2008-06

Advanced Ground Locomotion System for a Biologically Inspired Micro Morphing Air-Land Vehicle (MMALV)

Murphy, Corry

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

ADVANCED GROUND LOCOMOTION SYSTEM FOR A BIOLOGICALLY INSPIRED MICRO MORPHING AIR-LAND VEHICLE (MMALV)

by

Corry P. Murphy

June 2008

Thesis Advisor: Xiaoping Yun
Second Reader: Ravi Vaidyanathan

THIS PAGE INTENTIONALLY LEFT BLANK

January 24, 2019

SUBJECT: Change in distribution statement for *Advanced Ground Locomotion System for a Biologically Inspired Micro Morphing Air-Land Vehicle (MMALV)* – June 2008.

1. Reference: Murphy, Corry P. *Advanced Ground Locomotion System for a Biologically Inspired Micro Morphing Air-Land Vehicle (MMALV)*. Monterey, CA: Naval Postgraduate School, June 2008.
   UNCLASSIFIED [Distribution authorized to DoD Components only; Critical Technology; 15 June 2008. Other requests for this document must be referred to President, Code 261, Naval Postgraduate School, Monterey, CA 93943-5000 via the Defense Technical Information Center, 8725 John J. Kingman Rd., STE 0944, Ft. Belvoir, VA 22060-6218].

2. Upon consultation with NPS faculty, the School has determined that this thesis has been approved for public release, distribution is unlimited, effective January 23, 2019.

University Librarian
Naval Postgraduate School

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 2008	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE Advanced Ground Locomotion System for a Biologically Inspired Micro Morphing Air-Land Vehicle (MMALV)			**5. FUNDING NUMBERS**
6. AUTHOR(S) Corry P. Murphy			
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) Space and Naval Warfare Systems Center San Diego, CA 92152-5000			**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 ~~Distribution authorized to DoD Components only (Critical Technology); (15 June 2008). Other requests for this document must be referred to President, Code 261, Naval Postgraduate School, Monterey, CA 93943-5000 via the Defense Technical Information Center, 8725 John J. Kingman Rd., STE 0944, Ft. Belvoir, VA 22060-6218.~~ Approved for public release; distribution is unlimited.			**12b. DISTRIBUTION CODE** ~~E~~ A

13. ABSTRACT (maximum 200 words)

It is the purpose of this thesis to further advance research conducted in the development of an autonomous ground control for a Micro-Morphing Air-Land Vehicle. (MMALV). The intent of this system is to provide a small, low cost reconnaissance asset in the form of an unmanned aerial vehicle that is capable of flying, landing, and crawling to its target location. The biologically inspired MMALV has already been successfully integrated with the Kestrel Autopilot proving it capable of semi and full autonomous flight. This thesis will focus on the advanced development of the ground locomotion system by integrating a Gumstix microprocessor with the Kestrel Autopilot system. Research in this thesis has: 1) drawn upon biological inspiration to enhance MMALV's robustness, 2) extended development of the ground locomotion system to allow MMALV to navigate on the ground both semi and full autonomously, and 3) extended the capabilities of the MMALV by introducing on-board processing. Operational capability has been established through extensive hardware tests in realistic hostile environments.

14. SUBJECT TERMS Unmanned Aerial Vehicle, Robotics, Autopilot, Biologically-Inspired, Gumstix, Micro Morphing Air-Land Vehicle, Surveillance, Reconnaissance, WHEGs™			**15. NUMBER OF PAGES** 83
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** ~~UL~~ UU

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

THIS PAGE INTENTIONALLY LEFT BLANK

ADVANCED GROUND LOCOMOTION SYSTEM FOR A BIOLOGICALLY INSPIRED MICRO MORPHING AIR-LAND VEHICLE (MMALV)

Corry P. Murphy
Captain, United States Marine Corps
B.S.A.E, University of Maryland, College Park, 1999

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 2008

Author: Corry P. Murphy

Approved by: Xiaoping Yun
Thesis Advisor

Ravi Vaidyanathan
Second Reader

Jeffrey B. Knorr
Chairman, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

It is the purpose of this thesis to further advance research conducted in the development of an autonomous ground control for a Micro-Morphing Air-Land Vehicle (MMALV). The intent of this system is to provide a small, low cost reconnaissance asset in the form of an unmanned aerial vehicle that is capable of flying, landing, and crawling to its target location. The biologically inspired MMALV has already been successfully integrated with the Kestrel Autopilot proving it capable of semi and full autonomous flight. This thesis will focus on the advanced development of the ground locomotion system by integrating a Gumstix microprocessor with the Kestrel Autopilot system.

Research in this thesis has: 1) drawn upon biological inspiration to enhance MMALV's robustness, 2) extended development of the ground locomotion system to allow MMALV to navigate on the ground both semi and fully autonomously, and 3) extended the capabilities of the MMALV by introducing on-board processing. Operational capability has been established through extensive hardware tests in realistic hostile environments.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ABBREVIATIONS, ACRONYMS, AND SYMBOLS

BASIC stamp	Microcontroller with a BASIC interpreter built into ROM
BTUART	Bluetooth Universal Asynchronous Receiver/Transmitter
DC	Direct Current
ESC	Electronic Speed Controller
FFUART	Full Function Universal Asynchronous Receiver/Transmitter
FHSS	Frequency-Hopping Spread Spectrum
GCS	Ground Control Station
GPS	Global Positioning System
I^2C	Inter-Integrated Circuit
I/O	Input/Output
KAP	Kestrel Autopilot System
Kestrel	Procerus Technologies Autopilot
LOS	Line of Sight
MAV	Micro Air Vehicle
MMALV	Morphing Micro Air Land Vehicle
OE	Open Embedded
Packet	Data sent either via wireless or wire
PIC	Programmable Intelligent Computer
PWM	Pulse Width Modulation
RAM	Random Access Memory
RS-232	Recommended Standard 232
SPI	Serial Peripheral Interface
STUART	Standard Universal Asynchronous Receiver/Transmitter
TCP/IP	Transport Control Protocol/Internet Protocol
TTL	Transistor–Transistor Logic
UART	Universal Asynchronous Receiver/Transmitter
UAV	Unmanned Aerial Vehicle
WHEG	Wheel/Leg design developed at Case Western Reserve University

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

This thesis is to continue previous work on the Micro Morphing Air-Land Vehicle Unmanned Air Vehicle (MMALV UAV). The MMALV is a result of a US Air Force grant to develop an UAV to "have a wingspan under 12 inches, weigh less than 1 lb, and be capable of flying more than a mile to a target. Upon landing, it will morph into a terrestrial vehicle to crawl to a target area. Furthermore, MMALV will be able to regain flight after landing from a high position such as a high building, thus delivering maximum intelligence and surveillance to the warfighter" [1,3]. The focus of this thesis is to further the advancement of the ground locomotion mode of the MMALV by incorporating a Gumstix microprocessor.

Previous work on this project used a 12 inch wingspan and a 16 inch wingspan airframe. The 12 inch wingspan model was found unsuitable for the required payload. Both versions were also found to be unstable and not extremely robust. A new version of the 16 inch model is currently under development by the MMALV team. The new model is designed to be more controllable and robust than the previous models. Previous work also required the use of a small, lower power data transfer device. This requirement resulted from the need to reduce weight and also to have the ability to provide an aerial relay platform to network with the MMALV while it is in the ground locomotion mode.

The research on the previous MMALV designs included flight stability tuning with the Kestrel Autopilot System and minimum ground locomotion testing and development. The ground locomotion testing included different drive system considerations and a software interface with the Kestrel autopilot. The software interface controlled the Wheel/Legs (WHEGs™) through the autopilot by utilizing extra servo channels. A Logitech controller was used to send PWM signals to servos or ESCs for the WHEGs™ control. Procerus Technologies' Kestrel Autopilot is still currently utilized in the MMALV for autonomous flight and stability. This autopilot will be used in the development of the ground locomotion system, which is the purpose of this thesis.

Once the MMALV is in ground locomotion mode it will need to be controlled to reach a target location. A target could be a suspicious vehicle spotted by the MMALV while flying a reconnaissance mission overhead. Once a suitable landing spot is designated, the MMALV would land then crawl to the suspect vehicle and investigate. It is essentially a mobile sensor package armed with cameras, microphones and other sensors. The MMALV team is working to develop a robust WHEGs drive system that will survive the impact of landing and successfully maneuver the MMALV around the target area. Furthermore, the adding of an onboard microprocessor will be used to receive data from the autopilot and ground station to control the MMALV to a desired target location.

The ultimate goal of this thesis is to develop and improve the control of the ground locomotion system. The integration of a Gumstix microprocessor and the Kestrel Autopilot System into the new 16 inch wingspan model will bring the MMALV team closer to that goal. The addition of a microprocessor will also further expand the capabilities of the MMALV by introducing onboard processing. The microprocessor could also be used to control other sensor nodes and collect data.

The objective of this thesis is to develop an improved autonomous ground locomotion system that will be incorporated into the new 16 inch wingspan MMALV. The ground locomotion system will consist of three major components: drive system, controller, and Kestrel Autopilot System (KAP). The introduction of an on-board Gumstix microprocessor will function as the controller, which will be used to interface the autopilot and drive system. The desired end state is to control the ground locomotion system manually, or autonomously by giving it a GPS coordinate from the ground station. The microprocessor will process the data from the autopilot and ground station to control the drive system's WHEGs™.

In order to develop the ground locomotion system, six phases needed to take place: (1) derive a ground locomotion control algorithm, (2) develop the Gumstix for ground locomotion control processing, (3) interface the Gumstix to the drive system, (4)

interface the Gumstix to the autopilot for data input, (5) create a TCP/IP application that sends data packets from the ground station to the autopilot, and (6) integrate phase one through five to achieve end state.

The ground control algorithm generates the commands to drive the two WHEGs™ on the MMALV. The error is calculated from the difference between the command heading and the desired heading. The commanded direction is sent from a ground station and can be entered manually or calculated from the current position and target position. The current position and heading are measurements from the autopilot's sensors. The error determines the turn rate of the MMALV. The error limits is -180 to 180 degrees. Zero error the MMALV will move forward. As the error departs from zero, the left or right WHEG™ will decrease its rotation rate, resulting in a higher turn rate to move the MMALV to the desired heading.

The Gumstix will process this algorithm by integration of the KAP with WHEGs™ drive system. A lot of initial development work of the Gumstix was done prior to this integration. The Gumstix was first tested with two drive system configurations: two continuous servos configuration with servo controller, and two DC motors configuration with motor controller. Once these configurations were functioning properly with Gumstix, the Gumstix was integrated with the KAP. The Gumstix and KAP integration consisted of the modification of Gumstix and Kestrel Autopilot integration code created by Procerus Technologies. It was modified to fulfill the requirements of the ground control algorithm. This included receiving packets from the autopilot and ground station. These packets contained data consisting of the current heading, desired heading, current position, target position, and desired speed. Once the Gumstix was independently integrated with the drive system and KAP, they could now be united.

The consolidated ground locomotion system was completed by inserting the drive system control algorithm into the modified Gumstix and Kestrel Autopilot code. Once the combined code was function properly with the proper WHEG™ movements, the ground locomotion system was tested statically with success.

There is much more work to be done with the Gumstix and the ground locomotion system. Flight testing with this system still needs to be accomplished. Also, the ground locomotion system needs to be operationally tested with additional sensors and reconnaissance equipment, such as cameras. The Gumstix also has an abundant amount of expansion configurations that could be use to increase the mission and functionality of the MMALV.

ACKNOWLEDGMENTS

I would like to thank my thesis advisor, Professor Xiaoping Yun of the Electrical Engineering Department and my second reader, Professor Ravi Vaidyanathan of the Systems Engineering Department for their help, guidance and inspiration during this project. Professor Vaidyanathan brought this project to the Naval Postgraduate School after realizing the value the military students could bring. Next, I would like to thank Rich Bachman of BioRobots LLC for his support and guidance. He has helped further the progress of the project to where it exists now. I would also like to thank James Calusdian and Michael Clement of NPS. Their knowledge and support has helped me accomplish the goals of this thesis. Furthermore, I would like to acknowledge the financial support of SPAWAR, for allowing the purchase of the equipment used in this thesis. This work was performed under Contract N6600108WR00108. I would like to also thank the team at Procerus Technologies for their Gumstix software support used in this thesis. Last, but not least, I want to thank my wife, Whitney, and my three children, Caroline, Nathan, and Whitney. They have been an inspiration to me, and always gave me love and support.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PURPOSE

The purpose of this thesis is to continue previous work on the Micro Morphing Air-Land Vehicle Unmanned Air Vehicle (MMALV UAV). The MMALV is a result of a US Air Force grant to develop an UAV small in size and capable of flying more than a mile. Upon landing, it will morph into a ground locomotion mode and crawl to a target area, thus delivering intelligence and surveillance to the warfighter [1]. The development team working towards this goal is made up of the Naval Postgraduate School and BioRobots LLC. The focus of this thesis is to further the advancement of the ground locomotion function of the MMALV by incorporating a Gumstix microprocessor.

