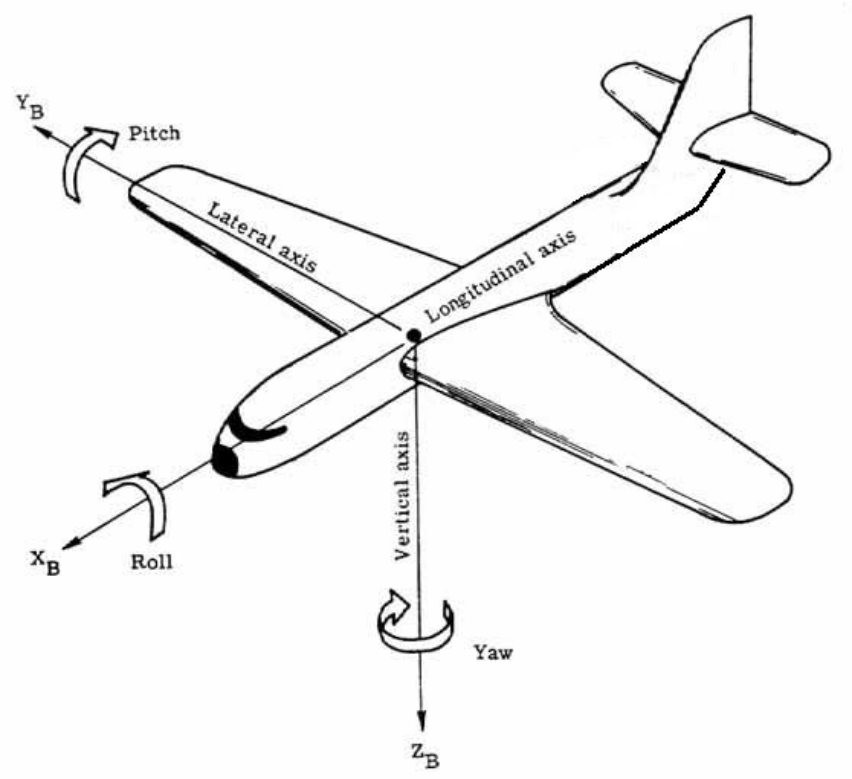


Figure II.13 Body Frame Angular Rates Definition

4. Wind or Flight Path Coordinate System {*w*}

The Wind Coordinate System is also a right hand orthogonal system with its origin at the center of gravity of the air vehicle. The *x-axis* is aligned with the velocity vector of the air vehicle. The orientation of the Wind Coordinate System with respect to the Body-Fixed Coordinate System is defined in terms of the angles α and β. The equations for α and β are given below:

$$\alpha = \tan^{-1}\left(\frac{w}{V}\right) \qquad (3)$$

$$\beta = \sin^{-1}\left(\frac{v}{V}\right) \qquad (4)$$

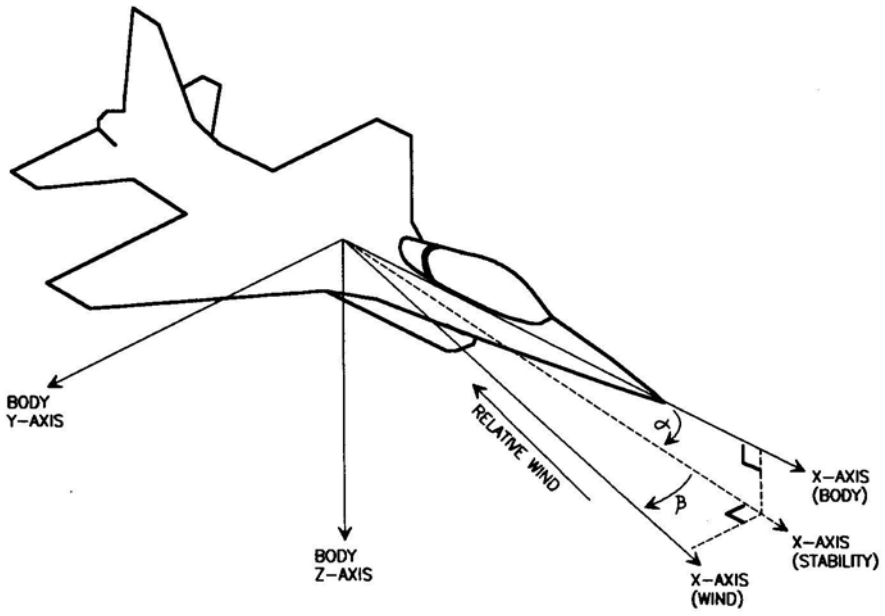


Figure II.14 Wind Coordinate System Definition

5. Coordinate Transformations

In order to use the coordinate systems mentioned above, one must be able to transform between them freely. For this thesis two transformations are going to be used:

a. Body to Inertial {b} to {I}

The Euler Angles ϕ, θ and ψ, named roll, pitch, and yaw are defined in order to express the orientation of the Body-Fixed Coordinate System with respect to the Inertial Coordinate System. For the purposes of this thesis, a 3-2-1 Euler angle transformation will be used. The 3-2-1 transformation is given without an explanation but a good development can be found in [Ref. 3 and 4]. Nature of the angular rotation is more apparent when the transformation is expressed as the product of three rotation matrices. In the case of a 3-2-1 rotation sequence, the three matrices in Equation (5) correspond to

rotations about the yaw, pitch, and roll axes of the air vehicle.

$$ {}^{i}\mathbf{V} = \begin{bmatrix} \cos\Psi & \sin\Psi & 0 \\ -\sin\Psi & \cos\Psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} {}^{b}\mathbf{V} \quad (5) $$

Of course, the three matrices can be multiplied out for an analytic result contained in a single matrix:

$$ {}^{i}\mathbf{V} = \begin{bmatrix} \cos\theta\cos\Psi & -\cos\phi\sin\Psi + \sin\phi\sin\theta\cos\Psi & \sin\phi\sin\Psi + \cos\phi\sin\theta\cos\Psi \\ \cos\theta\sin\Psi & \cos\phi\cos\Psi + \sin\phi\sin\theta\sin\Psi & \sin\phi\cos\Psi + \cos\phi\sin\theta\sin\Psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix} {}^{b}\mathbf{V} \quad (6) $$

Where ${}^{i}V$ is a free vector resolved in $\{I\}$ and ${}^{b}V$ is the same vector resolved in $\{b\}$. The inverse is also defined, since the transformation is orthonormal. The inverse is the transpose of the rotation matrix shown in Equation (6).

As previously mentioned, the IMU used on the FROG for this thesis is equipped with rate gyros that provide the angular velocity components in the Body Coordinate System (p, q and r). The body reference frame's angular rate can be related to the change of Euler Angles by a transformation matrix [Ref.4]. This is given by:

$$ \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\Psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \quad (7) $$

b. Wind to Body {w} to {b}

The angles α and β define the orientation of the Wind Coordinate System with respect to the Body-Fixed Coordinate System, therefore a transformation matrix can be obtained that relates a free vector resolved in {*w*} to the same vector resolved in {*b*}. The transformation is expressed as:

$$ {}^{b}\boldsymbol{V} = \begin{bmatrix} \cos\alpha\cos\beta & -\cos\alpha\sin\beta & -\sin\alpha \\ \sin\beta & \cos\beta & 0 \\ \sin\alpha\cos\beta & -\sin\alpha\sin\beta & \cos\alpha \end{bmatrix} {}^{w}\boldsymbol{V} \qquad (8) $$

Where ${}^{b}V$ is a free vector resolved in {*b*} and ${}^{w}V$ is the same vector resolved in {*w*}.

F. WIND ESTIMATION THEORY

As stated in the previous point (E.), for this thesis Local Tangent Plane Coordinate System {LTP} is considered to be True Inertial Coordinate System {*I*}, therefore the Wind Estimation can be obtained by solving the following equation:

$$ {}^{i}\boldsymbol{V}_{w} = {}^{i}\boldsymbol{V}_{b,i} - {}^{i}_{w}\boldsymbol{R}\boldsymbol{V}_{b,w} \quad (9) $$

The vector ${}^{i}V_{w}$ stands for “wind velocity in the inertial frame” which is actually the True Wind that needs to be estimated.

The vector ${}^{i}V_{b,i}$ stands for “velocity of the body with respect to the inertial frame solved in the inertial frame”. ${}^{i}V_{b,i}$ is given directly by the DGPS and for Wind Estimation purposes only components of the *x-axis* and *y-axis* are considered.

The vector $V_{b,w}$ stands for “velocity of the body with respect to the wind frame”.

The matrix ${}^{i}_{w}R$ represents the transformation matrix form the Wind Coordinate System to the Inertial Coordinate System, where:

$$ {}^{i}_{w}\boldsymbol{R} = {}^{i}_{b}\boldsymbol{R}\,{}^{b}_{w}\boldsymbol{R} \quad (10) $$

The matrices ${}^{i}_{b}R$ and ${}^{b}_{w}R$ are rotational matrices defined by Equations (6) and (7), therefore when both matrices are applied to a free vector resolved in the Wind Coordinate System $\{w\}$, the same vector resolved in the Inertial Coordinate System $\{I\}$ is obtained.

$$ {}^{i}_{w}\boldsymbol{R}\boldsymbol{V}_{b,w} = {}^{i}_{b}\boldsymbol{R}\,{}^{b}_{w}\boldsymbol{R}\boldsymbol{V}_{b,w} = {}^{i}\boldsymbol{V}_{b,w} \qquad (11) $$

The vector ${}^{i}V_{b,w}$ stands for “velocity of the body with respect to the wind frame solved in the inertial frame”.

Equation (5) that solves for Wind Estimation can be graphically represented by:

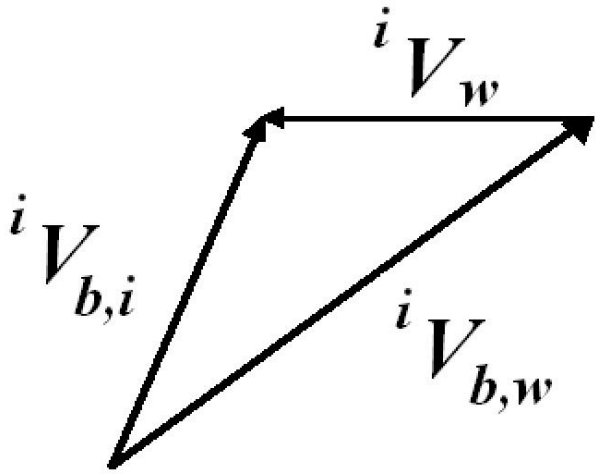


Figure II.15 Wind Estimation Solution

THIS PAGE INTENTIONALLY LEFT BLANK

III. WIND ESTIMATION MODEL

Due to the complexity of the model at hand it will be presented using smaller modules.