Previous work on this project used a 12 inch wingspan and a 16 inch wingspan UAV. The 12 inch wingspan model was found unsuitable for the required payload. Both versions were also found to be unstable and not very robust. A new version of the 16 inch model is currently under development by the MMALV team. The new model is designed to be more controllable and robust than the previous models. Previous work also required the use of a small, lower power data transfer device. This requirement resulted from the need to reduce weight and also to have the ability to provide an aerial relay platform to network with the MMALV while it is in the ground locomotion mode.

Once the MMALV is in ground locomotion mode it will need to be controlled to reach a target location. A target could be a suspicious vehicle spotted by the MMALV while flying a reconnaissance mission overhead. Once a suitable landing spot is designated, the MMALV would land then crawl to the suspect vehicle and investigate. It is essentially a mobile sensor package armed with cameras, microphones and other sensor nodes. The MMALV team is working to develop a robust Wheel/Legs (WHEGs™) drive system that will survive the impact of landing and successfully maneuver the MMALV around the target area. Furthermore, the adding of an onboard microprocessor will be used to receive data from the autopilot and a ground station to control the MMALV to a desired target location.

The ultimate goal of this thesis is to develop and improve the control of the ground locomotion system. The integration of a Gumstix microprocessor and the Kestrel Autopilot System into the new 16 inch wingspan model will bring the MMALV team closer to that goal. The addition of a microprocessor will also further expand the capabilities of the MMALV by introducing onboard processing. The microprocessor could also be used to control other sensor nodes and collect data.

B. PREVIOUS WORK

As mentioned above, previous work included research on a 12 inch and 16 inch wingspan model, shown in Figure 1. These early airframes were designed by University of Florida. Their designs had proved to be unstable and difficult to fly for military applications. The airframes also proved to be structurally unsound. As a result, BioRobots LLC developed an improved 16 inch wingspan MMALV. The hardened airframe is designed to survive hard landings and also be more aerodynamically stable. This design also took into account the need for more room to accommodate the ground locomotion system.

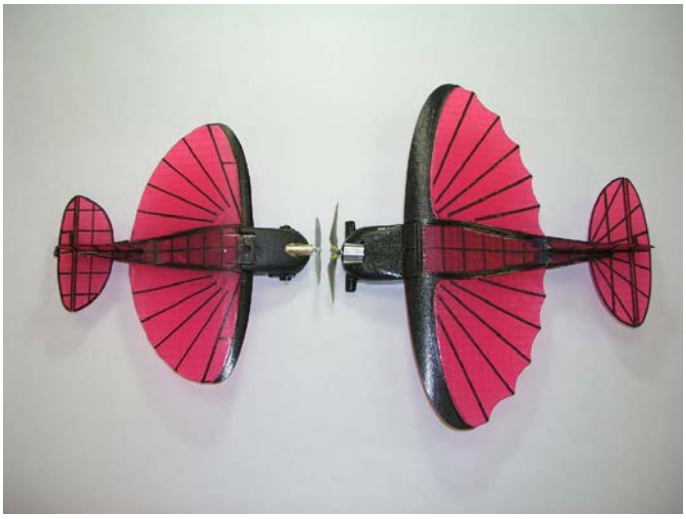

Figure 1. 12 inch and 16 inch MMALV [From 3].

The research on the previous MMALV designs included flight stability tuning with the Kestrel Autopilot System and minimum ground locomotion testing and

development. The ground locomotion testing included different drive system considerations and software interface with the Kestrel Autopilot. The software interface controlled the WHEGs through the autopilot by utilizing extra servo channels. A Logitech controller was used to send PWM signals to servos or electronic speed controller (ESC) for the WHEGs™ control [2]. Procerus Technologies' Kestrel Autopilot is still currently utilized in the MMALV for autonomous flight and stability. This autopilot will be used in the development of the ground locomotion system, which is the purpose of this thesis. A description of the Kestrel Autopilot system and WHEGs™ is discussed below.

Captain Robert Bledsoe, USMC had done previous research on replacing the current large, high power modem on the autopilot, with a smaller low powered modem which gives the MMALV data encryption and mobile ad hoc networking capabilities (XBee) [3]. This capability will be essential when the MMALV is on the ground and not in line-of-sight range of a ground station. A data relay or routing capability will be necessary to control the MMALV on the ground as well as to collect sensor data.

C. KESTREL AUTOPILOT SYSTEM

The Kestrel Autopilot System is created by Procerus Technologies. The autopilot is the heart of the MMALV, which is shown in Figure 2.

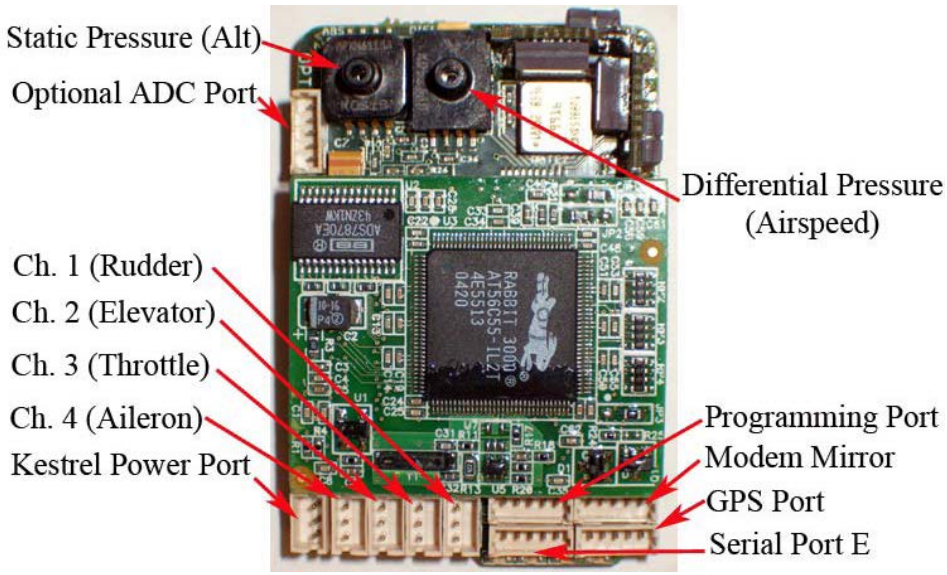

Figure 2. Kestrel Autopilot [From 2].

It provides the MMALV with autonomous take-off, landing, and flight, as well as GPS navigation. The system also includes Virtual Cockpit software and a ground communication station. The sensors onboard the autopilot includes the following: 3-axis angular rate sensor, 3-axis accelerometer, 2-axis magnetometer, absolute pressure sensor and differential pressure sensor. There is also an attached GPS receiver. In addition there are four serial ports that double as I/O ports. The GPS port is dedicated to the GPS receiver. The modem port is optional if modem is not already plugged into the header or if using an off-the-shelf modem. Two other ports are available for the user to employ, Port A (programming port) and Port E. All serial ports operate at TTL levels (0V to 3.3V) and can be configured for standard serial, SPI, or I2C communication. Serial port pin descriptions are shown in Figure 3.

Pin	Serial A	Serial E	GPS	Modem
1	GND	GND	GND	GND
2	PWR (3.3V or 5V)	PWR (3.3V or 5V)	PWR (3.3V or 5V)	PWR (3.3V or 5V)
3	TxA	TxE	TxD	TxF
4	RxA	RxE	RxD	RxF
5	Reset/Smode	Clock E	Clock D	Clock F

Figure 3. Kestrel Autopilot Pin Layout.

The serial ports can be configured using the payload window in the Virtual Cockpit software, shown in Figure 4. The payload window can be used to configure individual pins on the autopilot for a variety of functions. The functions include the following: bi-directional I/O pins, analog-to-digital converter pins, serial port pass through, camera pin switching, hardware-in-the-loop setup, and modem port mirroring. The integration of the autopilot and the Gumstix microprocessor will use serial port A as a modem mirror. The modem mirror allows all data transmission passed from the ground communication station to the autopilot's modem to be mirrored out this port.

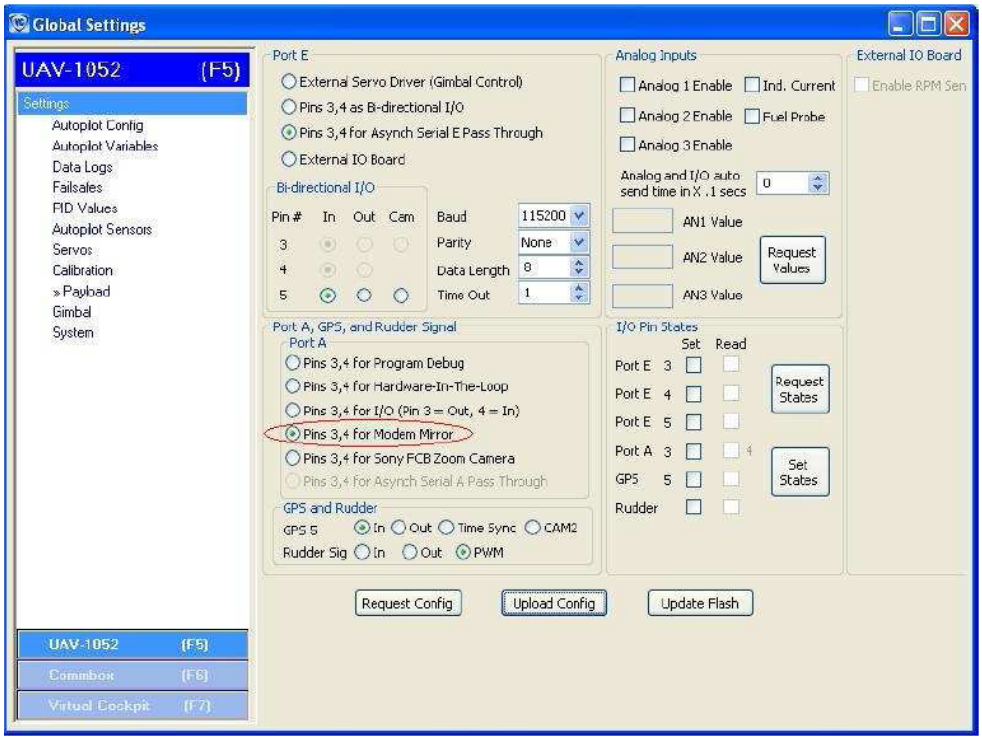

Figure 4. Virtual Cockpit Payload Window [From 6].

The components of the ground communication station consist of the Virtual Cockpit software and Commbox. Virtual Cockpit is a Windows based ground control, mission planning flight software for the Kestrel Autopilot. It allows the user to dynamically update mission profiles, such as airspeed, altitude and GPS waypoints. It also enables the user to apply different autonomous and semi-autonomous flight modes. The Virtual Cockpit is used to flight tune the MMALV by allowing user inputs with a

Logitech game pad and by enabling and disabling the autopilot control loops. Flight tuning on the new MMALV is concurrently being conducted by Captain Bryant Pater, USMC [4].

The Commbox transmits all data to and from the Virtual Cockpit and autopilot. It has a processor and a wireless digital modem. This modem is identical to the one connected to the autopilot. The modem currently being used is a 900 MHz FHSS Aerocomm modem. Although any wireless modem could be used with some firmware updates. The Commbox connects to a laptop with the Virtual Cockpit, by a RS232 serial cable. The layout is shown in Figure 5.

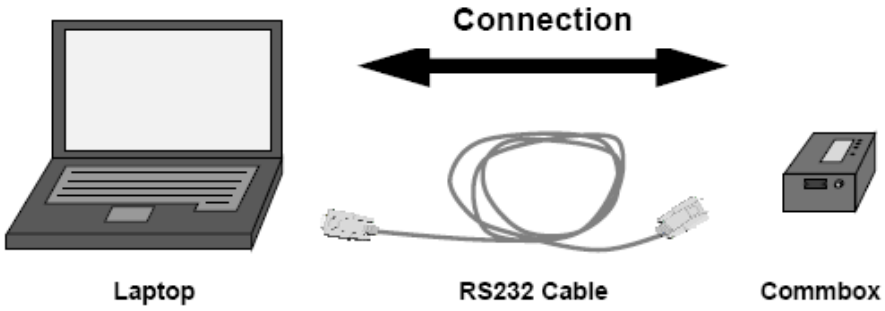

Figure 5. KAP Ground Control Station [From 5].

D. WHEGS™

WHEGs™ are a design that was developed by the Biorobotics Laboratory at Case Western University. It combines the efficiency of **Wh**eels and the terrain traversable ability of l**eg**s. The WHEGs are what enables the MMALV to crawl while in ground locomotion mode. They have a higher ground clearance than a wheel with the same radius, as shown in Figure 6. The focus of this thesis is to develop the ground control system to drive these WHEGs™ with the integration of a Gumstix microprocessor. The MMALV will have a left and right WHEG™ on the front of the aircraft. It will be controlled by differential steering. WHEGs™ obstacle clearance abilities and compliance was previously researched in a thesis by Capt Josh Kiihne, USMC [2]. The WHEGs™ are incorporated in the ground locomotion drive system.

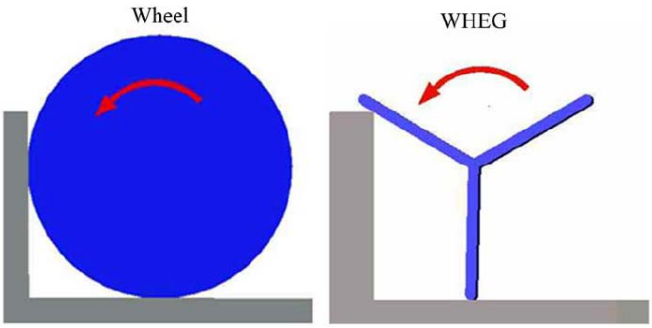

Figure 6. WHEGs™ [From 2].

E. THESIS OBJECTIVE

The objective of this thesis is to develop an improved autonomous ground locomotion system that will be incorporated into the new 16 inch wingspan MMALV. The ground locomotion system will consist of three major components: drive system, controller, and Kestrel Autopilot System. The introduction of an onboard Gumstix microprocessor will function as the controller, which will be used to interface the autopilot and drive system. The desired end state is to control the ground locomotion system manually, or autonomously by giving it a GPS coordinate from the ground station. The microprocessor will process the data from the autopilot and ground station to control the drive system's WHEGs™.

In order to develop the ground locomotion system six phases needed to take place: (1) derive a ground locomotion control algorithm, (2) develop the Gumstix for ground locomotion control processing, (3) interface the Gumstix to the drive system, (4) interface the Gumstix to the autopilot for data input, (5) create a TCP/IP application that sends data packets from the ground station to the autopilot, and (6) integrate phase one through five to achieve end state.

F. THESIS ORGANIZATION

The thesis is organized as follows: Chapter II will discuss the approach and theory of the ground locomotion system. Chapter III will discuss the microprocessor selection and development in order to control the ground locomotion of the MMALV. Chapter IV describes different drive system considerations and testing. Chapter V will discuss the

integration of the Gumstix microprocessor with the autopilot and ground station. Chapter VI will describe the complete integration of all ground locomotion subsystems. Chapter VII will summarize the work completed and include the shortfalls and recommendation for improvements.

II. APPROACH AND THEORY OF GROUND LOCOMOTION SYSTEM

A. CHAPTER OVERVIEW

This chapter will discuss the approach and theory of the design of the ground locomotion system. The first part of the chapter will discuss the approach to achieve the desired end state. The last part of the chapter will describe the theory of the ground locomotion system control algorithm.

B. APPROACH TO ACHIEVE END STATE

The desired end state of the ground locomotion system is to drive the WHEGs in order to move the MMALV to a desired ground target location. To accomplish this we need information to tell us where the MMALV is and where it needs to go. This information needs to be processed into commands to move the MMALV to its desired location. The ground locomotion system consists of three subsystems: the drive system, the Gumstix, and the Kestrel Autopilot System (KAP). The drive system will provide mechanical motion to move the MMALV on the ground. The Gumstix processes the control algorithm that produces the commands that are sent to the drive system. The KAP provides the information that tells us where the MMALV is and where it needs to move. A diagram of the ground locomotion design is shown below in Figure 7.

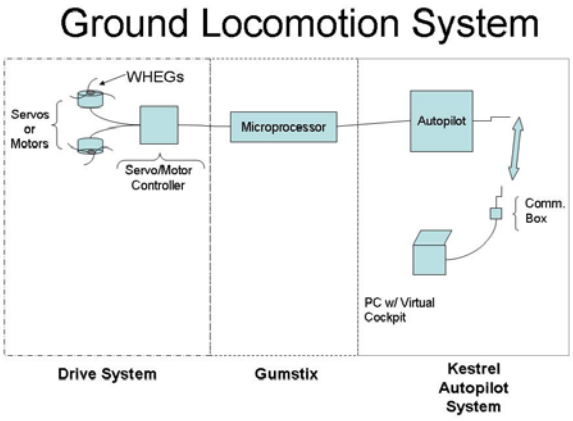

Figure 7. Ground Locomotion System Design.

In order to achieve a desired end state we needed to test the different components separately. Software and hardware development and testing were first done on a Linux host computer, and once the components were functioning correctly, the package was then built and installed on the Gumstix. Once all the components and software were functioning properly they were integrated to achieve the desired end state.

C. CONTROL ALGORITHM

The MMALV drive system will be driven by two WHEGs, powered by either two servos or motors. To maneuver the MMALV on the ground, differential steering is needed to turn left or right. It is desired to do this manually and autonomously. The easiest way to achieve this is to have the MMALV follow a specific compass direction. The direction could be entered manually or calculated from the autopilots GPS location and the target location (for autonomous mode) is sent from the ground station. Its speed can be varied at values of -10 reverse to +10 forward. This data will be sent from the ground station. Once the desired heading is established, the error is calculated by subtracting the current heading, received from the autopilots magnetometer, and the desired heading. This error is used to compute the turn rate, the higher the error the faster the turn rate. The maximum error is $\pm 180°$; negative error requires a left turn rate and a positive error requires a right turn rate. If the error is zero both WHEGs rotate at the same rate. If the error is non-zero one WHEG will rotate at max desired rate and the other rotate at a lower rate depending on the size of the error. The control algorithm is shown below in Equation 1. For the purpose of this discussion the WHEGs maximum rotation range could be 100 RPM.

$$\text{WHEGs}_{\text{rotation rate}} = \text{Max}_{\text{rotation range}} \times \frac{|\text{Speed}|}{10}\left[1 \pm \frac{\text{error}}{180}\right] \tag{1}$$

While in autonomous mode, distance to target is also computed from its current location and target location. Once the MMALV has reached its target location (i.e., distance to target is zero) it will stop and await further commands. An illustration of this control algorithm is shown below in Figure 8. This algorithm will be tested and utilized on the Gumstix.

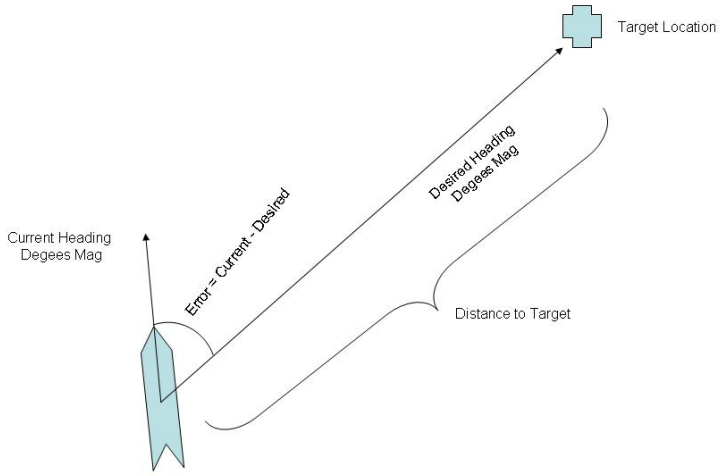

Figure 8. Illustration of the Ground Control Algorithm.

D. SUMMARY

A lot of hardware and software development was needed to complete the ground locomotion system. The next several chapters will discuss the steps that went into its development. Chapter III discusses the initial development of the Gumstix to enable us to build the program that computes the control algorithm and processes the data from the KAP in order to structure the commands to drive the WHEGs. Chapter IV will discuss initial control algorithm testing of the drive system configurations. Chapter V will discuss the integration of the KAP with the Gumstix in order to capture the useful autopilot data and ground station data. Chapter VI will integrate all the subsystems of the ground locomotion system.

THIS PAGE INTENTIONALLY LEFT BLANK

III. MICROPROCESSOR SELECTION AND DEVELOPMENT

A. CHAPTER OVERVIEW

This chapter will discuss the research that went into the design of adding a separate microprocessor to control the ground locomotion. The chapter begins by discussing two different microprocessor considerations. Next, we compare three different microcontrollers/microprocessors for use onboard the MMALV, and the basis for selecting the Gumstix microprocessor. An overview of the Gumstix and the setting up the build environment is then discussed. Finally, initial programming and testing of the Gumstix will also be discussed.

B. BACKGROUND

As stated in Chapter I, previous work had been conducted with the ground locomotion mode using a Logitech controller to maneuver the MMALV on the ground. The joystick movements sent signals of varying pulse widths through the Virtual cockpit and out the Commbox, which were received by the autopilot's modem and sent out a serial port to actuate the servo movements. This method of control required reprogramming of the autopilots firmware, which needed to be executed by Procerus Technologies. In order to alleviate this problem it is necessary to keep the autopilot modular and incorporate a new method in controlling the ground locomotion. This problem is solved by providing the MMALV with processing abilities by incorporating a microprocessor.

C. MICROPROCESSOR CONSIDERATIONS

The desired end state is to control the MMALV either manually or autonomously while in the ground locomotion mode. A microprocessor enables the MMALV to process control algorithms, and receive or send data from the autopilot's serial port. This will enable the MMALV team to continuously and freely update its own software without the dependency of Procerus Technologies. The microprocessor also increases he MMALV capabilities and functionality.

As stated in Chapter II, the objective is to input a desired target latitude and longitude in order navigate to the target autonomously. There are two methods of processing and delivering commands to the drive system of the MMALV. One method is to process the data at the ground station, another it to process the data on board the MMALV itself. Both methods where researched and discussed below.

1. Ground Station

Processing the data on the ground can be done using any type of programming language on the computer that is operating the Virtual Cockpit software. The user would enter the desired target location and an algorithm would compute the heading error between the MMALV and desired heading. The algorithm would then compute the direction the MMALV would need to travel and send the corresponding signals to the drive system. The heading would be constantly evaluated until the MMALV has reached its desired location. The drive system signals are transmitted out of the computer's serial port to the Commbox. The Commbox will then transmit the signals to the autopilot and out one of its serial ports to the drive system.

2. On Board the MMALV

Processing the data onboard the MMALV is done by incorporating a small microprocessor or microcontroller. The microprocessor will work in a similar fashion as a ground station. The microprocessor is interfaced between the autopilot and the drive system. It receives the ground target location via the ground communication station, as stated above. Once the MMALV is on the ground it will start the algorithm previously stated and drive the MMALV to desired target location.

3. Comparison of Ground vs. On Board Processing

Both of the above drive system methods are viable options. After further researching and investigation it was determined that the use of an onboard microprocessor would be the best approach. The first reason is that an onboard microprocessor would have less transmission error than processing from a ground station. When using a ground station for processing, every command would need to be

transmitted from the ground to the MMALV's wireless modem. There is more of a chance that the data will be lost or have errors through the wireless channel. The processed data that is being transmitted must also compete with the data being sent from the Virtual Cockpit. The second reason is that the sampling rate will be much higher with an onboard processor. There will not be the delays associated with the wireless medium such as propagation and transmission delays. Finally, onboard processing brings autonomy to the MMALV, once it receives the target destination there will be no need for further transmission over the wireless channel.

Using an onboard microprocessor would ensure the integrity of the data that is needed for ground locomotion. Data from the sensors on the autopilot are received directly from the autopilot, processed by the microprocessor and transmitted for ground locomotion. The ground station would only transmit small amounts of data at limited times. Some potential data transmitted from the ground station could be desired heading or GPS target location. Chapter 5 will cover in detail the integration and testing of the KAP and microprocessor.

D. MICROPROCESSOR SELECTION

Three different microprocessors or microcontrollers were considered for the ground locomotion of the MMALV. They were the PIC, BASIC Stamp, and Gumstix microprocessors. The determination for the selection of the microprocessor or microcontroller was based on the following: cost, size, straightforwardness, programming language, expansion and compatibility. After extensive evaluation of the three processors, the Gumstix proved to be the best match for the requirements of the MMALV.

The Gumstix proved to be the best for the MMALV due to its processing power, size, cost and compatibility. The Gumstix is a very powerful low-cost motherboard and is the size of, as the name states, a stick of gum (80mm X 20mm). The Gumstix supports C and C++ and many other programming languages. Many legacy software applications written in C and C++ were available for the integration of the Gumstix in the MMALV. Gumstix motherboards have several low-cost expansion boards available, which are

suitable for the needs for the drive system design. Procerus Technology had already done some initial testing with the Gumstix and the autopilot. This code was available for us to employ and modify. All these factors above steered the decision to utilize the Gumstix.

E. GUMSTIX MICROPROCESSOR

The Gumstix motherboard is the brains for the ground locomotion. The Gumstix motherboard selected for the MMALV is the verdex XL6P, shown in Figure 9. It has a 600MHz processor and 128 MB RAM. The motherboard has three different connections for expansion boards: a 60-pin Hirose connector, a 120-pin MOLEX connector and a 24-pin flex ribbon. The two expansion boards used with the motherboard are the breakout-vx and the console-vx which both have a 60-pin connection and are shown in Figure 10.

Figure 9. Gumstix Motherbord [From 7].