The general layout of the Wind Estimation Model is shown in Figure (III.1):

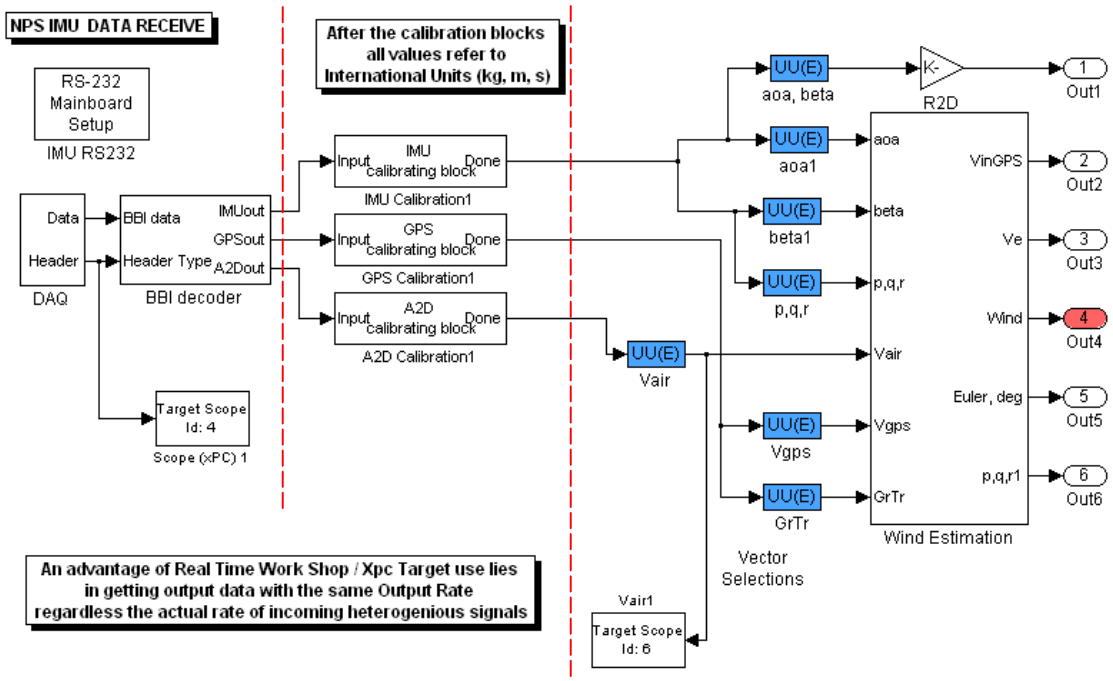


Figure III.1 Wind Estimation Model Layout

Figure (III.1) includes three main sections. The right hand side module computes Wind, the central module executes necessary calibrations of the incoming data in order to obtain physical meaning of it, and the left hand side module addresses data capture, i.e., capturing data coming from the FROG's IMU through both Tattletale8 and decoding it.

The above solution was constructed in a SIMULINK environment and subsystems/blocks from the xPC Target

library where included in the capturing/decoding module, so a Real-Time Workshop model could be later build.

A. WIND ESTIMATION SOLUTION MODULE

The Wind Estimation solution will be based on the theory presented on Chapter two (II). Recall Equation (8), (9), (6) and (7).

$$ {}^{i}\boldsymbol{V}_{w} = {}^{i}\boldsymbol{V}_{b,i} - {}^{i}_{w}\boldsymbol{R}\boldsymbol{V}_{b,w} \quad (8) $$

$$ {}^{i}_{w}\boldsymbol{R} = {}^{i}_{b}\boldsymbol{R}\,{}^{b}_{w}\boldsymbol{R} \quad (9) $$

$$ {}^{i}\boldsymbol{V} = \begin{bmatrix} \cos\theta\cos\Psi & -\cos\phi\sin\Psi + \sin\phi\sin\theta\cos\Psi & \sin\phi\sin\Psi + \cos\phi\sin\theta\cos\Psi \\ \cos\theta\sin\Psi & \cos\phi\cos\Psi + \sin\phi\sin\theta\sin\Psi & \sin\phi\cos\Psi + \cos\phi\sin\theta\sin\Psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix} {}^{b}\boldsymbol{V} \quad (6) $$

$$ {}^{b}\boldsymbol{V} = \begin{bmatrix} \cos\alpha\cos\beta & -\cos\alpha\sin\beta & -\sin\alpha \\ \sin\beta & \cos\beta & 0 \\ \sin\alpha\cos\beta & -\sin\alpha\sin\beta & \cos\alpha \end{bmatrix} {}^{w}\boldsymbol{V} \quad (7) $$

From the above set of equations, Wind Estimation will be obtained by solving for ${}^{i}V_{w}$ (wind velocity in the inertial frame). This will be the output of interest for the estimation model and the following inputs will be needed:

- ${}^{i}V_{b,i}$ (for the model $V_{in}GPS$): Obtained from the DGPS ground speed. For Wind Estimation purposes only components of the *x-axis* and *y-axis* are to be calculated and considered. In this component calculation, DGPS heading (for the model G_rT_r) and DGPS ground speed (for the model V_{gps}) are required as the inputs.

- $V_{b,w}$ (for the model V_{air})): Obtained from the Air Data sensors. Again, for Wind Estimation purposes only components of the *x-axis* and *y-axis* are to be calculated and considered.
- α and β (for the model *aoa* and *beta*): Obtained from the Air Data sensors.
- φ, θ and ψ (for the model *phi, theta* and *psi*): The Euler Angles are obtained from the angular rates *p, q* and *r* given by the FROG's IMU.

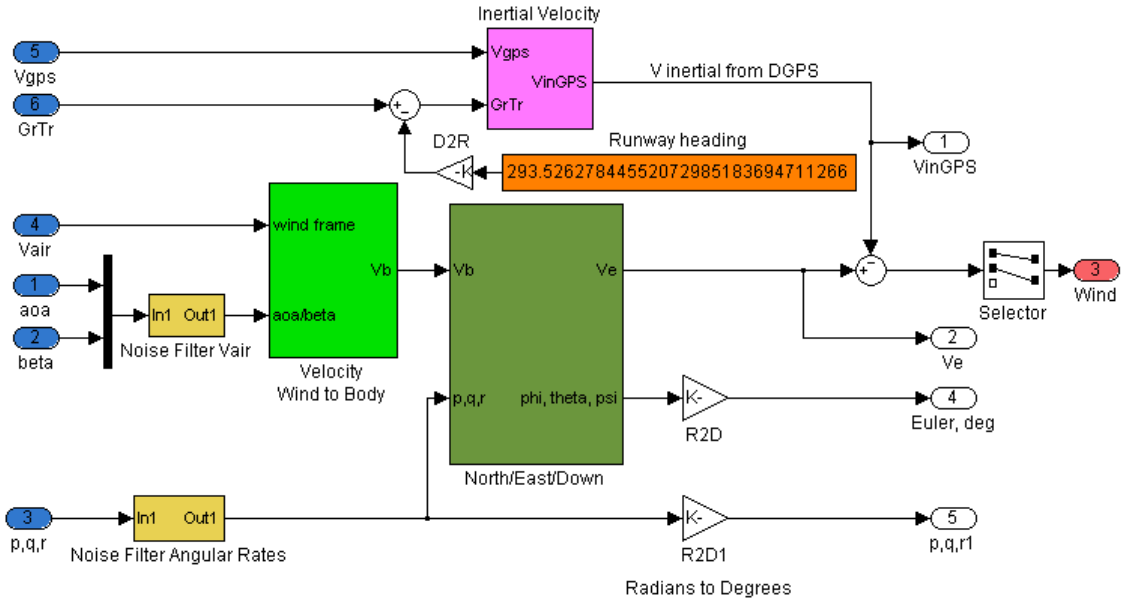


Figure III.2 Wind Estimation Input Visualization and General Layout

Details of the main blocks for the above figure can be found in Appendix B.

In Figure (III.2), the vertical rectangular blocks in the center part of the model (green colors) perform the mathematics related with ${}^{i}_{w}RV_{b,w} = {}^{i}_{b}R\,{}^{b}_{w}RV_{b,w}$ and the square shape block at the top part of the model (magenta color)

solves for ${}^{i}V_{b,i}$. These two last results are subtracted and the Real Wind velocity is obtained.

The selector on the right hand side of the model disregards of the z component for the estimated wind, as only components of x and y axes are going to be used.

When solving for the velocity of the platform in the inertial frame, the DGPS input for the Ground Track G_rT_r must consider an initial condition related with the heading of the FROG at the take-off position. This is accomplished by the subtraction of the constant (293.52627…) that corresponds to the orientation of the runway in degrees given by the DGPS at the take-off position of the FROG.

B. DATA RECEPTION MODULE

The Wind Estimation model uses a RS-232 Mainboard block from the xPC Target library of SIMULINK, to setup the serial port used for receiving incoming data from the Tattletales. As seen on Chapter Two (II.), the arriving data will be presented in three different types of sentence structures, which can be recognized by its headers. Once the data has been received it must be decoded in order to obtain the individual input variables that the model will handle.

In the decoding part, first a header must be found in the incoming serial data and depending on its identification a decoding procedure must be applied to the rest of that particular sentence to obtain variables of interest.

To accomplish the header/data identification and the later data decoding procedures, software interface drivers where written in C Programming and conform to S-function

standards [Ref. 5]. Dr. Vladimir Dobrokhodov (NPS) made this work, which is out of the scope of this thesis. The, codes are included on Appendix B for further reference.