The console-vx provides three RS-232 UART serial ports through miniDIN8 connectors and a USB mini-B connection. The board also has the capability for three TTL UART ports through 0.1 inch holes, with some minor modification. The breakout-vx provides the three TTL UART ports through 0.1 inch holes and a USB mini-B connection. The console-vx is used for development and programming because of its miniDIN8 connectors. The breakout-vx is used in the MMALV in final development because of its compact size and reduced weight.

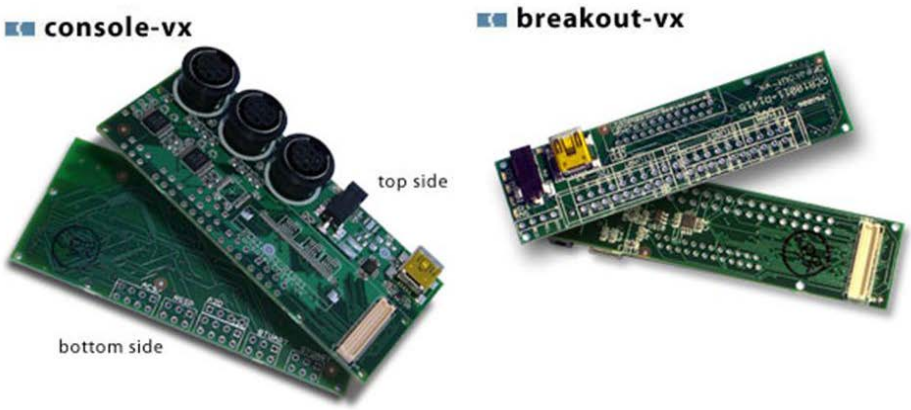

Figure 10. Gumstix Expansion Boards [From 7].

The integration between the autopilot and drive system will use the STUART and the BTUART serial ports on the Gumstix expansion board, Figure 11. The FFUART serial port is used for programming and interface with a host computer. The STUART interfaces to the autopilot and the BTUART interfaces to the drive system.

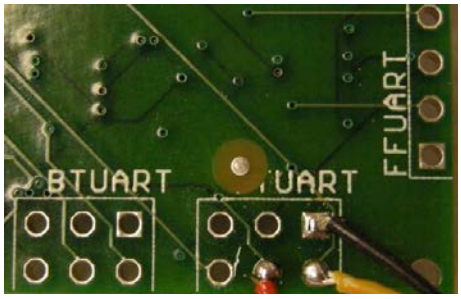

Figure 11. TTL UART Ports Console-vx.

F. GUMSTIX SOFTWARE DEVELOPMENT

Gumstix uses the Open Embedded (OE) build system. The OE system recently replaced the build-root system for software development on the Gumstix. OE requires a Linux distribution in order to begin software development, but plans are in place to allow it to work under Windows operating systems. OE uses the Bitbake task executor in combination with the OE metadata. Basically OE is a build system that can generate (cross-compile) software packages for embedded targets (Gumstix).

Before the programming can even begin on the Gumstix there are several steps that must be completed. The Gumstix is shipped as a bare-bones motherboard with no documentation. All documentation for the Gumstix is contained online at their developer's website (www.gumstix.net) [8, 9]. The Gumstix also requires an expansion board used for input and output computation. The console-vx expansion board will be used initially for transferring information to-and-from the Gumstix and the Linux host. The steps for getting started will be explained in the sections below. These steps include the following: setting up the build environment, setting up serial connection, and replacing the file system image. All of the steps are documented on the Gumstix developer's website.

1. Setting Up the Build Environment

Prior to setting up the build environment a Linux distribution was required. To begin with, a computer was loaded with an open source copy of the Fedora 8 Linux operating system. The computer was then installed with all the necessary applications needed to run OE. After that, the Gumstix OE source code was downloaded. Before installing the source code on the Gumstix, a global system cache needed to be set up. The Gumstix OE build process downloads the source code tarballs for the Linux kernel and other software packages. A tarball is a compressed file format. Setting up the global cache makes sure that these tarballs only needed to be downloaded once. Once this was completed, basic root file system was built using bitbake. The bitbake function will be explained later in this chapter. The basic root file system for the Gumstix took 12 hours to download. Once the basic root file system was downloaded and compiled, the file system image and the kernel image for the Gumstix were placed in the following directory: ~/gumstix/gumstix-oe/tmp/deploy/glibc/images/gumstix-custom-verdex. Now they are ready to be installed on the Gumstix. Before installing packages or programs on the Gumstix, it is necessary to set up a serial connection, as explained in the next section.

2. Setting Up a Serial Connection

The Gumstix is connected to the host computer using a null modem cable. This connection is from the host computer's serial port to the FFUART miniDIN8 connector on the Gumstix console-vx expansion board, which is the middle miniDIN8 connector. Next a communication terminal program was required, such as Kermit or Minicom, to enable serial connection. Kermit was recommended on the Gumstix development website. Once Kermit was installed, the serial connection parameters on the host computer needed to be configured for communicating with the Gumstix. Before connecting to the serial port it may be necessary to login as root or super user. The first command, kermit –l /dev/ttyS0, sets up the host computer's serial port. The serial parameters are saved in the following directory: gumstix/gumstix-oe/extras/kermit-setup. Typing the Kermit command 'take' and the file location above will set up the connection parameters. Connection to the serial port is done using the kermit command 'connect.' Then connecting the power to the Gumstix will start the normal boot sequence. The connection setup is summarized in Figure 12. Once the Gumstix boot sequence is complete one can login for the first time using: username root and password gumstix. Once the serial connection is established, the Gumstix's flash memory system can be reprogrammed.

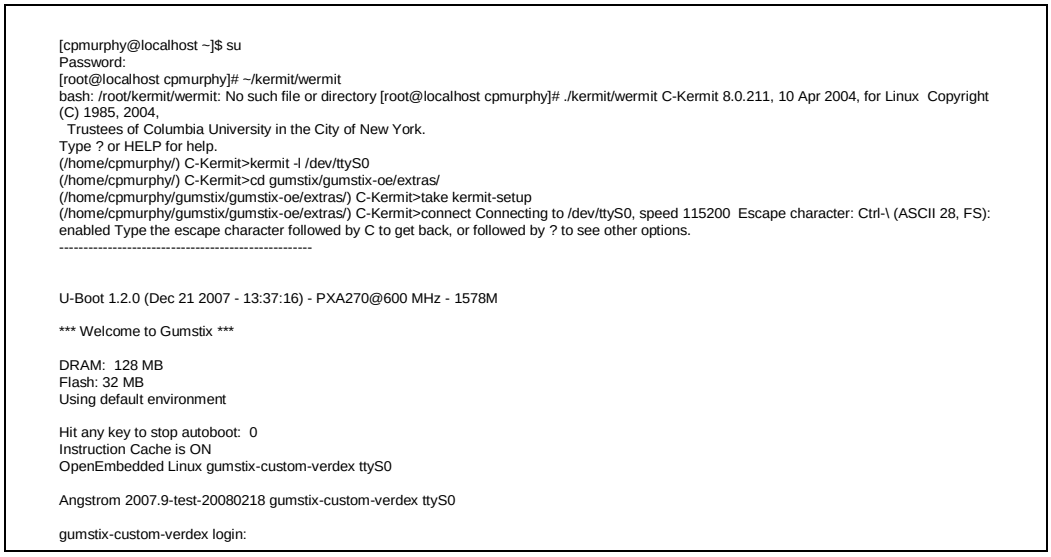

Figure 12. Gumstix Initialization.

3. Replacing the File System Image

Now that the new root file system is built and a serial connection is established, we can replace the file system image on the Gumstix. This is done by transferring the new root file system to Gumstix's RAM, protecting 2 boot sectors, erasing the rest of flash, and then finally programming the flash. First the root file system is sent to the Gumstix at memory location a2000000. After protecting sectors 0 and 1 in the flash, which contain the universal boot loader, the rest of the flash is erased. Next the root file system is copied from a2000000 to the flash address 400000. Once this is complete the kernel image is sent to the gumstix from the host, and then loaded and installed at flash address 100000. Now the Gumstix is configured and ready for development.

4. Programming

Programming on the Gumstix is done using a tool called BitBake, which is like 'make' function on Linux systems. BitBake is a tool for executing tasks and managing metadata. BitBake uses a recipe format that tells the build system the following: description of the package or program, version, source code location, how to configure and build source code and where to install the package. There are several ways to write these recipes, we will only talk about two methods: direct compilation of local sources and using a make file with local sources.

a. Direct Compilation of Local Sources

To initially test the Gumstix the simple C program *hello world* was used. The program is simply a "printf" statement that prints to the screen hello, world. This program was tested first by using direct compilation of local sources. It means exactly what its name states; the source files are located in the same file directory as the BitBake recipe on the host machine. The recipes are typically given the following name structure: **packagename_versionnumber.bb**. The hello world recipe will then be named, helloworld_1.0.0.bb. The recipe was provided by Gumstix on their development page and is shown below in Figure 13. Figure 13 also explains the recipes functions.

DESCRIPTION = "hello world sample program"
 	
PR = "r0"
 	
DEPENDS = ""
 	
SRC_URI = " \
  file://hello.c \
"
 	
S = "${WORKDIR}"
 	
do_compile () {
  ${CC} ${CFLAGS} ${LDFLAGS} -o hello hello.c
}
 	
do_install () {
  install -d ${D}${bindir}/
  install -m 0755 ${S}/hello ${D}${bindir}/
}
 	
FILES_${PN} = "${bindir}/hello"
	- Describes what the package is.
	
	-Revision number.
	
	-Dependencies
	
	-Source file location
	
	-Where the source code should be copied and built by the system. S is a BitBake variable that refers to the “unpacked source code directory.
	-Details on how to compile the package. Hello world uses a c cross compiler with standard compiler and linker settings, hello is the output file.
	-Tell the package management system how to install the program. It creates a directory and installs the hello command in that directory. D is a BitBake variable that specifies the destination directory. The bindir variable is a BitBake way to specify usr/bin
	
	- The final line tells the build system that we would like to create a package called "helloworld" that installs a single file: the binary "hello" that we generated above. The PN variable is yet another special BitBake variable that contains the package name.

Figure 13. Bitbake Recipe Format for Hello World.

Each package that is created is placed in the gumstix/gumstix-oe/user.collection/packages directory. The package directory is typically named the same as the recipe itself. The package recipe is placed in the packages directory and the source files are placed in a subdirectory called files. Once the recipe is complete, you simply type ‘bitbake helloworld’ from any directory location to compile and build the package. BitBake knows where to look for the recipes. After the build is complete the package will be placed in the following location with its appropriate name:

gumstix/gumstix-oe/tmp/deploy/glibc/ipk/armv5te/helloworld_1.0.0-r0_armv5te.ipk

There are two steps to install the program or package on the Gumstix. The first step is to send the above file to the Gumstix. While logged into the Gumstix type ‘rz’, that will set up the Gumstix to receive the file. Go back to the host machine and ’send’ the file using kermit. The second step installs the program/package on the gumstix. Connect to the Gumstix and type ‘ipkg install helloword_1.0.0-r0_arm5te.ipk’. Now the package is installed on the Gumstix and executed with the command hello.

b. Using a Makefile with Local Sources

BitBake can also use a makefile to build a package. If there is a legacy program that was previously built using a makefile this is a good approach to use. We will use the previous example to explain this. The recipe format does not change too much. Add the makefile to the SRC_URI list and remove the entire do_compile function. In the case of the helloworld example you will add ‘file://Makefile’ and remove the following:

```
do_compile(){
    ${CC} ${CFLAGS} ${LDFLAGS} –o hello hello.c
}
```

This function is executed in the makefile itself. The steps to build and test are the same as above.

G. SUMMARY

This chapter discussed the research that went into the decision of using the Gumstix as the ground locomotion system’s microprocessor. This chapter also explained how to develop and setup the Gumstix in order to begin testing and integrating the ground locomotion system. Once these steps were complete, drive system testing could begin using the Gumstix.

IV. GROUND LOCOMOTION DRIVE SYSTEM

A. CHAPTER OVERVIEW

The purpose of this chapter is to describe different drive system configurations that could be used on the MMALV. This chapter will also discuss the initial testing of these configurations with the Gumstix.

B. BACKGROUND AND THEORY

As stated in Chapter II, the ground locomotion system consists of three major components: drive system, Gumstix, and Kestrel Autopilot System. Extensive drive system considerations were discussed in depth in a previous thesis by Captain Josh Kiihne, USMC [2]. As stated in Chapter I, the design of the drive system WHEGs is currently under development with BioRobots LLC. There will be some discussion dealing with its design and development in this chapter.

The theory of controlling the drive system, whether it is a servo motor or a small DC motor, is essentially the same. Both operate by pulse width modulation. A servo motor expects a pulse at a period of 20 ms. In that period a pulse is transmitted, whose pulse width usually ranges from .9 to 2.1 ms with some nominal voltage. Sending a series of these pulses will rotate the servos to the desired location. If the servo is continuous, the servo will be at neutral position or stopped at a pulse width of 1.5 ms. Varying the pulse width less than or higher than 1.5 ms will cause the servo to rotate clockwise or counterclockwise at varying speeds.