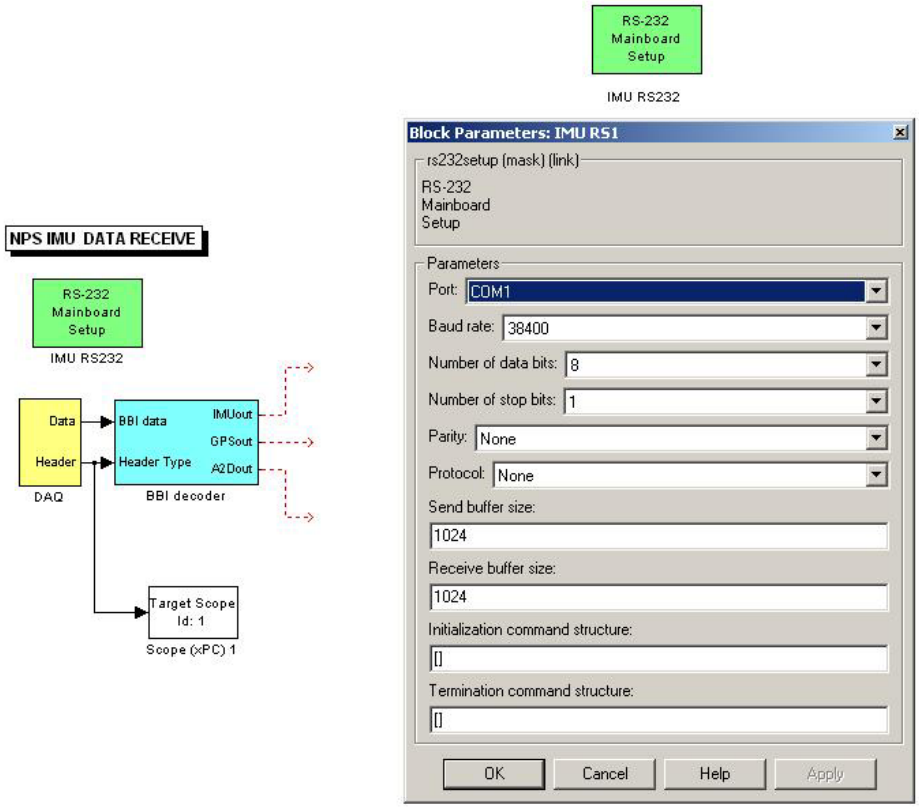


Figure III.3 Data Receive Layout and RS-232 Setup

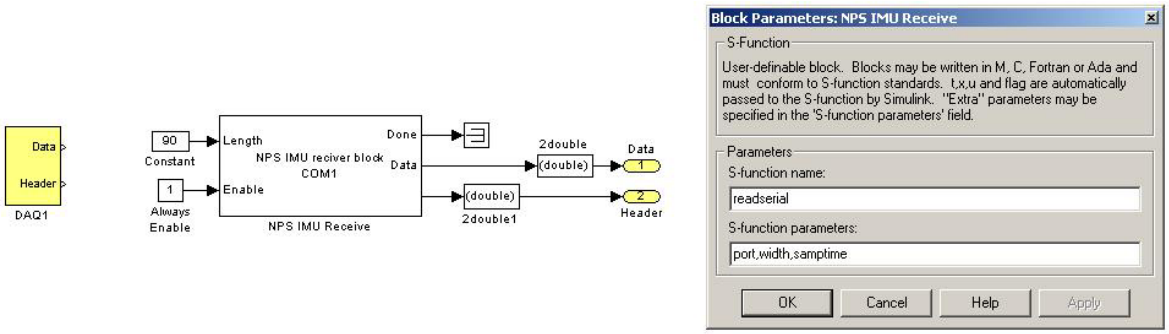


Figure III.4 Data/Header Block Details

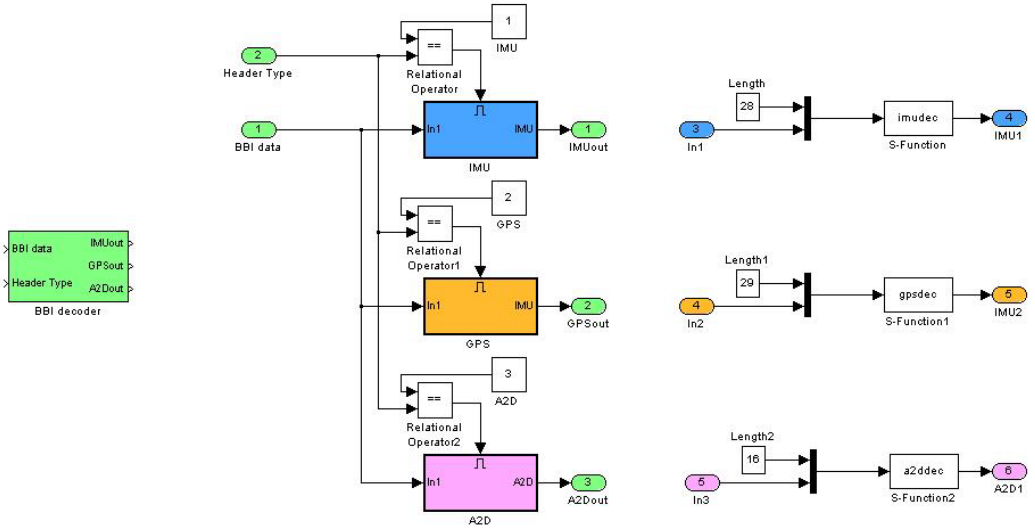


Figure III.5 Data Decoder

From Figures (III.4) and (III.5) the written software interfaces drivers where “readserial”, which generates the output vectors Header and Data; “imudec”, which generates the output vector IMUout; “gpsdec”, which generates the output vector GPSout, and “a2ddec”, which generates the output vector A2Dout (codes for software interfaces drivers are included on Appendix B).

C. DATA CALIBRATION MODULE

The Calibration Module receives three vectors IMUout, GPSout and A2Dout. For each vector, every element contains a numerical representation of data that has been originally measured by a sensor, pre processed by the Tattletales and decoded in the Data Reception Module.

The purpose of the Calibration Module is to perform a correlation between those numerical quantities and their corresponding value in the MKS Units System, so that every measurement performed by a sensor could be used as a variable in the Wind Estimation Model.

The Wind Estimation Model uses only some elements of each vector and they will be explained separately.

1. The IMUout Vector

Only five elements of this vector are used as variables of the Wind Estimation Model: Angle of Attack (aoa), Sideslip Angle (beta) and the three angular rates obtained from the IMU's rate gyros (*p*, *q* and *r*):

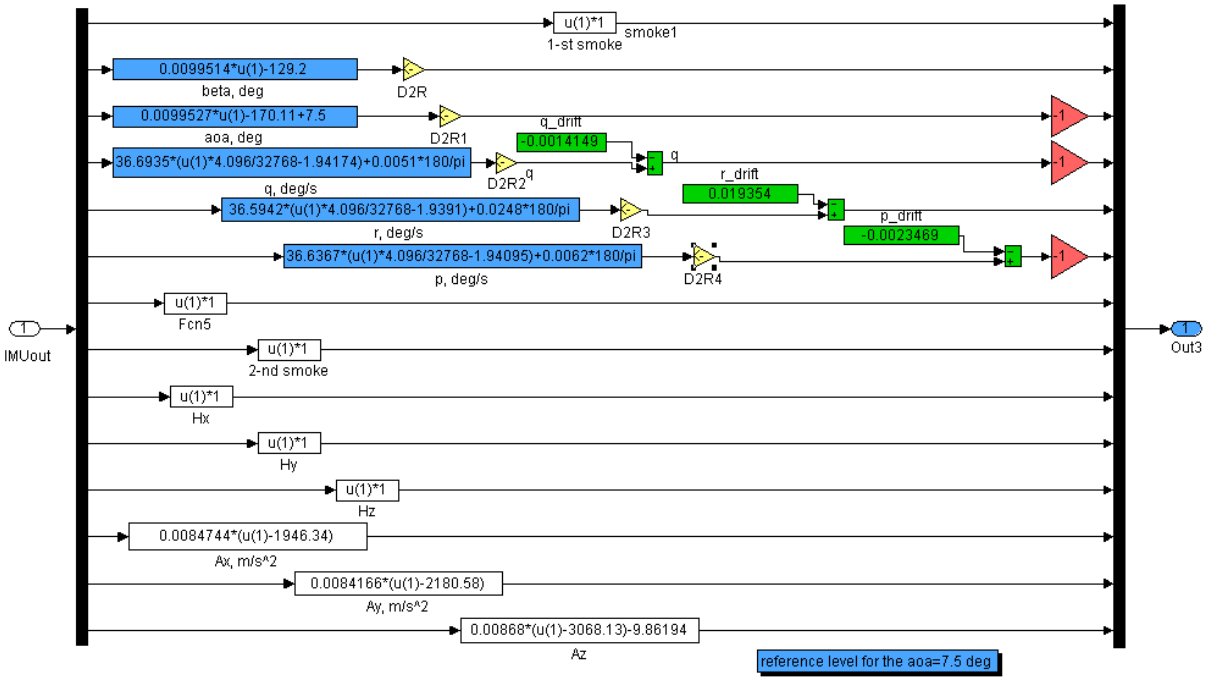


Figure III.6 Calibration Module IMUout Vector

For the five variables shown in Figure (III.6) the incoming numerical values are converted into Degrees (for the angles) and Degrees per Second (for the angular rates). Once these new representations are obtained, Degrees are converted to Radians so MATLAB trigonometric functions can be later used.

In order to obtain the mathematical expressions contained in the rectangular blocks (blue) of Figure (III.6), sensor calibration procedures must be carried out.

This work was performed in Laboratory under controlled environments and consists on obtaining a wide range of numerical representations given by the data acquisition package of the model (i.e. Tattletale processing and decoding procedures) against it corresponding measured value.

The data is then tabulated and the calibration equations are obtained via curve fitting tools.

For Alpha and Beta samples where recorded every 5° over a range of ± 40° and the following plot of the tabulated values was obtained [Ref. 2]:

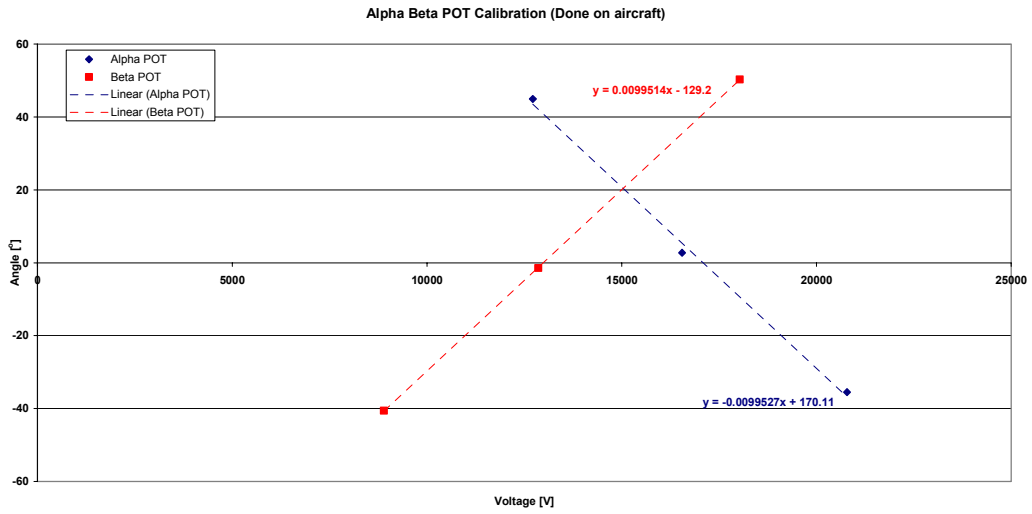


Figure III.7 Alpha and Beta Sensor Calibration Results

Comparing the curve fitting equations shown in Figure (III.7) with the corresponding blocks of Figure (III.6), it can be verified that for Beta the equation was applied directly. For Alpha, the sign difference was corrected by a (–1) multiplication (red triangular block) and a 7.5° correction was included because of an angular difference between the Alpha vane pod and the longitudinal axis of the FROG once the sensor was installed on the wing tip.

To obtain angular rate equations (p, q and r), the FROG's IMU was mounted on a Tilting Rotary Table Model TRT7 built by, HAAS Automation Inc. and available at the Controls Laboratory of the NPS Electrical Engineering Department.

An accurate control system provided by the rotary table, allowed tilting the IMU at different angular rates while numerical values from the rate gyros where obtained by the data acquisition package of the model. Again, all corresponding data was tabulated and equations where found.

From the obtained data, constant drift characteristics where observed on each gyro and those values where corrected as can be seen in Figure (III.6) (green blocks). Furthermore, from the sensor calibration procedures it was detected that the rate gyros for p and q where oriented in the wrong direction. Later this was physically confirmed and was corrected by a (–1) multiplication (red triangular block).

2. The GPSout Vector

Only two elements of this vector are used as variables of the Wind Estimation Model: Ground Speed (Vgps) and Ground Track (GrTr).

Both signals are obtained from the DGPS Trimble Ag132 and do not require any kind of sensor calibration. The sentence format shown in Chapter two (II.) is captured and decoded by the data acquisition package of the model.

The only issue that has to be considered at this point is that the DGPS velocity given in Knots and the track in Degrees. Corrections are made in the red triangular blocks, as velocity must be converted to Meters per Second (MKS) and Degrees to Radians (MATLAB's requirement).

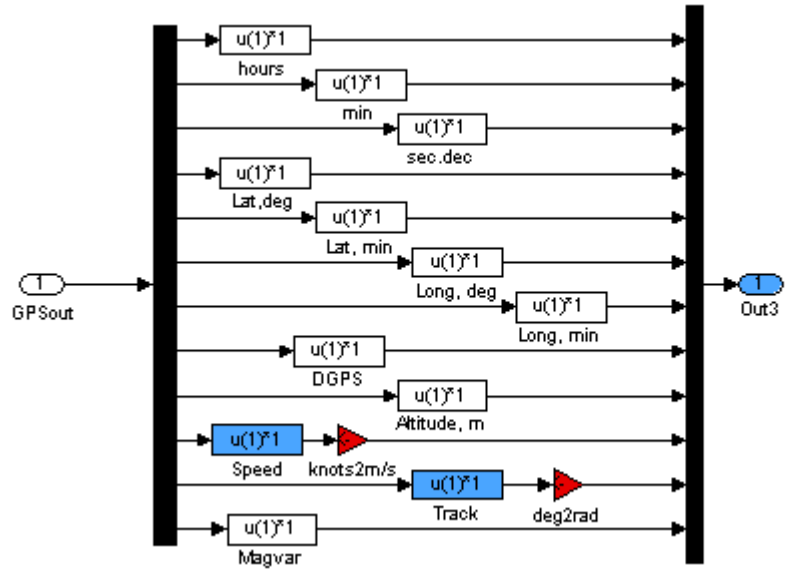


Figure III.8 Calibration Module GPSout Vector

3. The A2Dout Vector

Only one element of this vector is used as an input variable of the Wind Estimation Model: Air Speed (Vair).

A similar sensor calibration procedure to those used for signals in vector IMUout was performed and a quadratic equation was implemented in this module. Details of this sensor calibration can be found in [Ref. 2].

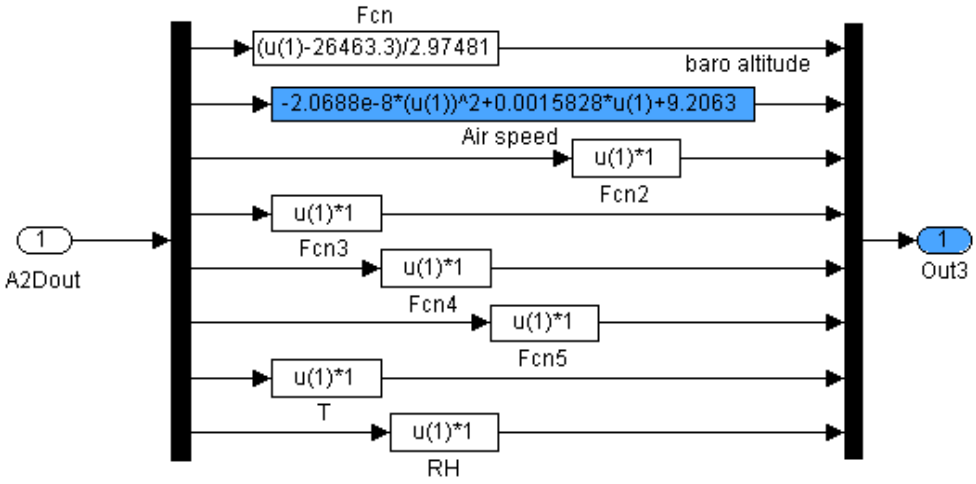


Figure III.9 Calibration Module A2Dout Vector

IV. FLIGHT TEST

The Wind Estimation Model was constructed in a SIMULINK environment using subsystems from the xPC Target library. This characteristic of the model allows using MATLAB's Real-Time Workshop to provide a real-time development environment from where Wind Estimation results are obtained.

To accomplish this, FROG is connected with a rapid prototyping target computer PC-104 (xPC Target). The target computer is linked to the physical sensors and microprocessors (Tattletale8) to carry out data acquisition. Real-Time Workshop transforms the Wind Estimation Model to C code and creates an executable of the model and places it on the target system.

The model is downloaded from a ground station Host PC via the Data Link assembly described in Chapter Two (II.), which also allows the required real-time monitoring of wind estimation results. Figure (IV.1) shows the general layout of this environment:

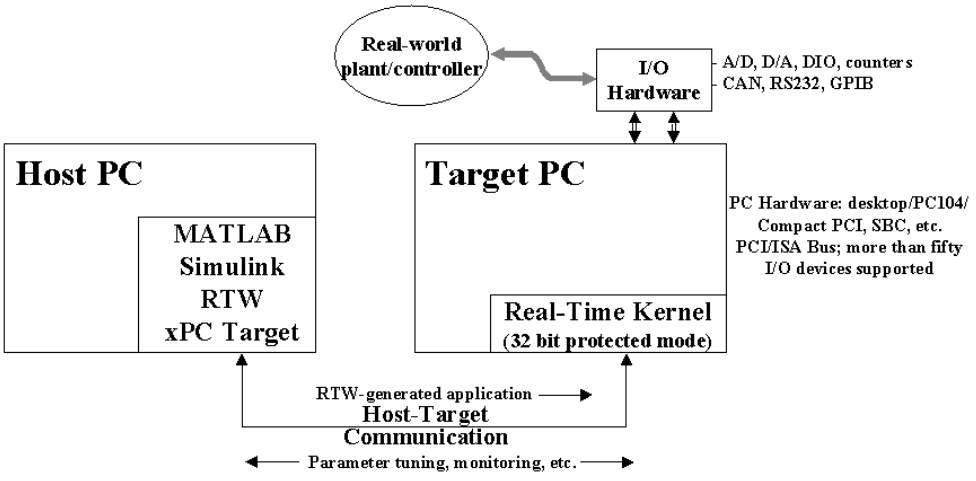


Figure IV.1 xPC Target Environment [From Ref. 6]

A. FLIGHT TEST PROFILE AND GENERAL PROCEDURES

The FROG was flown at the McMillan Airfield at Camp Roberts, California on 9 October 2002. The main objective of the test was to verify the Wind Estimation Model by comparing its wind estimation results against real wind measurements performed by the NPS Meteorology Department. The flight profile for the UAV was straight and level passes on the runway heading with a turn at each end at an approximate flight level of 50 feet AGL. Figure (IV.1) shows the flight profile of the test flight obtained with the GPS unit onboard the FROG and an aerial photo of the airfield. The flight profile is in GPS coordinates (Longitudinal Minutes versus Lateral Minutes).

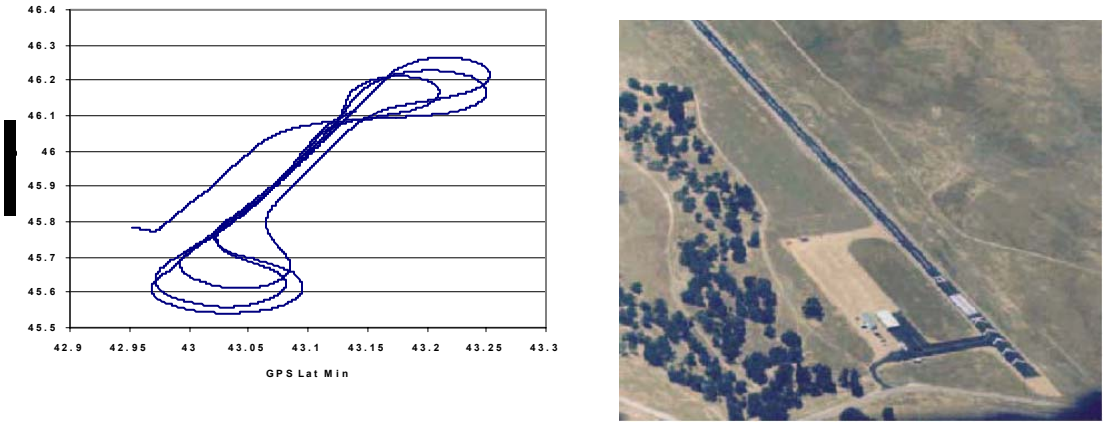


Figure IV.2 Flight profile of FROG during test flight (left) and McMillan Airfield (right)

NPS Meteorology Department performed data collection at the demonstration site. The collection was done with three ground-stations and one Rawinsonde system (balloon), which allowed a 3-D description of the tested air volume.

The ground station systems operated continuously during the entire time collection period. The Rawinsonde

system was used at scheduled times to collect profiles of vector winds, temperature, and humidity.

For the continuous ground-based measurements, portable instrumented meteorological towers were installed on October 2, 2002 in continuous operation mode until removed October 10, 2002. The tower designation and location were:

West Tower: 50 ft South of the NW end of the runway.
East Tower: on a hill several hundred ft North of the midpoint of the runway.
North Tower: 50 ft north of the SE end of the runway.

Figure IV.3 Instrumented Portable Meteorological Tower

The towers were instrumented for true vector wind (speed and direction reference to true North), air pressure, air temperature and humidity. The sensors were sampled at 1 Hz and the output averaged over a two-minute interval.

B. TEST FLIGHT RESULTS

Validation of the Wind Estimation Model was based on a nine-minute flight at 50 ft high over McMillan Airfield, following the flight profile showed on Figure (IV.). The data collected during the flight was used to estimate wind and compare it to measured true wind by the meteorological equipment.

Validation legs were referred to the runway orientation of 290° so data of interest was collected during general DGPS headings of 300° and 120°. The average speed for the FROG for the valid paths was 66 knots.

Next figure presents the results of wind direction and velocity (left side) as well as the data collected by the meteorological towers (right side). This data corresponds to a general heading of the FROG of 280°.

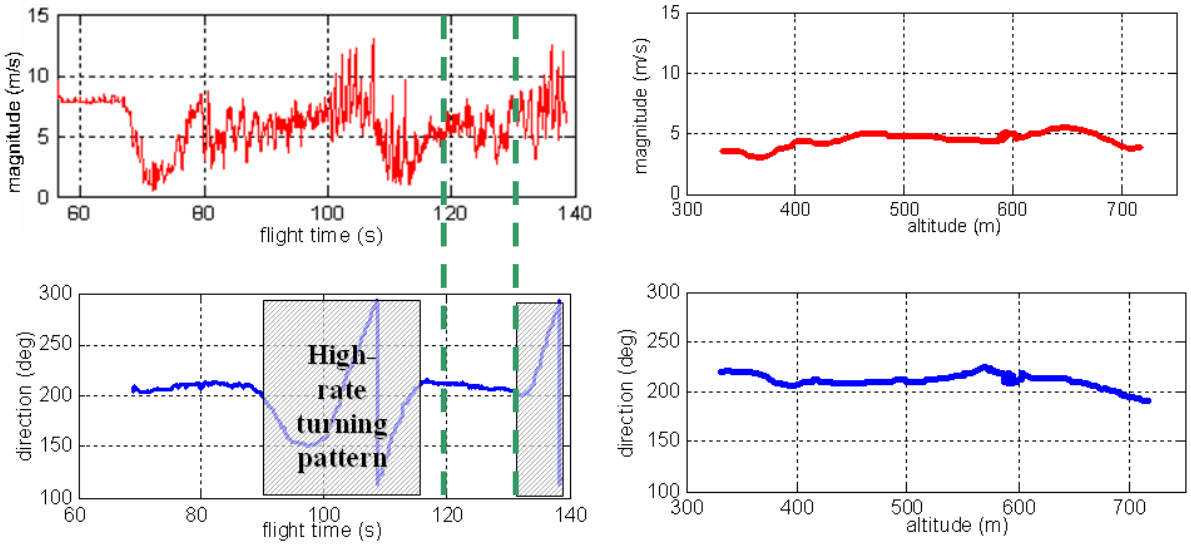


Figure IV.4 Wind Estimation Results Straight Path

Summary of the comparison of the averaged results is presented in a table below.

	Wind Magnitude	Wind Direction
Real-Time Model Results	6.5 m/s	210°
Meteorological Measurements	4.9 m/s	214°
Error	1 - 3 m/s	1°-5°

Table IV.1 Wind Results Comparison

From the above table we have:

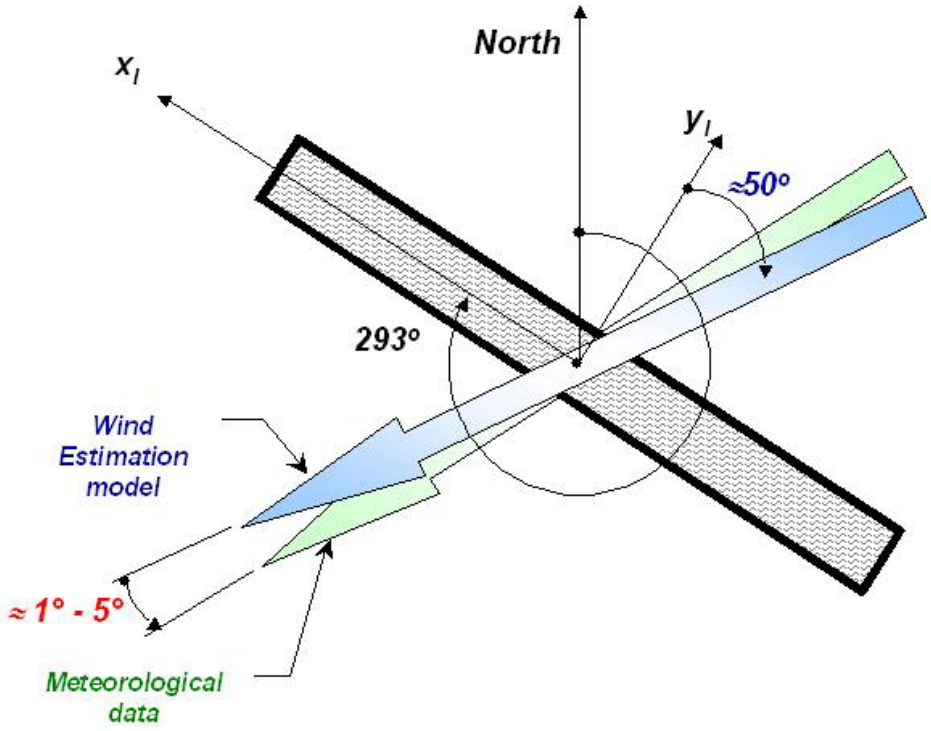


Figure IV.5 Wind Direction Results

Clearly, the wind velocity and direction results match those obtained by the meteorological towers fairly well. This comparison indicates that Wind Estimation based on the UAV sensors was successful.