A small D.C. motor operates with the same type theory as the servo motors. The motors will rotate when it receives a pulse width at a nominal voltage, varying from .9 to 2.1 ms. The motor is turned off at .9 ms and full speed at 2.1 ms. A motor controller will convert the pulses into a rate at which the transistors inside will switch on to drive the motors. The larger the pulse width the longer the switch is left on and the faster the motors will turn.

There are multiple ways of sending this data to the drive motors. One method is to send the pulse width directly to the servos or motors. Another is to use some type of controller, either a motor controller or servo controller.

C. DRIVE SYSTEM CONFIGURATIONS

There are two different drive system configurations that were considered for the MMALV: servo and motor driven. Both have their good and bad points. Previous work had discussed in detail these good and bad points [2]. The servo is the simplest method for the drive system. It can receive the signal directly from a three pin plug or a servo controller. The motor driven system requires an electronic speed controller or a motor controller to operate. The ESC receives the PWM signal and translates to a regulated voltage to drive the motors. The next two sections will discuss two different configurations of the drive system.

1. Dual Continuous Servo Configuration

Most servos operate with 90 to 180 degrees of motion. Servos will need to rotate continuously in order to drive the WHEGs™. To get the servos to operate continuously they need to be hacked into and modified, but certain manufacturers do produce continuous servos. To control the servo rotation, code can be written to send the required PWM or an easier method is to use a servo controller. A serial servo controller can generate the PWM required for the servo movement and it can do this for up to 16 servos almost simultaneously. The advantage to using a serial servo controller is that it is relatively easy to command multiple servos quickly. One major disadvantage is that it will add more weight to the MMALV and also add one more piece of hardware to troubleshoot. This device is a bit easier to configure than writing code that generates the required PWM signals.

2. Dual Motor Configuration

Brushless DC motors are another way to provide power to the drive system. In order to control these motors an electronic speed controller or a motor controller is necessary. For the MMALV drive system, it requires two small DC motors which will require two ESCs on one dual motor controller. In addition to the motors and controllers a gear box system would be required to provide the necessary torque for the WHEGs™. All of these components add weight and complexity to the MMALV. This type of drive system is viable to the MMALV because it adds robustness with higher torque and speed than a servo driven system.

D. DRIVE SYSTEM COMPONENTS

Each component used in the design consideration of the ground locomotion drive system will be discussed in the following sections. The Gumstix computes the control algorithm and outputs the required commands to a servo controller for servo movement or motor controller for DC motor movements. The servo controller and motor controller produce the required pulse width modulation to rotate the WHEGs™ at a rate that maneuvers the MMALV in the required direction. The drive system was tested and integrated with the Gumstix using two configurations: two continuous servos and two small DC motors. The servos were tested with a Pololu Micro Serial Controller and the motors were tested with the Pololu Low Voltage Serial Motor Controller. Both configurations were first tested on a Linux machine and then the Gumstix. The components of the two configurations are discussed below.

1. Micro Serial Servo Controller

A micro serial servo controller was used to control two Parallax continuous servos. The Pololu Micro Serial Servo Controller shown in Figure 14 computes the processor-intensive task of simultaneously generating multiple servo control PWM signals. The controller at the top left of Figure 14 is a Pololu Micro Serial Servo Controller with a DB9 serial connection for easy connection with a host computer. The controller generates the pulses from .25 ms to 2.75 ms. Internally, the servo controller

maintains a servo position that is two times the pulse width, measured in microseconds. A pulse width of 1.0 ms is represented by 2000 microseconds internally. The internal position ranges are from 500 to 5500.

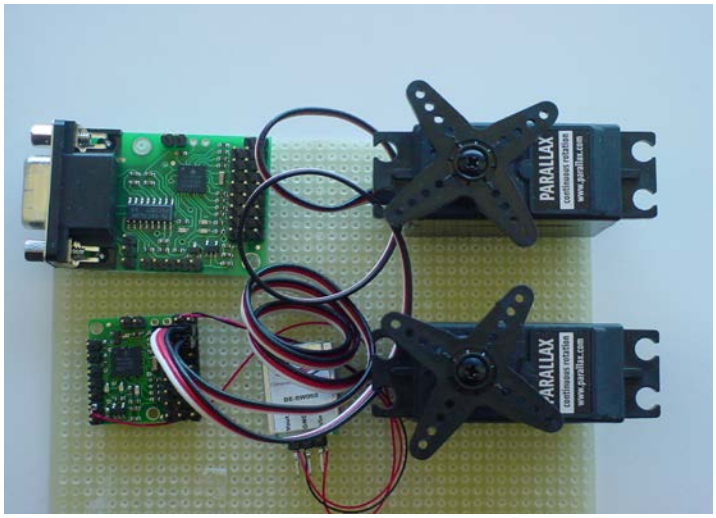

Figure 14. Dual Continuous Servo Test Configuration.

Serial commands are sent to the servo controller 5 or 6 bytes at a time, depending on the command. This controller can receive both TTL and RS-232 serial inputs. The controller's pin layout is shown in Figure 15.

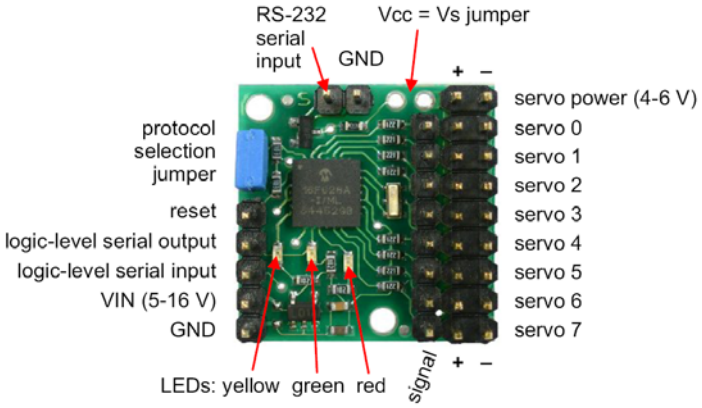

Figure 15. Pololu Micro Servo Controller Pin Layout [From 10]

Various commands deal with setting the internal ranges mentioned above. For the purposes of this thesis we will be using continuous servos. The command used for controlling the continuous servos is set position (absolute). The command structure for the byte sequence is shown below in Figure 16 [10].

Start byte = 0x80	Device ID = 0x01	Command = 0x04	Servo num	Pos =500 to 5500

Figure 16. Pololu Micro Servo Controller Command Structure.

2. Dual Serial Motor Controllers

Serial motor controllers are similar to the servo controllers. They are used to control small, low current DC motors (1-5 Amps). The Pololu Low-Voltage and Micro Dual Serial Motor Controllers were both used in the development of the drive system. The initial testing of the dual motor configuration was done using the Pololu Low-Voltage Motor Controller and the Tamiya double gearbox kit both shown in Figure 17.

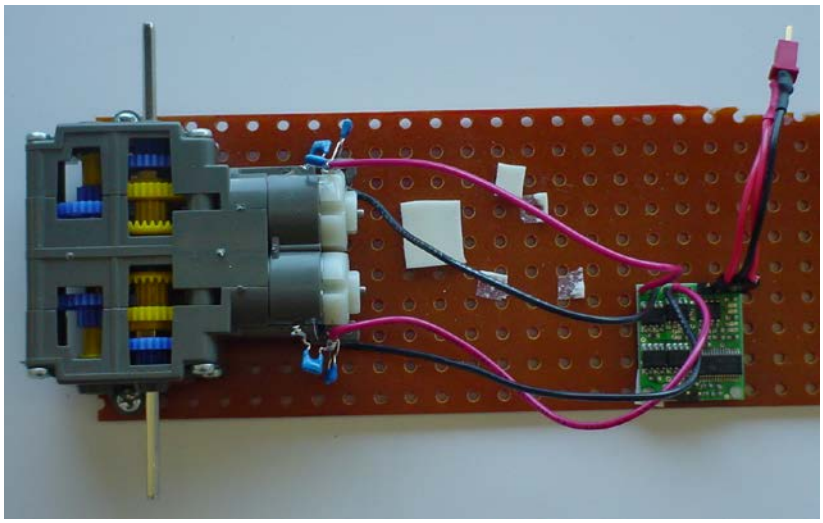

Figure 17. Dual DC Motor Test Configuration.

Both controllers use a similar command structure as the Pololu servo controller, but only use 4 bytes. The command structure is shown below in Figure 18.

Start byte = 0x80	Device ID = 0x00	Motor # and direction	Motor speed= 0 to 127

Figure 18. Pololu Dual Motor Controller Command Structure.

The Low-Voltage controller can receive both TTL and RS-232 serial inputs. This controller was used in initial development. The Micro controller only uses TTL serial inputs. This controller will be used in the final design due to small size and weight. These controllers receive the serial commands from the Gumstix by the same method as the servo controller. Both motor controllers pin layouts are shown below in Figure 19.

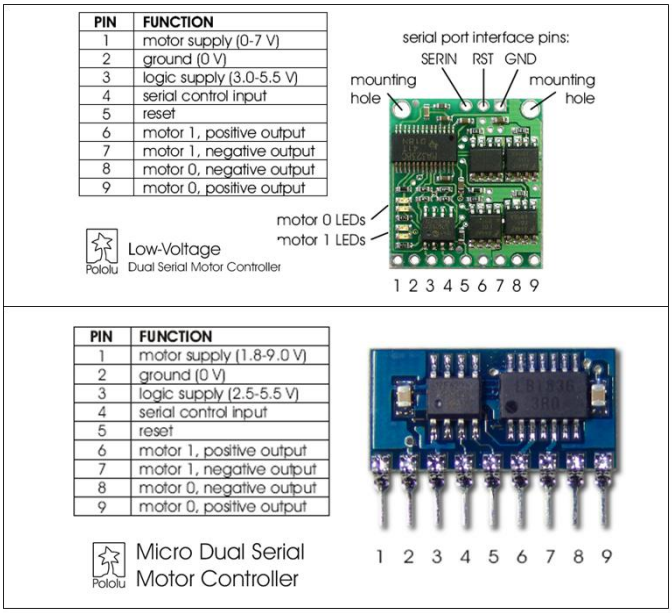

Figure 19. Pololu Dual Serial Motor Controller Pin Layout [From Refs. 11, 12].

3. Drive System Final Configuration

The basic operation of servos and the motors are essentially the same. Transition from the test equipment to the final design will be relatively easy to do. The drive system that will be used in MMALV will be a dual motor design. The design uses two Solarbotics 150:1 mini metal gear motors (Figure 20) controlled by the Pololu Micro

Motor Controller. The motors will be connected to the WHEGs through a clutch system to prevent motor and gear damage when landing. The clutch system schematic is shown in Figure 21.

Figure 20. Solorbotics 150:1 Mini Metal Gear Motors [From 13].

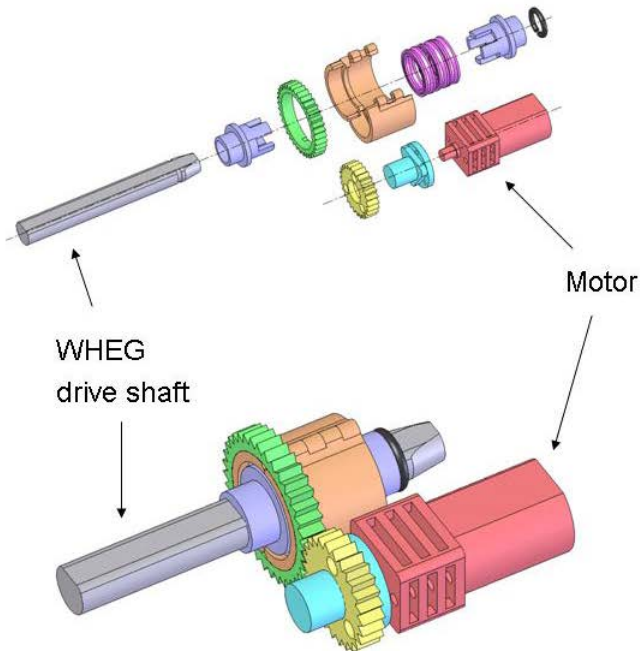

Figure 21. Clutch Schematic Exploded and Fully Assembled [From 14].

E. DRIVE SYSTEM DEVELOPMENT, TESTING AND INTEGRATION

In order to achieve a desired end state we needed to test the different configurations separately. Software and hardware development and testing were first done on a Linux host machine and once the components were functioning correctly the package was then built by BitBake and installed on Gumstix. Once all the components and software were functioning properly they were integrated with the Gumstix.

The motor and servo controllers were tested on a Linux machine using C programming. Programming examples can be found on the Pololu website. The program used in development of the drive system was written by a NPS research associate, Michael Clement. The program opens the serial port, sets up the serial parameters, and writes the command data structure explained above. The code is incorporated in the final ground locomotion program in appendix A. Once the servos or motors were functioning properly they were tested on the Gumstix. In order to test the program on the Gumstix the program needed to be built with bitbake using the helloworld example in Chapter III.