C. REAL-TIME PRESENTATION FOR WIND ESTIMATION

The real-time presentation of the results was done using SIMULINK/Real-Time Workshop available features.

A simple SIMULINK model as the one shown in Figure (IV.6) can be run on the Host PC. The "From xPC Target" blocks available in the xPC Library capture real-time output variables from the Wind Estimation Model. This is possible since both computers are connected via the Data Link assembly.

Once these variables are available in the SIMULINK model, they can be mathematically related and results can be displayed using blocks from the Dials and Gages Library. This can be seen in Figure (IV.6):

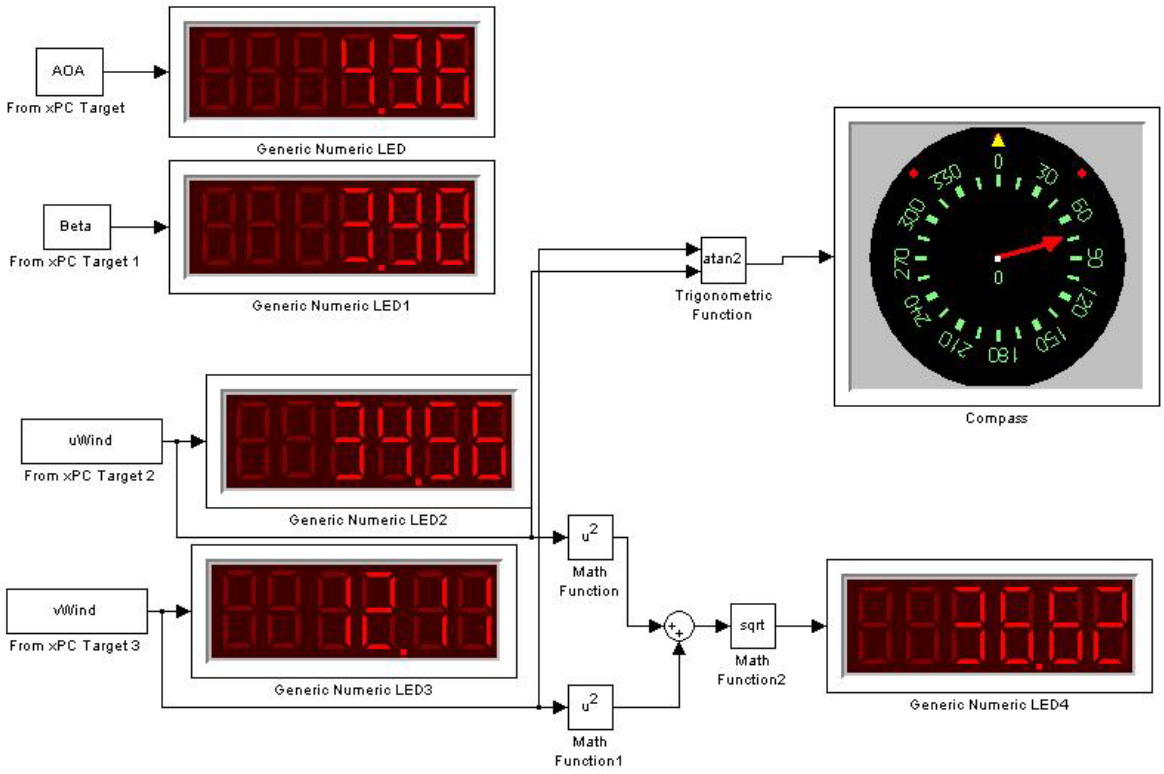


Figure IV.6 Real-Time Presentation Example

V. CONCLUSIONS AND RECOMENDATIONS

A. CONCLUSIONS

The primary goal of this thesis was to obtain real-time Wind Estimation based on data collected by embedded sensors of a UAV platform and to validate those results by a comparison to wind measurements obtained by calibrated meteorological equipment.

Highly accurate meteorological and navigation information has been obtained during the flight tests that prove the efficiency of the UAV employment. Developed hardware architecture has confirmed the idea of a real-time data acquisition airborne unit for the task of meteorological prediction.

Currently employed hardware components provide a state of the art in portability of UAV system deployment. Created real time software has shown its compatibility with real-time processing requirements, adequate accuracy and robustness.

Analyzed results have revealed a significant potential and promising direction in UAV based system that should be further addressed.

B. RECOMMENDATIONS

Future work would include an improvement of hardware design that allows more flexibility in hardware rigging. It should support an exchangeable utilization of more precise and numerous heterogeneous sensors including a “full” variety of possible chemical/biological agents detectors.

Software enhancement should address two principal issues that allow moving the project onto direction of

increased autonomy. The first topic includes an implementation of complimentary filtering technique to provide better resolution of the heterogeneous information from variety of possible sensors. The other issue should address the development and implementation of pilot support tools to extend the operational area and simplify the navigation task. It can be achieved by the development and implementation of such trajectory pattern (grid) where UAV is autonomously guided and also by employing a modern GPS based technique through the real-time visualization of navigational data.

APPENDIX A. DESCRIPTION OF THE FOG-R UAV (NPS FROG)

BAI TERN

Type
Multipurpose, semi-expendable UAV.

Development
The TERN (Tactical Expendable Remote Navigator) was originally designed and developed by H-Cubed Corporation of Columbia, Maryland. The programme was acquired by BAI Aerosystems in 1993. Capable of a variety of applications, it is recoverable in peacetime training missions but of sufficiently low cost that it could be discarded after a battlefield mission if necessary. On 1 October 1992, a TERN set up an endurance record in FAI Class F3a for unmanned aircraft of 33 hours 39 minutes 15 seconds. A straight-line distance record in the same class was set on 28 September 1993 with a flight of 245.80 n miles (455.23 km; 282.87 miles).

Airframe
Pod-and-boom fuselage with high-mounted wing, sweptback fin and rudder, and low-set tailplane; wing fitted with flaps; GFRP/epoxy construction. Engine mounted above wing centre-section; fixed tricycle landing gear.

Mission payloads
Can include colour TV camera, thermal imager or hazardous agent sensors. One variant **(TERN-C)** has carried a remote IR sensor for aerial detection of chemical warfare agents. Another **(FOG-R)** has been fitted with an Optelecom Fibre Optic DataLink (FODL) and flown non-line of sight 'over and below the hill' for uplink command of the UAV and downlink video imagery. The latter version is stated to be proof against jamming.

Guidance and control
UHF (400 MHz) radio command uplink for remote control; D-band (1.8 GHz) video downlink with data sideband. Options include programmable autopilot, GPS navigation and fibre optic datalink. Wing flaps can be trimmed to enable slow flight speeds for surveillance, or fully extended to assist landing in restricted spaces.

Transportation
Wings detach for containerised storage and transportation.

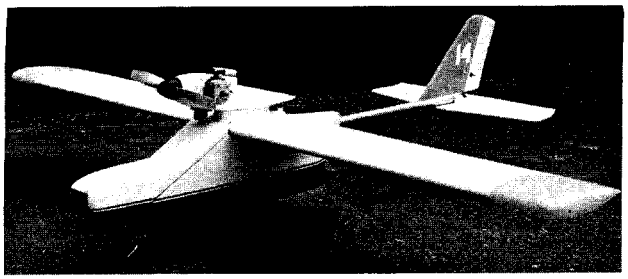
Launch
Conventional wheeled take-off.

Recovery
Conventional wheeled landing.

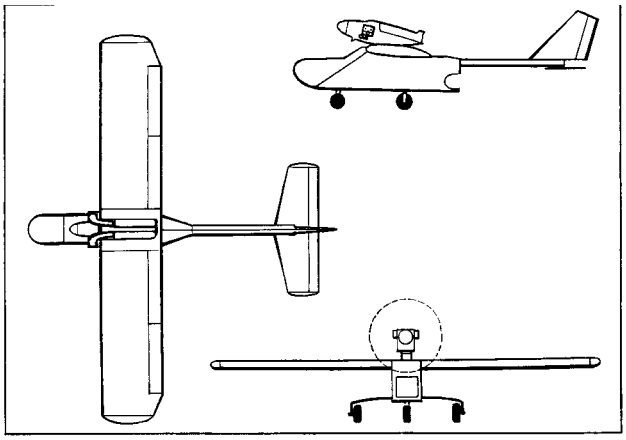
Operational status
In production.

Customers
Has been used by US Army as surrogate fibre optic guided missile (FOG-M) and as an NBC sensor vehicle. Also used by Naval Weapons Center and ARDEC.

Prime contractor
BAI Aerosystems Inc, Easton, Maryland.



BAI TERN semi-expendable multirole UAV 1998/0001584



TERN air vehicle three-view (Jane's/John W Wood) 1998

Power plant
One 7.5 kW (10 hp) 150 cc two-cylinder two-stroke engine (type not known); two-blade fixed-pitch wooden propeller.

Dimensions	
Wing span	3.10 m (10 ft 2.0 in)
Wing area	1.57 m² (16.94 sq ft)
Length overall	2.48 m (8 ft 1.5 in)
Height overall	0.86 m (2 ft 10.0 in)
Wheel track	0.74 m (2 ft 5.0 in)
Weights	
Weight empty	17.7 kg (39.0 lb)
Max payload	13.6 kg (30.0 lb)
Max T-O weight	34.0 kg (75.0 lb)
Performance	
Max level speed	70 kt (129 km/h; 80 mph)
Normal cruising speed	48-52 kt (88-96 km/h; 55-60 mph)
Loiter speed	35-43 kt (64-80 km/h; 40-50 mph)
Stalling speed	31 kt (57 km/h; 35 mph)
* Range	8.6 n miles (16 km; 10 miles)
Typical endurance at above cruising speed	3 h

* *Extendable in autonomous flight mode*

Figure A. 1 FROG Main Characteristics

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. WIND ESTIMATION MODULE DESCRIPTION

The objective of this appendix is to provide a detailed description of the Wind Estimation Module used in the Wind Estimation Model. Mathematics in this appendix refer to Chapter Two (II.)