The serial connections for servo and motor controllers can utilize both RS-232 and TTL voltage levels. The RS-232 serial line connects the controllers to the Linux Host machine using pins 3 (transmit) and 5 (ground) of the DB9 serial port connector. The TTL serial line connects the controllers to the Gumstix using the BTUART serial port's 0.1 inch through-hole pins, pin 1 (ground), pin 2 (transmit) and pin 3 (Vcc, logic supply). The Gumstix connections are shown in Figure 22 for the console-vx and the breakout-vx. Initial function testing of servo and motor configurations are discussed in the following sections.

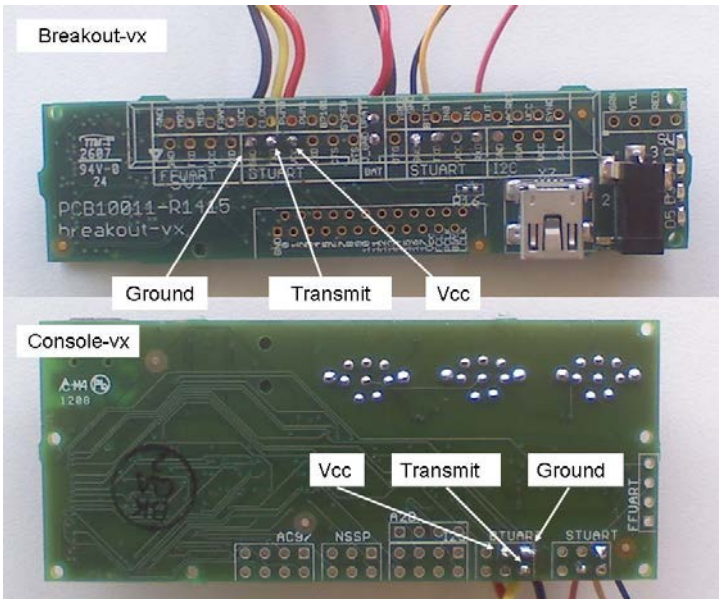

Figure 22. Console-vx and Breakout-vx TTL Serial Connections.

1. Servo Configuration

As stated above, the servo configuration was tested with the Pololu Micro Serial Servo Controller and two Parallax continuous servos. Even though this configuration is not going to be currently used on the MMALV, it still may be a viable option in the future. The testing consisted of finding the neutral point of the servos and their max speeds. Byte 5 of the command structure, in Figure 16 above, is the range for the servo position ranging from 500 to 5500. The neutral point was found at 3000. Once the neutral or stop position was found, commands were sent to the controller above and below 3000 to find the max rotation rate. Maximum clockwise rotation was found at 3500 and maximum counterclockwise rotation was found at 2500, no where near the limits of the of the command range.

Once these values were found, the modified control algorithm from Chapter II was tested. The rotation rate can vary between 0 and ±500 from the neutral position for the clockwise and counterclockwise directions. The corrected algorithm is shown in Equation (2).

$$\text{WHEGs}_{\text{rotation rate}} = 3025 \pm \left[500 \times \frac{|\text{Speed}|}{10} \left[1 \pm \frac{\text{error}}{180} \right] \right] \quad (2)$$

The algorithm was tested by inputting headings (desired and current) and speeds. Once these tests were successful on the Linux host and Gumstix, the motor controllers were tested.

2. Motor Configuration

The initial testing for the motor configuration was done on the Pololu Micro Dual Motor Controller and the Tamiya double gearbox kit. This motor controller was used for initial testing because it used both the RS-232 and TTL serial lines. The motor controllers operate by sending it a 3 or 4 byte command. The three byte command structure sets up the motor configuration, the third byte is shown below in Figure 23. The motor controllers for the drive system were set up for 2 motors with the right motor number '0' and the left '1'. The range for the motors speed is simply 0 for off, to 127 for fully on. This is set by byte four of the command structure in Figure 18, where bit 7 is always 0. To reverse the motor you must set bit 0 of the byte 3 (1=forward, 0=reverse).

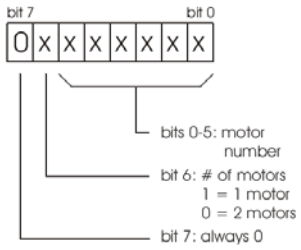

Figure 23. Byte 3 Motor Command Structure [From 11].

As in the servo controller, the motor was tested with a modified algorithm from Chapter II. The algorithm is shown below in Equation 3.

$$\text{WHEGs}_{\text{rotation rate}} = 127 \times \frac{|\text{Speed}|}{10}\left[1 \pm \frac{\text{error}}{180}\right] \quad (3)$$

This algorithm was also tested by inputting headings and speeds. The negative or reverse speeds were accomplished by a condition statement in the code. If the speed is negative, change command byte 3, bit 0 to a zero, Figure 23. Once the code was running correctly is was built and installed on the Gumstix. Both Pololu motor controllers will function from the same code as long as they are both configured the same.

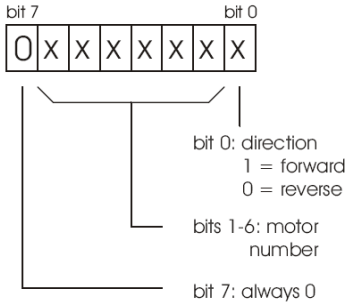

Figure 24. Byte 3 Motor Configuration Command Structure [From 11]

F. SUMMARY

This chapter discussed two different drive system configurations and their testing. Although servos are slow and require high peak currents, the servo configuration is a viable option because of its simplicity when compare to a motor gear box configuration. However, the dual motor configuration provides the MMALV with greater speeds and torque. The small Solarbotics motors provide the MMALV with a small, powerful and light weight (8grams) option. These motors weight less than most servos. They also have their own gear box and are relatively inexpensive. These motors and the Pololu Dual Micro Motor Controller had shown to be the best configuration for the MMALV's ground locomotion drive system. The integration of the final drive system configuration into the ground locomotion system will be explained in Chapter VI.

THIS PAGE INTENTIONALLY LEFT BLANK

V. GUMSTIX AND KESTREL AUTOPILOT SYSTEM INTEGRATION

A. CHAPTER OVERVIEW

This chapter discusses the integration of the Gumstix and the KAP. The first part of this chapter explains the integration of the Gumstix and autopilot using development open source code provided by Procerus Technologies. The next part of this chapter illustrates the development of the TCP/IP application that enables the sending and receiving of ground locomotion data packets from the ground station. The last section of this chapter will discuss the testing of the Gumstix and KAP application software.

B. AUTOPILOT AND GUMSTIX INTEGRATION

The control algorithm from Chapter II requires sensor readings from the autopilot. The Gumstix's STUART serial port is used to obtain the MMALV's current heading and position from the autopilot's serial port, set as a modem mirror. The connection is shown below in Figure 25.

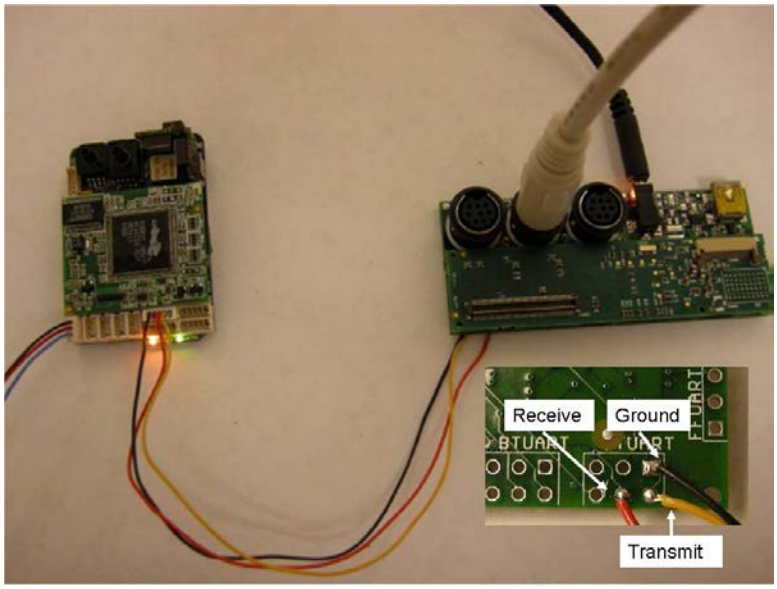

Figure 25. Gumstix and Autopilot Connection [After 6].

The magnetic heading is obtained from the magnetometer onboard the autopilot. The position is obtained from the autopilot's GPS receiver. Receiving this data is done by modifying Gumstix development example code created by Procerus Technologies [6]. The code is written in C++ and can be installed on any Linux system as well as the Gumstix. This code enables the user to communicate with the Kestrel autopilot. The code sets up the serial communication protocol, and enables the user to send and receive packets from the autopilot. The code's details and testing are discussed in the sections below. To understand how to modify the code for use on the MMALV the autopilot's packet format is described below.

1. Kestrel Autopilot Packets

The Kestrel Autopilot System uses packets to send and receive messages from the Kestrel autopilot and ground control station. The Kestrel communication protocol has many different packet types, which are called Kestrel packets. The ground locomotion system only needs to utilize a few of them. The Kestrel packet format is shown below in Figure 26. Three types of packets are used for the ground locomotion system, which are GPS navigation (packet 248), standard telemetry (packet 249), and modem mirror bridge (packet 245). The navigation and telemetry packets are sent by the autopilot out of its modem mirror port on receipt of a navigation or standard telemetry request packet (packet 27 or 26).

Byte	Name	Description
0	Start Byte	(Always **0xFF**) Signals start of packet
1	Type	This particular packet type
2	Src Addr Low	Low byte of packet source address (sender)
3	Src Addr High	High byte of packet source address
4	Dest Addr Low	Low byte of packet destination address (receiver)
5	Dest Addr High	High byte of packet destination address
6	Packet Data	Start of packet data, determined by packet type
...	...	
n-3	Packet ID number	A sequential number generated by the ground station and returned by the autopilot
n-2	XOR Check 1	High byte of XOR Check
n-1	XOR Check 2	Low byte of XOR Check
n	End Byte	(Always **0xFE**) Signals end of packet

Figure 26. Kestrel Autopilot System Packet Format [From 15].

These packets contain a lot of information, but only the required data bytes will be pulled from these packets. The current heading is pulled from byte 14 of the standard telemetry packet, and the latitude and longitude are pulled from bytes 14 and 20, respectfully, of the navigation packet. The modem mirror bridge is used to forward data to and from the modem mirror port. These packets will be used to send data from the ground station to the Gumstix or from the Gumstix to the ground station. These packets will be discussed in more detail below in the ground station TCP/IP application section.

2. Gumstix Code Development

The Procerus example code doesn't initially do much for the ground locomotion system. However this code will become the backbone of the system after development. The main() function in the code performs three separate functions or threads. The main() function will also be the location for the ground control algorithm. The first thread sets up the protocol that receives the packets from the serial port. The second function opens the serial port. The serial port on the Gumstix that is being used is the logic level STUART port (/dev/ttyS2). The third function of the code creates a thread that requests the standard telemetry packet and GPS telemetry packet at a rate of 10 Hz. The code has an object file that implements the sending and receiving of packets from the serial port, which is the CommSerialLinux class. The following section will describe the CommSerialLinux class, sending of packets, and receiving of packets.

a. CommSerialLinux Class

The CommSerialLinux class enables the sending, receiving and formatting of the Kestrel autopilot packets. The object it creates is `g_CommSerial`. The `CommSerialLinux` class produces a thread that is always monitoring the serial port for the packets. If a packet is received on the port a registered callback function (`ReceivePacket()`) is used to process the packet. This class also has a function to send packets to the autopilot, which is called `SendRaw()`.

b. Sending Packets

To send packets to the autopilot another class was created within the code, which is `CWriteRawPacket`. This class builds the packets up into its proper format shown in Figure 26. The example code initially comes with a request telemetry function call that sends the request standard telemetry packet (26). The request is written as follows: `CWriteRawPacket ReqStdTelmPkt (26, 0xFFFF).` The ReqStdTelmPkt is a casting variable that will be used to send the packet. The number 26 refers to the packet identification number and the 0xFFFF is the broadcast address. The request for standard telemetry packet (26) is an empty data packet. If the packet contained data, the functions in Figure 27 would be used for various data sizes.

Function name	Data type
AppendUnsignedChar	Unsigned 1-byte
AppendUnsignedShort	Unsigned 2-byte (Kestrel communication refers to these as unsigned ints)
AppendShort	Signed 2-byte (Kestrel communication refers to these as ints)
AppendUnsignedInt	Unsigned 4-byte (Kestrel communication refers to these as longs)
AppendFloat	4-byte floating-point

Figure 27. Data Writing Functions [After 15].

The last step in building the packet is to assign a packet sequence number, which is returned when sending the packet. The command is written as follows: `ReqStdTelemPkt.SetPacketIDNum (counter ++).` Now the packet is built and ready to be written to the serial port. The function for writing comes from the `CommSerialLinux` class, which is `SendRaw().` This function will compute the checksum of the packet, apply the escape character encoding, and send it to the serial port. The command to write the packet built above is the following:

```
g_CommSerial.SendRaw(&ReqGPSTelemPkt).
```

These commands will request the standard telemetry packet from the autopilot; as a result the autopilot will send the packets to the modem mirror port. These packets will then be received by the Gumstix serial port. The standard telemetry packet will have the magnetometer data needed for our control algorithm, which will be the current heading of the MMALV. To obtain the MMALVs current location, the GPS telemetry packet must also be requested. This is done in a similar fashion as the request for standard telemetry packets, described above. The request GPS telemetry code is shown below in Figure 28.

```
//Request the GPS telemetry
CWriteRawPacket ReqGPSTelemPkt(GPS_TELEMETRY_REQUEST_PACKET, 0xFFFF);
ReqGPSTelemPkt.SetPacketIDNum(counter++);
g_CommSerial.SendRaw(&ReqGPSTelemPkt);
usleep(1000000); //Sleep for .1 seconds
```

Figure 28. Request for Autopilot GPS Telemetry Code.

c. Receiving Packets

Now that the required data packets are requested from the autopilot, we need to receive them. As stated above, a register call back function specifies which function will handle the incoming packets. When a packet is received it is dynamically allocated to memory by the `CommSerialLinux` class, and then is passed to the calling function. Once the packet is passed on it is erased from memory. The function that calls for the packets is called `ReceivePacket`(). In this function the generic packet is recast as the object type it really is. The code has three common packet type classes that make it easy to gain access to the packet's data elements: AckPacket, GPSTelemPacket, and StdTelemPacket. These classes parse their appropriate packet type and save the corresponding data bytes as user functions.

When the `ReceivePacket()` function is called it returns the packet's location in memory (`*NewPkt`) and its type. To pull the magnetometer heading from the standard telemetry packet a switch statement is used in the example code. The standard telemetry type is 248 or `STD_TELEMETRY_PACKET`. When a packet arrives

at the serial port and the type is identified at `STD_TELEMETRY_PACKET`, the NewPkt will be recast as StdTelemPacket. To retrieve the heading data from the packet the code uses functions that are defined in the StdTelemPacket class, such as `GetHeading().` The receive packet function is shown below in Figure 29.

```
void ReceivePacket(CCommPacket *NewPkt, PacketType Type)
{
        //Casting variables
        CStdTelemPacket *StdTelemPacket;
        CGPSTelemPacket *GPSTelemPacket;
        //Simple error checking
        if(NewPkt == NULL) return;
        switch(Type)
        {
        case STD_TELEMETRY_PACKET:
                StdTelemPacket =reinterpret_cast<CStdTelemPacket*>(NewPkt);
        Heading_Autopilot = StdTelemPacket->GetHeading()*57.3;
        break;
        case GPS_TELEMETRY_PACKET:
                GPSTelemPacket =reinterpret_cast<CGPSTelemPacket*>(NewPkt);
                //lat2 = GPSTelemPacket->GetGPSLatitude();
                //lon2 = GPSTelemPacket->GetGPSLongitude();
                lat2 = 36.595202;
                lon2 = -121.875072;
        break;
        case 245:
                Heading = NewPkt-> ReadUnsignedShort(7);
                speed   = NewPkt-> ReadShort(9);
                lat1    = NewPkt-> ReadFloat(11);
                lon1    = NewPkt-> ReadFloat(15);
                Mode    = NewPkt-> ReadUnsignedChar(19);
        break;
        }
}
```

Figure 29. Receive Packet Function.

At the bottom of Figure 29, there is a '`case 245`' statement. This is the modem mirror bridge packet that is sent from the ground station. This packet is parsed directly in the function because it doesn't have a class of its own, such as standard telemetry packet. Furthermore, the read functions shown in Figure 29 allow the user to access each data element. This packet type will be explained more in detail in the following sections.

The receive packet function will produce the global variables for the ground locomotion system, which are autopilot heading, autopilot latitude and longitude, desired heading, target latitude and longitude, speed, and mode. The desired heading, target location, speed and mode are all sent from the ground station via a TCP/IP application which will be described in the section below.

C. GROUND STATION TCP/IP APPLICATION

To control the MMALV in the ground locomotion mode there needs to be a way to send user defined data packets. The requirement is to have the MMALV follow a desired heading or navigate to a target location. It is also desired to be able to vary the speed of the MMALV on the ground. To accomplish this there needs to be a method to send data remotely. This is done by utilizing the TCP/IP socket server that is built into the Virtual Cockpit. This server accepts connections from outside applications. The application used to build and send packets is LabVIEW. The following sections will describe the packet generation and the development LabVIEW application.

1. Ground Station Packet Creation

The Kestrel modem mirror packet 245, shown in Figure 30, is the packet that will deliver the data required for the ground control algorithm. To send this packet to the autopilot it is necessary to use a development interface packet.

Byte Index	Type	Name	Description	Units
6	UCHAR	Number bytes	Number of data bytes contained in the packet	N/A
7	Char	Data	Bytes 1 through n	N/A

Figure 30. Modem Mirror Bridge Packet 245 [From 15]

The Virtual Cockpit development interface has several different types of interface packets. The interface packet type that is used to forward packet 245 is the pass-through packet (10). This packet is sent to the Virtual Cockpit's TCP/IP port 5005. Virtual Cockpit will translate the packet to a Kestrel packet (245) and structure the data to its proper format shown in Figure 26. The generic interface packet and pass-through packet are shown below in Figures 31 and 32.

Byte Index	Type	Name	Description	Units
0	32-bit INT	Type	The type of packet being sent. Use Table 4.1	N/A
4	32-bit INT	Data Size	Number of bytes in the data portion of the packet starting at byte index 8	N/A
8	BYTEs	Data	The data that makes up the packet. Up to 13,000 bytes	N/A

Figure 31. Generic Interface Packet Format [From 15]

Byte Index	Type	Name	Description	Units
8	16-bit INT	Dest/Src Address	The destination if sending or source address if receiving from an autopilot	N/A
10	8-bit UCHAR	Kestrel Packet Type	The kestrel packet type as described in chapter 4	N/A
11	BYTEs	Data	The data that makes up Kestrel communications packet	N/A

Figure 32. Pass-Though Packet Format [From 15]

To build the pass-through packet, the first thing that needs to be done is to build the modem mirror bridge packet (245) with the required data for the ground locomotion system. The data for the modem mirror packet will start at byte 11 in the pass-through packet. Once Virtual Cockpit processes it into packet 245 it will be placed at byte 6. The complete pass-though packet is shown in Figure 33.

Byte Index	Type	Name	Description	Units
0	32 bit INT	10	Type of Development Packet (pass-through)	N/A
4	32 bit INT	16	Data bytes starting at byte 8	N/A
8	16 bit INT	1032	Destination address	N/A
10	UCHAR	245	Type of Kestrel Packet	N/A
11	UCHAR	13	Number of data bytes in packet 245 (byte 6 of Kestrel packet)	N/A
12	16 bit INT	Heading	Target heading	degrees
14	16 bit INT	speed	WHEGs speed	N/A
16	FLOAT	lat1	Target Latitude	degrees
20	FLOAT	lon1	Target Longitude	degrees
24	UCHAR	mode	Mode	0 or 1

Figure 33. Ground Locomotion System Pass-through packet.

2. LabVIEW Application

There are many applications that could be used to send data packets to the Virtual Cockpit. Procerus created an open source Visual C++ application that issues some of the common interface commands, such as a pass-through packet creator. This application worked for testing the interface and for confirming that the packets were received by the Gumstix. The difficulty of this application was that in order to send data from pass-through packet creator, it needed to be written in hexadecimal. The code could be modified to suit the needs of the ground locomotion system. This would require a good working knowledge of Visual C++. This is what led to the decision to develop an interface application in LabVIEW. LabVIEW uses graphical programming that is easy to use.

The initial step in building this LabVIEW application was to understand the packet format shown in Figure 33. This is the building block of the application. Starting from byte zero, the bytes were built from top to bottom in the block diagram. Numeric controls were used for the variables and numeric constant blocks were used for the constants. A Boolean toggle switch was implemented to switch between manual and autonomous mode. Autonomous mode is defined as true (mode 1) and manual mode is false (mode 0). Next, these blocks where converted to a string of binary values using the little-endian byte format. Then the element strings are concatenated into one string of bits. Now the packet is built and ready to be sent to Virtual Cockpit's TCP/IP port (5005).

LabVIEW has several functions for TCP communications. Three of these functions are used to write the packet data. They are the TCP open connection, TCP write, and TCP close connection functions. All three of these functions are linked with error-to-warning connections that pass error information to the screen. The front panel and block diagram are shown below in Figure 34. The testing of this application will be discussed below.

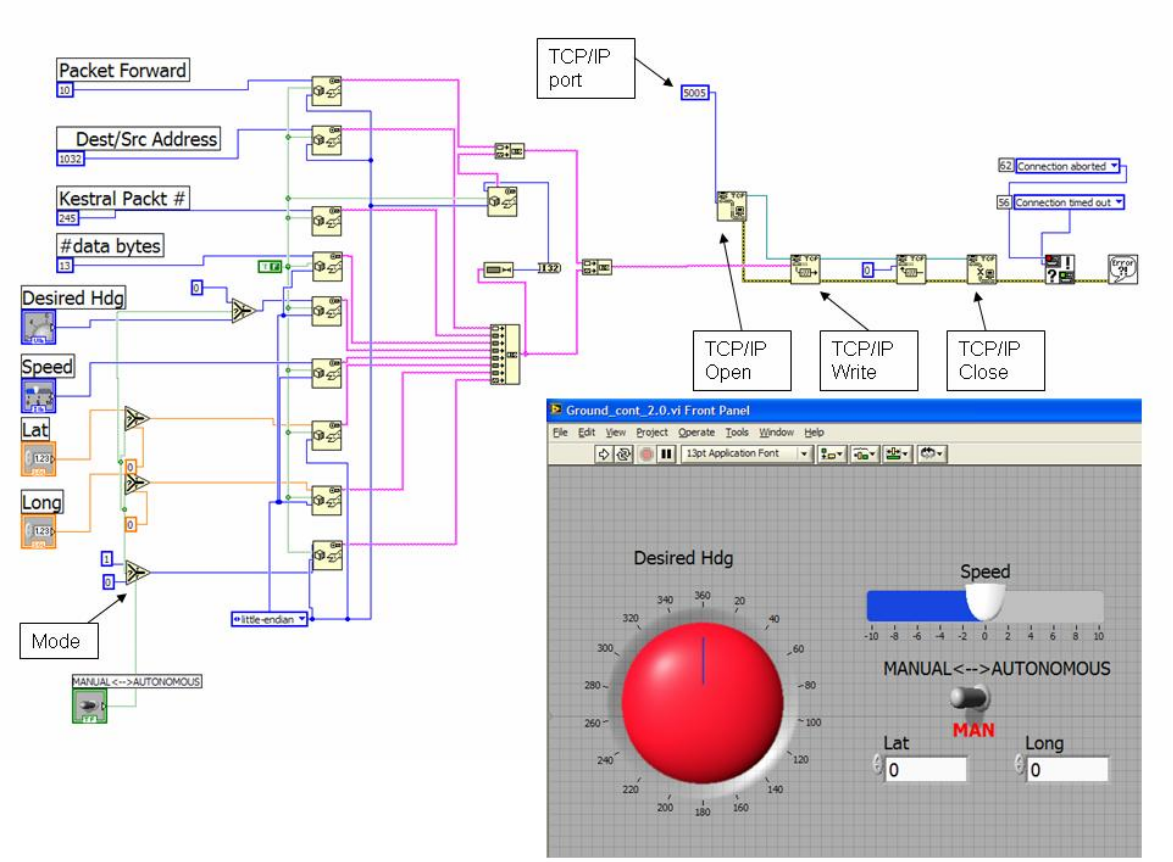

Figure 34. LabVIEW TCP/IP Application Front Panel and Block Diagram.

D. GUMSTIX AND AUTOPILOT SOFTWARE AND APPLICATION TESTING

This section discusses the testing and integration of the Gumstix example code and the LabVIEW application. The code was modified to request and receive the GPS telemetry packet and also receive the modem mirror bridge packet (245). The code was first tested on a Linux machine. Once the code was receiving the packets from both the autopilot and LabVIEW application it was built and transferred to the Gumstix.

The code was built on the Linux machine using make file. In order the build the code on the Gumstix with bitbake, it needed to use the recipe format for using a make file with local sources. This was explained in Chapter III. To accomplish this, all the source files of the code needed to be saved in the package's file directory. The bitbake recipe

will also have all the files listed in it's source file line including the make file. When the code was functioning properly on the Gumstix, it could now be integrated with the drive system code.