On Figure (B.1), the blue blocks correspond to SIMULINK library Variable Selectors that provide the necessary inputs to the module:

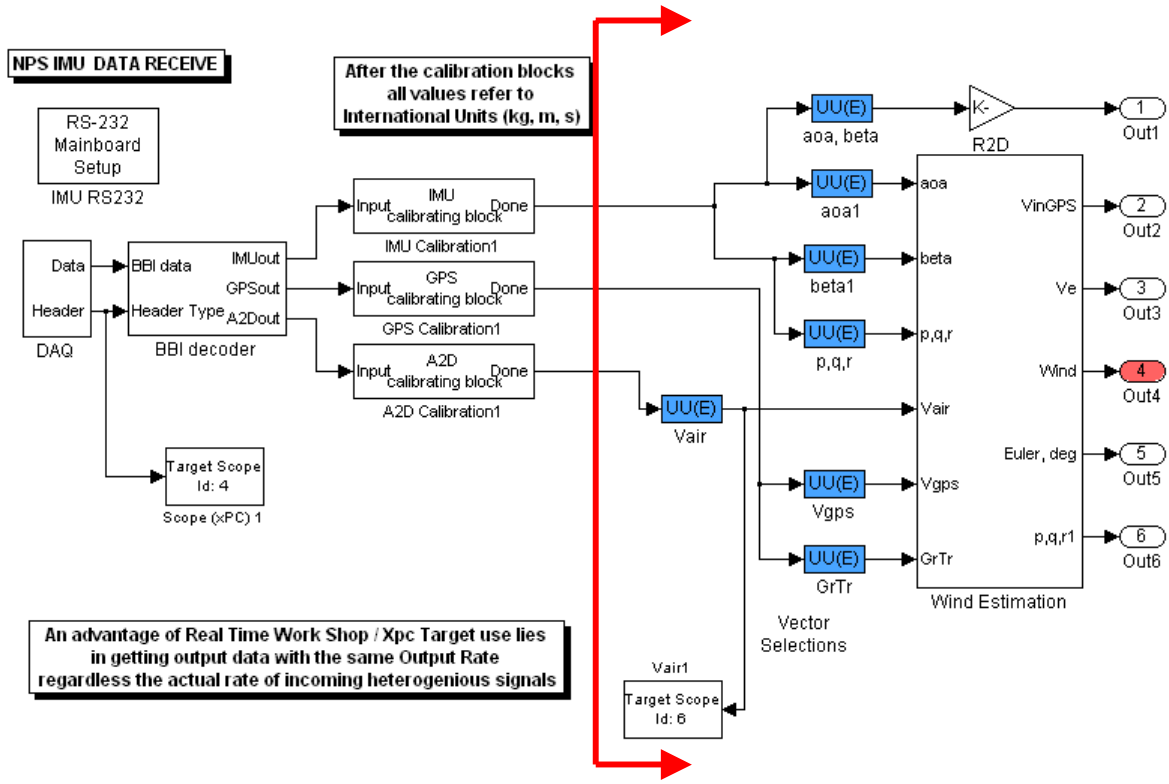


Figure B.1 Wind Estimation Module General Layout

The Wind Estimation Block (white on Figure B.1) mainly contains:

Inertial Velocity Block (magenta).

Velocity Wind to Body Block (light green).

North/East Down Block (dark green).

Recall Figure (III.2):

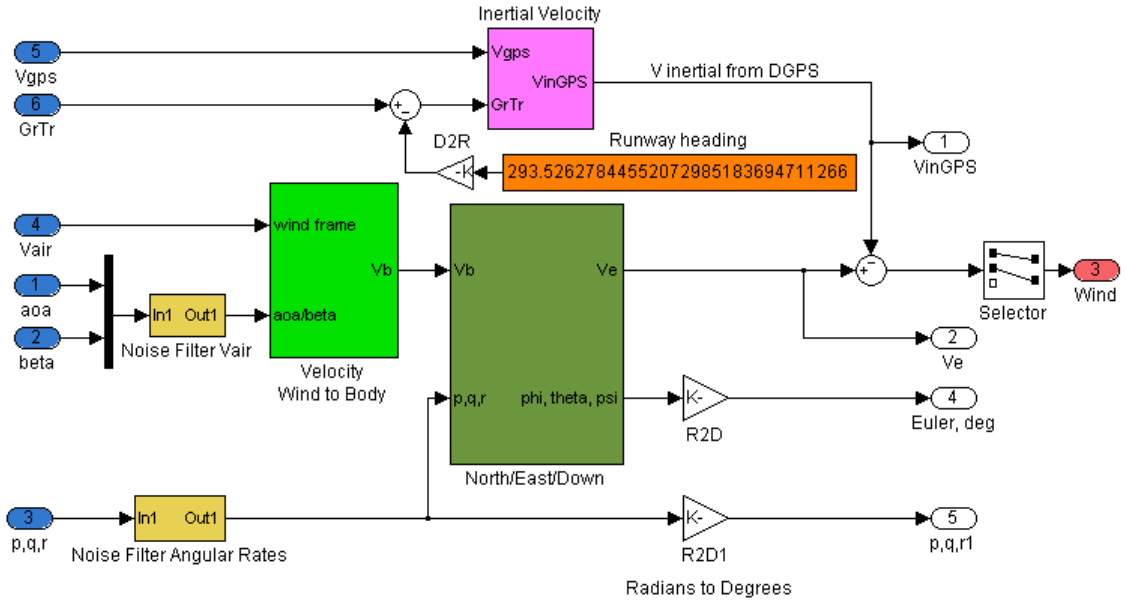


Figure B. 2 From Chapter Three Figure (III.2)

For the Inertial Velocity block we have:

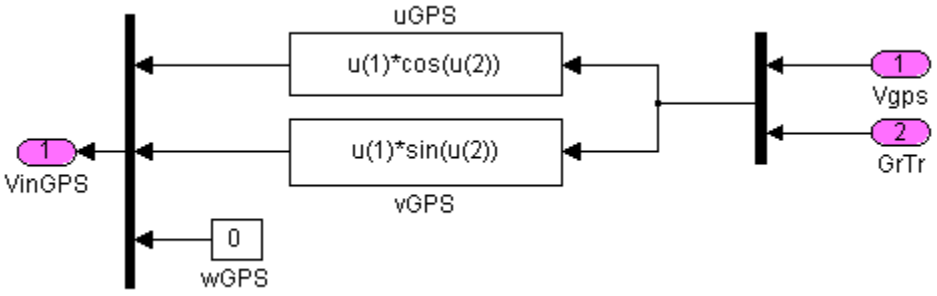


Figure B.3 Inertial Velocity Block.

Note that the *w* component of the DGPS velocity is cancelled.

For the Velocity Wind to Body Transformation block we have:

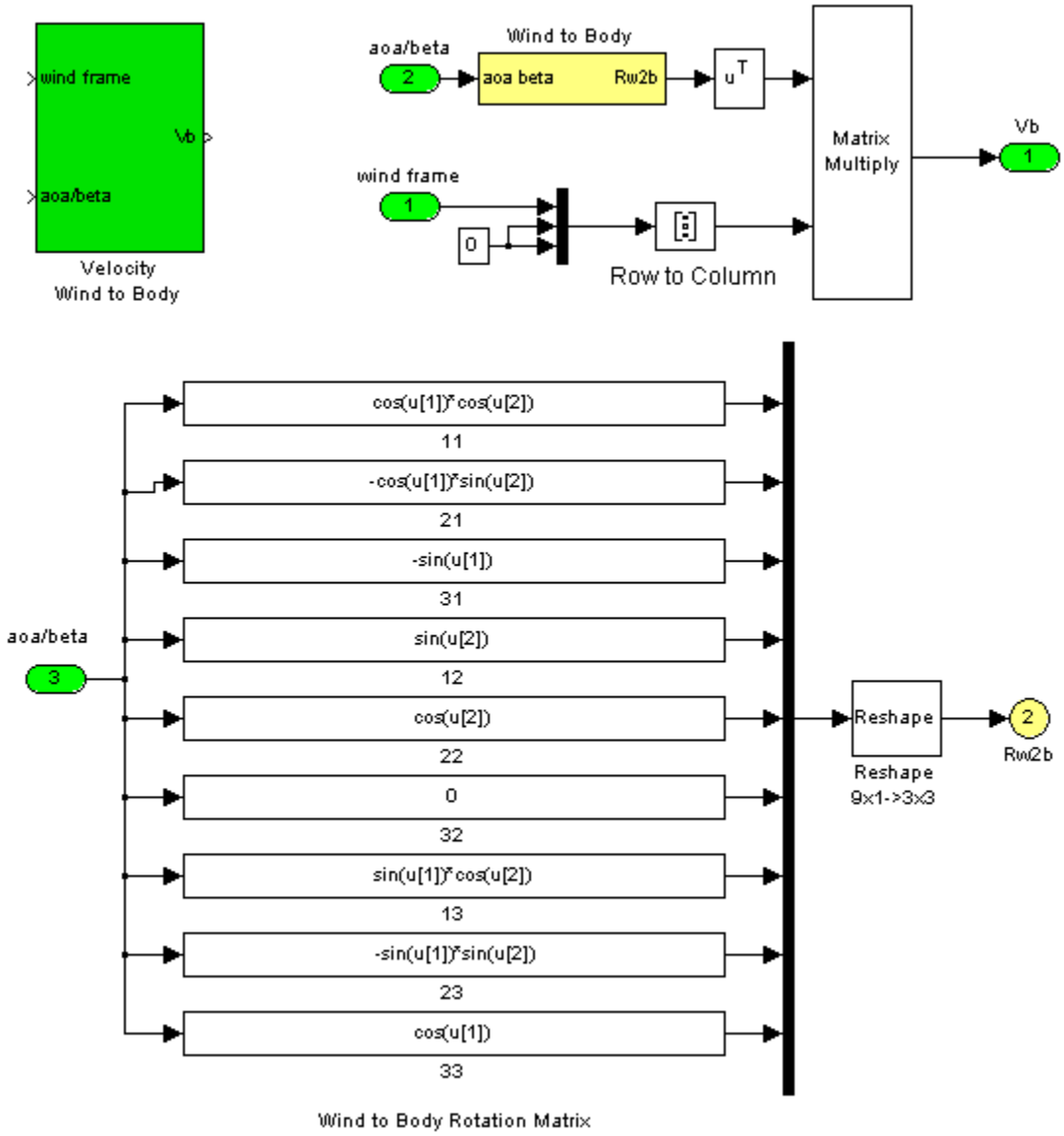


Figure B.4 Wind to Body Transformation

For the Wind Frame input only the *x* component is considered. This represents the impact wind over the pitot tube of the UAV (longitudinal axis).

For the North/East/Down block we have:

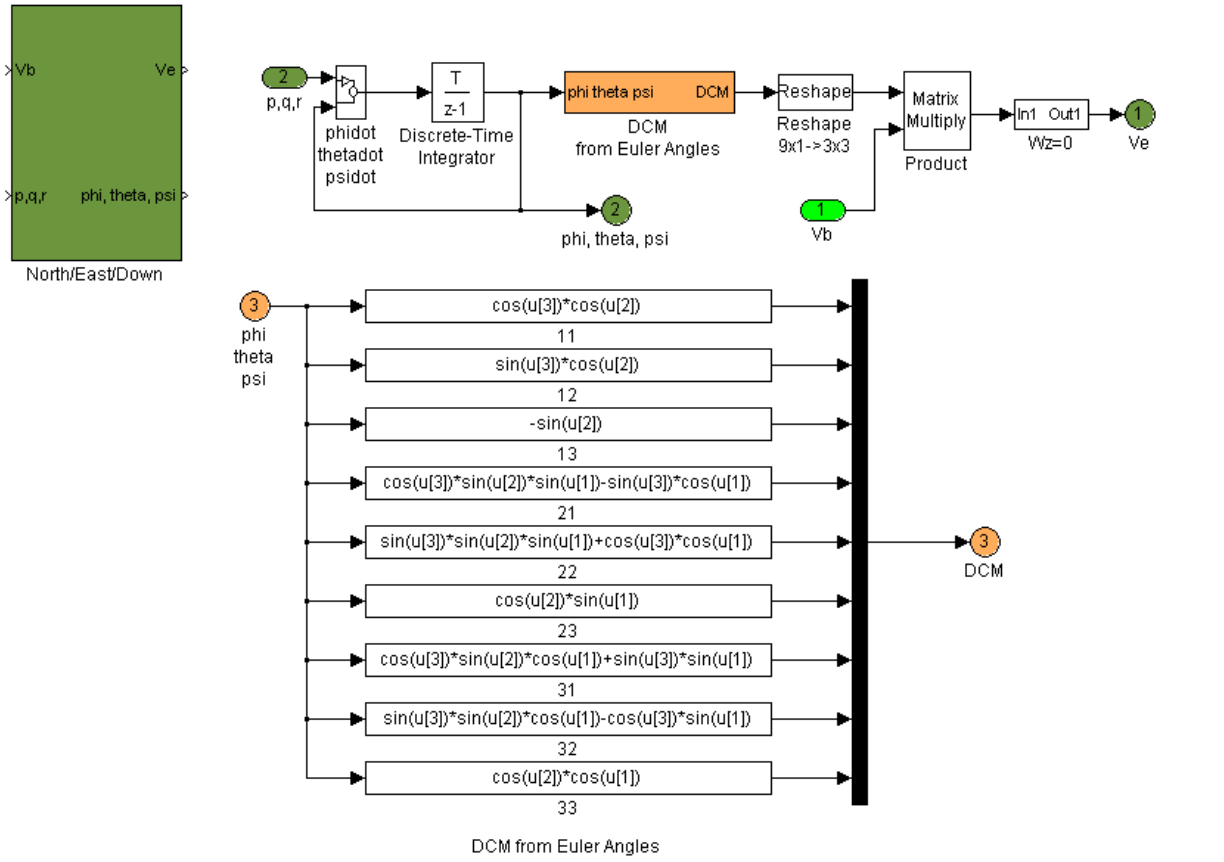


Figure B.5 Body to Inertial Transformation

And for the “phidot thetadot psidot” block we have:

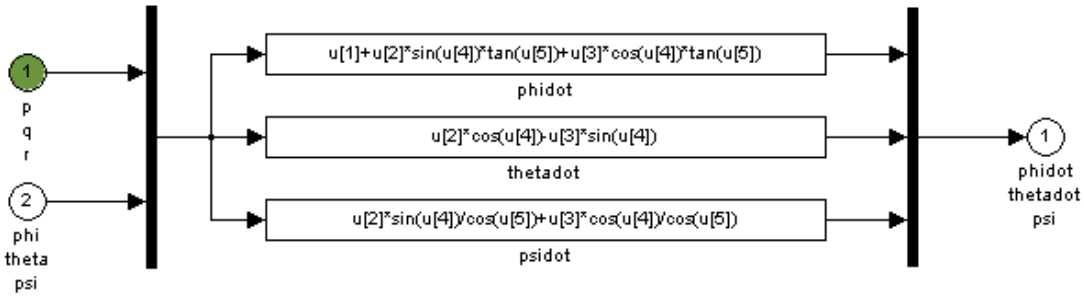


Figure B. 6 Phidot Thetadot Ppsidot Block

APPENDIX C. SOFTWARE DRIVERS

This appendix contains the C Programming source code written to receive and decode the data streams from the FROG's IMU. Dr. Vladimir Dobrokhodov wrote these drivers. All the C-codes had to be packaged into MATLAB's S-Function Level 2 structure which adopts a specific sequence to initialize a simulation block, update its states, control sampling rates, output data and terminate the function. Each set of code has to be "MEX" by a compatible C-compiler in MATLAB and 'build' into executable code by xPC's Real-Time Workshop before it can be called from within a SIMULINK block as a S-Function. Details of how this is done are discussed in [Refs. 5 and 6].

1. SERIAL DATA RECEIVE DRIVER

readserial.c

```
/* $Revision: 1.1 $ $Date: 2001/07/20 22:11:41 $ */
/* rs232rec.c - xPC Target, non-inlined S-function driver for RS-232 */
/* receive (asynchronous) */
/* Copyright 1996-2001 The MathWorks, Inc. */

#define S_FUNCTION_LEVEL 2
#undef  S_FUNCTION_NAME
#define S_FUNCTION_NAME readserial

#include <stddef.h>
#include <stdlib.h>

#include "tmwtypes.h"
#include "simstruc.h"

#ifdef MATLAB_MEX_FILE
#include "mex.h"
#else
#include <windows.h>
#include <string.h>
#include "rs232_xpcimport.h"
#include "time_xpcimport.h"
#endif
```

```
/* Input Arguments */

#define NUMBER_OF_ARGS  (3)                  /*Number of parameters in
                                               a block parameters
                                               dialog */
#define PORT_ARG        ssGetSFcnParam(S,0)  /*Number of Serial Port*/
#define WIDTH_ARG       ssGetSFcnParam(S,1)  /*Maximum width of
                                               INCOMING sentence per
                                               packet*/
#define SAMP_TIME_ARG   ssGetSFcnParam(S,2)  /*Sample time*/

#define NO_I_WORKS      (3)                  /*Current pos pointer in
                                               buf, rec length,
                                               bufCount*/
#define NO_R_WORKS      (0)
#define NO_P_WORKS      (0)
#define NO_D_WORKS      (1)                  /*for buf array*/

#define HEADER          (36)                 /*$- sign*/

static char_T msg[256];
extern int rs232ports[];
unsigned char remains;  /*global array to save remains after
                          subtracting procedure*/

static void mdlInitializeSizes(SimStruct *S)
{
#ifndef MATLAB_MEX_FILE
#include "rs232_xpcimport.c"
#include "time_xpcimport.c"
#endif

    ssSetNumSFcnParams(S, NUMBER_OF_ARGS);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        sprintf(msg,"Wrong number of input arguments passed.\n"
                "%d arguments are expected\n",NUMBER_OF_ARGS);
        ssSetErrorStatus(S,msg);
        return;
    }

    /* Set-up size information */

    ssSetNumContStates( S, 0);
    ssSetNumDiscStates( S, 0);
    ssSetNumOutputPorts(S, 3);     /*func-call,data, header index*/
    ssSetNumInputPorts( S, 2);     /*rec length, enable*/

    ssSetOutputPortWidth(S, 0, 1); /*Function-call*/

    ssSetOutputPortWidth(S, 1, (int)mxGetPr(WIDTH_ARG)[0]); /*Data*/
    ssSetOutputPortDataType(S, 1, SS_UINT8);

    ssSetOutputPortWidth(S, 2, 1);              /*Header index*/
    ssSetOutputPortDataType(S, 2, SS_UINT8);

    ssSetInputPortDirectFeedThrough(S, 0, 1);
    ssSetInputPortDirectFeedThrough(S, 1, 1);
```

```
    ssSetInputPortWidth(S, 0, 1);
    ssSetInputPortWidth(S, 1, 1);

    ssSetInputPortRequiredContiguous(S, 0, 1);
    ssSetInputPortRequiredContiguous(S, 1, 1);

    ssSetNumSampleTimes(S,1);
    ssSetNumIWork(S, NO_I_WORKS);
    ssSetNumRWork(S, NO_R_WORKS);
    ssSetNumPWork(S, NO_P_WORKS);
    ssSetNumDWork(S, NO_D_WORKS);

    ssSetDWorkDataType(S, 0, SS_UINT8);
    ssSetDWorkWidth(S, 0, (int)mxGetPr(WIDTH_ARG)[0]);
    ssSetDWorkWidth(S, 0, 2048);

    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs( S, 0);

/* This S-function's parameters cannot be changed in the middle of */
/* a simulation, hence set them to be non-tunable. */

     {
    int_T ntune;
        for (ntune=0; ntune < NUMBER_OF_ARGS; ntune++) {
            ssSetSFcnParamNotTunable(S, ntune);
        }
    }
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE |
                 SS_OPTION_PLACE_ASAP);
}
/* Function to initialize sample times */

static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, mxGetPr(SAMP_TIME_ARG)[0]);
    if (mxGetN((SAMP_TIME_ARG))==1) {
        ssSetOffsetTime(S, 0, 0.0);
    } else {
        ssSetOffsetTime(S, 0, mxGetPr(SAMP_TIME_ARG)[1]);
    }
    ssSetCallSystemOutput(S, 0);
}

/* Function: mdlStart                                              */
/* =============================================================== */
/* Abstract:                                                       */
/* At the start of simulation in RTW, print a message to the MATLAB */
/* command window indicating that something is going on.           */

#define MDL_START  /*Change to #undef to remove function*/
#if defined(MDL_START)
static void mdlStart(SimStruct *S)
{
#ifndef MATLAB_MEX_FILE

    ssGetIWork(S)[0] = 0;     /* set current buf pointer = 0 */
```

```
    ssGetIWork(S)[2] = 0;      /* set bufCount = 0 */

    printf("\n readserial.c: This is a beginning of data logging
           procedure");

#endif
}
#endif


/* Function to compute outputs */

static void mdlOutputs(SimStruct *S, int_T tid)
{
#ifndef MATLAB_MEX_FILE

int width = (int)mxGetPr(WIDTH_ARG)[0]; /*specify output port width
                                          =WIDTH_ARG that is the max
                                          length of GPS sentence*/
int port = (int)mxGetPr(PORT_ARG )[0] - 1; /*specify COM#*/
unsigned char tmp;                          /*temp char holder*/
unsigned char *buf = (unsigned char *)ssGetDWork(S, 0);/*uchar buffer
                                                         to contain bytes
                                                         from serial port*/
int *current = ssGetIWork(S);       /*current = addr of current position
                                     pointer in buffer*/
int *recLength = ssGetIWork(S) + 1;  /*recLength = addr of received
                                      data length*/
int *bufCount = ssGetIWork(S)+ 2;    /*count number of useful bytes in
                                      buffer*/

int  serbufCount; /*count number of useful bytes collected in Serial
                    buffer*/
int  i, j,checksum;
int *bl_header;   /*boolean values for IMU=1, GPS=2 and A2D=3
                    sentences, 0=nothing found*/
int  headwidth=2; /*length of header except*/
int  imulng=28,gpslng=31,a2dlng=18; /*initialize length of
                                      IMU(28),GPS(29+2{CR,LF}) and
                                      A2D(16+2{CR,LF}) sentences without
                                      header*/

    if (ssGetInputPortRealSignal(S, 1)[0] == 0) /*If there is no bytes
                                 available so function is disabled. Stop
                                 processing and get out */
       return;

    serbufCount = rl32eReceiveBufferCount(port); /*Check number of
                                                   bytes available*/

    while (serbufCount) {  /*transfer everything from serial buffer to
                             buffer*/
          tmp = rl32eReceiveChar(port);
```

```
        if ((tmp & 0xff00) != 0) {      /*only last 8 bits can be
                                         non-zero*/

            printf("RS232Receive Error: char & 0xff00 != 0 \n");

            return;
        }
        buf[(*current)++] = tmp & 0xff;   /*put valid char into buffer
                                       & means if both operand equal
                                       to 1 the output assigns to 1*/
        serbufCount--;           /*reduce serbufCount*/
        (*bufCount)++;           /*increase bufCount correspondingly*/
    }

    if (*bufCount < width) return;   /*Not enough bytes to decode,
                                      output old value*/

/* Initialize logical flags* /

    i = 0;

    while(((buf[i]==221 && buf[i+1]==221) ||
           (buf[i]==238 && buf[i+1]==238) ||
           (buf[i]==255 && buf[i+1]==255)) == 0 && I < (*bufCount-1))

         { i=i+1; }     /*end of while. Just looking for any first
                         header*/

/* Begin to substruct corresponding arrays if we have enough bytes */

    while (i+max(max(imulng,gpslng), a2dlng) <= (*bufCount))  {

        checksum = buf[i]*256 + buf[i+1];
        i = i + 2;

        if (checksum==65535 & i+imulng <= (*bufCount))  {
             /*IMU Header FF FF -255 255 take imulng bytes and
              send it out*/
            *bl_header=1; /*IMU found*/
            memcpy(ssGetOutputPortSignal(S,1),buf+i,imulng);
            i = i + imulng;  /*next byte to process*/
        }   /*if checksum matches*/

        if (checksum == 61166 & i+gpslng <= (*bufCount))  {
             /*GPS Header EE EE -238 238 take imulng bytes and
              send it out*/
            *bl_header=2; /*GPS found*/
            memcpy(ssGetOutputPortSignal(S,1),buf+i,gpslng);
            i = i + gpslng;  /*next byte to process*/
        }   /*if checksum matches*/

        if (checksum == 56797 & i+a2dlng <= (*bufCount))  {
             /*A2D Header DD DD -221 221 take imulng bytes and
              send it out*/
            *bl_header=3; /*A2D found*/
            memcpy(ssGetOutputPortSignal(S,1),buf+i,a2dlng);
            i = i + a2dlng;  /*next byte to process*/
```

```
          }   /*if checksum matches*/
              /*end of subtracting while*/

/* Send Header index to know how much bytes has to be decoded in each
   message */

          memcpy(ssGetOutputPortSignal(S,2),bl_header,1);
    }  /*end of while*/

/* Substitute remain bytes from this step at the beginning of "buf"
   save remain bytes into the "buf" and shift current to the end of
   a new buffer */

    if (i<(*bufCount))  {

       *bufCount=*bufCount-i; //number of remain bytes

          for (j=0;j<*bufCount; j++) {buf[j]=buf[i];i++;} /*end of for*/

       *current=*bufCount;
    }

    ssCallSystemWithTid(S, 0, 0);   /*issue done pulse to outport 0*/

    return;

#endif
}

/* Function to perform housekeeping at execution termination */

static void mdlTerminate(SimStruct *S)
{
}

#ifdef  MATLAB_MEX_FILE  /*Is this file being compiled as a MEX-file?*/
#include "simulink.c"    /*MEX-file interface mechanism*/
#else
#include "cg_sfun.h"     /*Code generation registration function*/
#endif
```

2. ANALOG TO DIGITAL DATA RECEIVE DRIVER

a2ddec.c

```
/* File : a2ddec.c
   $Revision: 1.00 $V.Dobrokhodov */

#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <iostream.h>
#include <string.h>

#define S_FUNCTION_NAME  a2ddec
```

```
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"

/* Input Arguments */

#define NUMBER_OF_ARGS (1)
#define WIDTH ssGetSFcnParam(S,0) /*WIDTH is the max. length of
                                    incoming IMU sentence*/

/* Build checking */

static char_T msg[256];

/* Function: mdlInitializeSizes
   Abstract:
   Setup sizes of the various vectors. */

static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S, NUMBER_OF_ARGS);

    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        sprintf(msg,"Wrong number of input arguments passed.\n"
                "%d arguments are expected\n",NUMBER_OF_ARGS);
        ssSetErrorStatus(S,msg);
        return; /*Parameter mismatch will be reported by Simulink*/
    }

    if (!ssSetNumInputPorts(S, 1)) return;

    ssSetInputPortWidth(S, 0, int)mxGetPr(WIDTH)[0]);/*DYNAMICALLY_SIZED
    ssSetInputPortDirectFeedThrough(S, 0, 1);

    if (!ssSetNumOutputPorts(S,1)) return;

    ssSetOutputPortWidth(S, 0, 8);  /*A2D length=8 DYNAMICALLY_SIZED*/
    ssSetNumSampleTimes(S, 1);

/* Take care when specifying exception free code. See sfuntmpl_doc.c */

    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE |
                 SS_OPTION_USE_TLC_WITH_ACCELERATOR);
}

/* Function: mdlInitializeSampleTimes
   Abstract:
   Specifiy that we inherit our sample time from the driving block */

static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}
```

```
/* Function: mdlOutputs */

static void mdlOutputs(SimStruct *S, int_T tid)
{
    int_T i=0,j=0;
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    /*Incoming data stream*/
    real_T *y = ssGetOutputPortRealSignal(S,0);
    int_T temp[100]; /*make an aliase for the uPtrs*/
    char ext[]="\0";
    int count=0,len_in;

    for (i=0; i<(*uPtrs[0]); i++)  {temp[i] = ( int_T)(*uPtrs[i+1]);}

    for (i=0; i<(*uPtrs[0]))  {
         *y++=(real_T)(temp[i+0]*256 + temp[i+1]);
         i=i+2;
    }        /*from GPSAtoD0 to GPSAtoD7*/
}

/* Function: mdlTerminate
   Abstract:
   No termination needed, but we are required to have this routine. */

static void mdlTerminate(SimStruct *S)
{
}

#ifdef  MATLAB_MEX_FILE  /*Is this file being compiled as a MEX-file?*/
#include "simulink.c"    /*MEX-file interface mechanism*/
#else
#include "cg_sfun.h"     /*Code generation registration function*/
#endif
```

3. GPS DATA RECIEVER DRIVER

gpsdec.c

```
/* File : gpsdec.c
   $Revision: 1.00 $V.Dobrokhodov */

#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <iostream.h>
#include <string.h>

#define S_FUNCTION_NAME  gpsdec
#define S_FUNCTION_LEVEL 2

#include "simstruc.h"

/* Input Arguments */
```

```
#define NUMBER_OF_ARGS (1)
#define WIDTH ssGetSFcnParam(S,0) /*WIDTH is the max length of incoming
                                    IMU sentence */
/* Build checking */

static char_T msg[256];

/* Function: mdlInitializeSizes
   Abstract:
   Setup sizes of the various vectors. */

static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S, NUMBER_OF_ARGS);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S))  {
        sprintf(msg,"Wrong number of input arguments passed.\n"
                    "%d arguments are expected\n",NUMBER_OF_ARGS);
        ssSetErrorStatus(S,msg);
        return;      /*Parameter mismatch will be reported by Simulink*/
    }

    if (!ssSetNumInputPorts(S, 1)) return;

    ssSetInputPortWidth(S, 0, (int)mxGetPr(WIDTH)[0]);
                                                    /*DYNAMICALLY_SIZED*/
    ssSetInputPortDirectFeedThrough(S, 0, 1);

    if (!ssSetNumOutputPorts(S,1)) return;

    ssSetOutputPortWidth(S, 0, 12); /*GPS length=12 DYNAMICALLY_SIZED*/
    ssSetNumSampleTimes(S, 1);  /*Take care when specifying exception
                                  free code - see sfuntmpl_doc.c */
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE |
                 SS_OPTION_USE_TLC_WITH_ACCELERATOR);
}

/* Function: mdlInitializeSampleTimes
   Abstract:
   Specifies that we inherit our sample time from the driving block. */

static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}

/* Function: mdlOutputs */

static void mdlOutputs(SimStruct *S, int_T tid)
{
    int_T  i=0,j=0;
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
                                              /*Incoming data stream*/
    real_T  *y = ssGetOutputPortRealSignal(S,0);
    int_T temp[100];                /*make an aliase for the uPtrs*/
    char ext[]="\0";
    int count=0,len_in;
```

```
        for (i=0; i<(*uPtrs[0]); i++){temp[i] = ( int_T)(*uPtrs[i+1]);}

        *y++=(real_T)(temp[0]);
        *y++=(real_T)(temp[1]);
        *y++=(real_T)(temp[2] + temp[3]/10);
        *y++=(real_T)(temp[4]);
        *y++=(real_T)(temp[5] + (temp[6]*2^24 + temp[7]*2^16 +
                     temp[8]*2^8+ temp[9])/1000000);
        *y++=(real_T)(temp[10]);
        *y++=(real_T)(temp[11] + (temp[12]*2^24 + temp[13]*2^16 +
                     temp[14]*2^8 + temp[15])/1000000);
        *y++=(real_T)(temp[16]);
        *y++=(real_T)(temp[17]*256 + temp[18] + temp[19]/100);
        *y++=(real_T)(temp[20]*256 + temp[21] + temp[22]/100);
        *y++=(real_T)(temp[23]*256 + temp[24] + temp[25]/100);
        *y++=(real_T)(temp[26]*256 + temp[27] + temp[28]/10);
}

/* Function: mdlTerminate
   Abstract:
   No termination needed, but we are required to have this routine. */

static void mdlTerminate(SimStruct *S)
{
}

#ifdef MATLAB_MEX_FILE  /*Is this file being compiled as a MEX-file?*/
#include "simulink.c"   /* MEX-file interface mechanism*/
#else
#include "cg_sfun.h"    /*Code generation registration function*/
#endif
```

4. IMU DATA RECEIVE DRIVER

imudec.c

```
/* File : imudec.c
   $Revision: 1.00 $V.Dobrokhodov */

#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <iostream.h>
#include <string.h>

#define S_FUNCTION_NAME  imudec
#define S_FUNCTION_LEVEL 2

#include "simstruc.h"

/* Input Arguments */

#define NUMBER_OF_ARGS (1)
#define WIDTH ssGetSFcnParam(S,0) /*WIDTH is the max length of incoming
```

```
                                    IMU sentence*/
/* Build checking */

static char_T msg[256];

/* Function: mdlInitializeSizes
   Abstract:
   Setup sizes of the various vectors. */

static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S, NUMBER_OF_ARGS);

    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S))  {
        sprintf(msg,"Wrong number of input arguments passed.\n"
                "%d arguments are expected\n",NUMBER_OF_ARGS);
        ssSetErrorStatus(S,msg);
        return;   /*Parameter mismatch will be reported by Simulink*/
    }

    if (!ssSetNumInputPorts(S, 1)) return;

    ssSetInputPortWidth(S, 0, int)mxGetPr(WIDTH)[0]);
                                                /*DYNAMICALLY_SIZED*/
    ssSetInputPortDirectFeedThrough(S, 0, 1);

    if (!ssSetNumOutputPorts(S,1)) return;
    ssSetOutputPortWidth(S, 0, 14); /*IMU length=14 DYNAMICALLY_SIZED*/
    ssSetNumSampleTimes(S, 1); /*Take care when specifying exception
                                 free code - see sfuntmpl_doc.c */

    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE |
                 SS_OPTION_USE_TLC_WITH_ACCELERATOR);
}

/* Function: mdlInitializeSampleTimes
   Abstract:
   Specifies that we inherit our sample time from the driving block. */

static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}

/* Function: mdlOutputs */

static void mdlOutputs(SimStruct *S, int_T tid)
{
    int_T  i=0,j=0;  /*Incoming data stream,*uPtrs[0]- determine the
                       length of useful bytes{28 for the IMU}*/
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    real_T  *y = ssGetOutputPortRealSignal(S,0);
                            /*int_T width = ssGetOutputPortWidth(S,0);*/

    int_T temp[100]; /*make an aliase for the uPtrs*/
    char ext[]="\0";
```

```
    int count=0,len_in;
    for (i=0; i<(*uPtrs[0]); i++)   {temp[i] = ( int_T)(*uPtrs[i+1]);}

    for (i=0; i<(*uPtrs[0]))  {
        *y++=(real_T)(temp[i+0]*256 + temp[i+1]);
        i=i+2;
    }
        /*notused(j,1), airtemp, airspeed,q, r, p, phi ,tetta, Hx, Hy,
          Hz, Ax, Ay, Az*/
}
/* Function: mdlTerminate
   Abstract:
   No termination needed, but we are required to have this routine. */

static void mdlTerminate(SimStruct *S)
{
}

#ifdef  MATLAB_MEX_FILE  /*Is this file being compiled as a MEX-file?*/
#include "simulink.c"    /*MEX-file interface mechanism*/
#else
#include "cg_sfun.h"     /*Code generation registration function*/
#endif
```

LIST OF REFERENCES

1. www.baiaerosystems.com. 04-05-2003.
2. Tan, Kwang Liang, *Precision Air Data Support for Chem/Bio Attack Response*, Master's Thesis, Naval Postgraduate School, Monterey, CA, March 2003.
3. Kaminer, Isaac, *AA3276: Intro to Avionics, Course Notes,* Naval Postgraduate School, Monterey, CA, September 1996.
4. Rogers, Robert M., *Applied Mathematics in Integrated Navigation Systems*, AIAA Education Series, 2000.
5. SIMULINK Dynamic System Simulation for MATLAB, *Writing S-Functions Version 4*, November 2000.
6. Real-Time Workshop For Use with SIMULINK, *User's Guide Version 4*, September 2000.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, VA

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, CA

3. Assoc Prof. Isaac Kaminer
   Dept. of Aeronautics and Astronautics
   Naval Postgraduate School
   Monterey, CA

4. Dr. Vladimir Dobrokhodov
   Dept. of Aeronautics and Astronautics
   Naval Postgraduate School
   Monterey, CA

5. Prof. Max Platzer
   Dept. of Aeronautics and Astronautics
   Naval Postgraduate School
   Monterey, CA

6. LCDR Cristián Sir
   Chilean Navy
   Naval Aviation Headquarters
   Con-Cón, Chile

THIS PAGE INTENTIONALLY LEFT BLANK