E. SUMMARY

This chapter described the integration and testing of the Gumstix and KAP for the purpose of the ground locomotion system. The KAP provides the ground locomotion system with sensor measurements from the autopilot and control data from the ground station. Now the information is available to be processed into control movements of the WHEGs™. This is done by integrating the modified Gumstix code and the drive system code from Chapter IV. Chapter VI will describe this integration and the complete final ground locomotion configuration.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. MMALV GROUND LOCOMOTION SYSTEM INTEGRATION AND TESTING

A. CHAPTER OVERVIEW

This chapter will discuss the integration of the drive system, Gumstix and Kestrel Autopilot System in order to assemble the complete ground locomotion system. The chapter will also discuss the final configuration and testing of the ground locomotion system.

B. GROUND LOCOMOTION SYSTEM INTEGRATION

By this point, the drive system and the KAP are functioning properly with the Gumstix, but not mutually. To accomplish this, the drive system algorithm code is inserted into the main() function in the modified Gumstix example code. The main() function contains an infinite loop that will continuously calculate the drive system control algorithm. As mentioned in the previous chapter the combined code will have global variables, which are shown in Figure 35. The combined code is shown in the Appendix.

Description	**Variable Name**	**Source**
Target latitude and longitude	lat1, lon1	Ground Station
MMALV latitude and longitude	lat2, lon2	Autopilot
Autopilot heading	Heading_Autopilot	Autopilot
Desired heading	Heading	Ground Station
Speed	speed	Ground Station
Autonomous or manual mode	Mode	Ground Station

Figure 35. Global Variables.

With the drive system control algorithm code inserted, the ground locomotion system will now function in manual mode, which is following a desired heading. If mode is equal to zero the heading error is determined from the difference of the autopilots heading and the desired heading sent from the ground station.

To operate the system in autonomous mode, the bearing and distance must be calculated from the MMALV current location and target location. The bearing and distance calculation are shown in Equations 4 and 5 [16]. These equations will only be calculated if mode equals one or TRUE.

$$\text{bearing} = \text{atan2}\left(\sin\left(lon1 - lon2\right)\cos\left(lat1\right), \cos\left(lat2\right)\sin\left(lat1\right) - \sin\left(lat2\right)\cos\left(lat1\right)\cos\left(lon1 - lon2\right)\right) \quad (4)$$

$$\text{distance} = \text{acos}\left(\sin\left(lat2\right)\sin\left(lat1\right) + \cos\left(lat2\right)\cos\left(lat1\right)\cos\left(lon1 - lon2\right)\right) \times Radius_{earth} \quad (5)$$

The error will be calculated from the difference of the autopilots heading and the result of Equation 4. Once this composite ground locomotion system code was complete and functioning properly, it was tested with the actual ground locomotion system components that will be used in the MMALV.

C. GROUND LOCOMOTION SYSTEM FINAL CONFIGURATION

The composite code described in the previous section was initially tested on the development components. This section will describe the testing and configuration of the actual components that will be used in the MMALV ground locomotion system. The components that are used are the following: Gumstix with breakout-vx, Pololu dual micro serial motor controller, and SolarBotics motors with clutch system. Merging these components didn't require any code modifications, only some initial configuring. The motor controller needed to be configured to insure it had a two motor setup, with the right motor numbered as '0' and the left number as '1'. This configuration is explained in Chapter IV.

The Gumstix needed to be configured with the breakout-vx expansion board. The initial testing involving the servo and motor configurations was done using the console-vx expansion board for the Gumstix. This expansion board was used because of its mini

DIN8 connections which allowed easy interface with the host computer. It also allowed the monitoring of the development process and observation of the data transfer between the ground locomotion system components. The breakout-vx is used in the final design configuration due to its smaller size and battery connections. These connections are shown below in Figure 36.

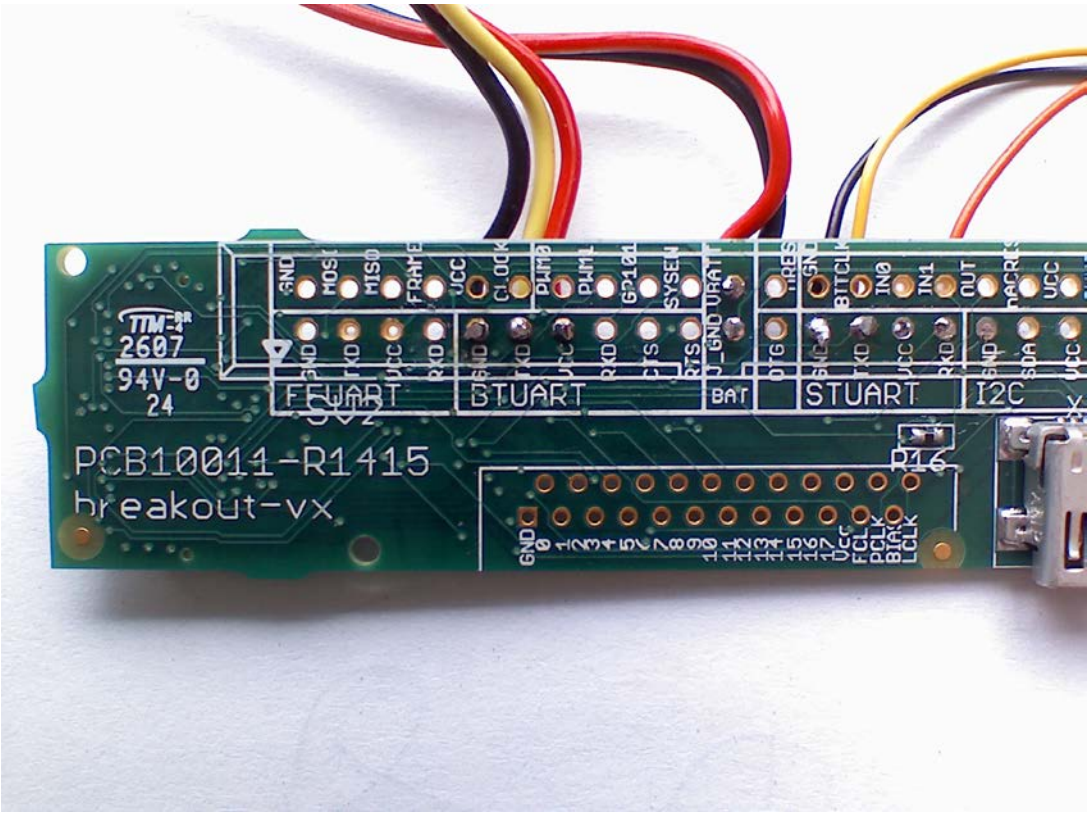

Figure 36. Breakout-vx Connections.

The ground locomotion system also needed to function without the serial connection of the host computer. To accomplish this, a code execution script was added to the Gumstix's boot initialization process at the default run level 5. This enables the ground locomotion system to start once battery power is applied to the Gumstix and after 95 % of its boot sequence. The directory listing of the run level 5 scripts and the ground locomotion execution script are shown in Figure 37. The highlighted line is the ground locomotion execution script (S95grdctl). Once this was accomplished the ground

locomotion system can operate on battery power and can be controlled through the LabVIEW application described in Chapter VI. The testing of this final configuration is described in the next section.

```
root@gumstix-custom-verdex:/etc/rc5.d$ ls -l
lrwxrwxrwx    1 root     root  18 Jan  1 00:00 S10dropbear -> ../init.d/dropbear
lrwxrwxrwx    1 root     root  13 Jan  1 00:00 S20boa -> ../init.d/boa
lrwxrwxrwx    1 root     root  16 Jan  1  1970 S20dbus-1 -> ../init.d/dbus-1
lrwxrwxrwx    1 root     root  14 Jan  1 00:00 S20ntpd -> ../init.d/ntpd
lrwxrwxrwx    1 root     root  16 Jan  1 00:00 S20syslog -> ../init.d/syslog
lrwxrwxrwx    1 root     root  21 Jan  1  1970 S23bluetooth -> /etc/init.d/bluetooth
lrwxrwxrwx    1 root     root  20 Jan  1 00:00 S25alsa-state -> ../init.d/alsa-state
lrwxrwxrwx    1 root     root  17 Jan  1  1970 S30ntpdate -> ../init.d/ntpdate
lrwxrwxrwx    1 root     root  17 Jan  1 00:00 S50bonjour -> ../init.d/bonjour
lrwxrwxrwx    1 root     root  14 Jan  1 00:00 S65cron -> ../init.d/cron
lrwxrwxrwx    1 root     root  13 Jan  1 00:00 S90i2c -> ../init.d/i2c
lrwxrwxrwx    1 root     root  18 Jan  1 00:05 S95grdctl -> /etc/init.d/grdctl
lrwxrwxrwx    1 root     root  19 Jan  1 00:00 S99rmnologin -> ../init.d/rmnologin
root@gumstix-custom-verdex:/etc/rc5.d$ cat S95grdctl
#!/bin/sh

echo -n "Starting Ground Control... "
/usr/bin/GrnCntl > /dev/null 2>$1 &
echo "done."
```

Figure 37. Gumstix Ground Control Initialization Script.

D. GROUND LOCOMOTION SYSTEM TESTING

The testing of the complete ground system was done with the final configuration describe above. The layout of the final design configuration is shown below in Figure 38.

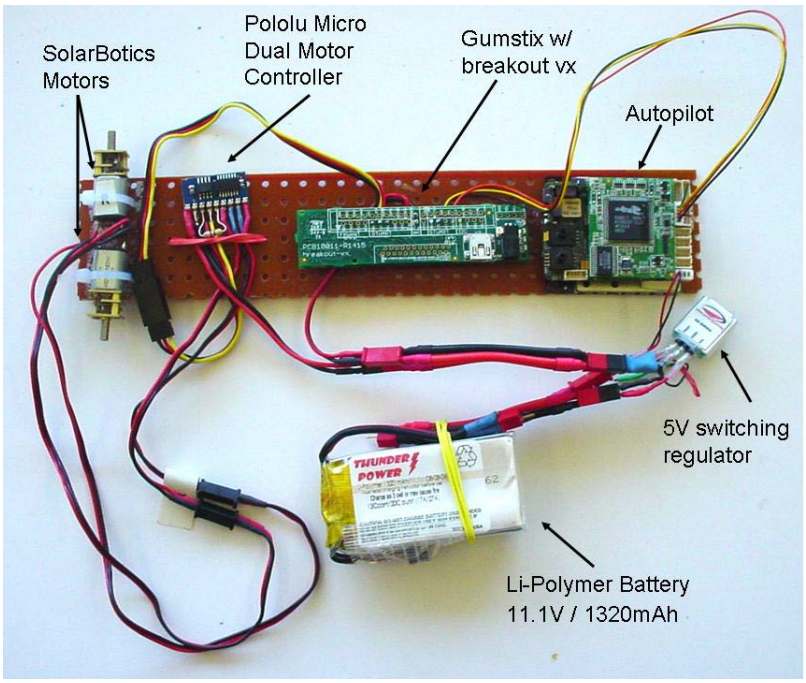

Figure 38. Final Ground Locomotion System Configuration.

The battery operates all components of the MMALV, including the ground locomotion system. The battery is an 11.1V lithium polymer and operates at 1320 mAh. A 5V switching voltage regulator is used to power the Gumstix, motor controller, and WHEG™ motors.

The ground locomotion system can be controlled once the Gumstix runs the code execution script and power is applied to all components. The initial state of the system is all of the global variables are zeroed. The MMALV will remain stationary until data packets are received from the ground station. As mentioned previously, the ground locomotion system will respond to either a desired heading or target location input from the LabView application at the ground station. Stationary testing was done by observation of the proper motor rotation rates. (These two modes were tested with success and described below.)

Manual mode controls the MMALV to follow a specific compass heading or bearing. This is done by dialing in the desired heading and speed on the LabVIEW application. The system responds by turning towards the desired heading. One motor

would rotate at the set desired speed the other motor would rotate at the rate proportional to the heading error. Therefore, for a left turn the left motor will slow to a rotation rate proportional to the error. When the error is zero both motors rotate at the same rate.

Autonomous mode navigates the MMALV to a desired target latitude and longitude. This was done by sending the target location and desired speed from the LabVIEW application. The ground control algorithm uses the data to compute the desired heading to the target location. The MMALV will continuously navigate to the target location and constantly update the desired bearing to target. It tracks this heading in the same manner as it did in manual mode. This mode was tested stationary by observing the correct bearing and distance calculations to a specific target location, and also observing the correct motor rotation rates. Additionally, once the MMALV is within a one meter range of the target location, the speed is set to zero.

All of this testing was done with the configuration in Figure 38. The testing was only done stationary due to the deficiency of new 16 inch wingspan MMALV. The desire was to test the system using this airframe with the integrated clutch system and WHEGs™, but due to manufacturing delays it was not accomplished. These test results do validate that this system will function as designed when integrated with the new MMALV airframe. The conceptual ground locomotion system integrated into the 16 inch MMALV airframe is shown in Figure 39.

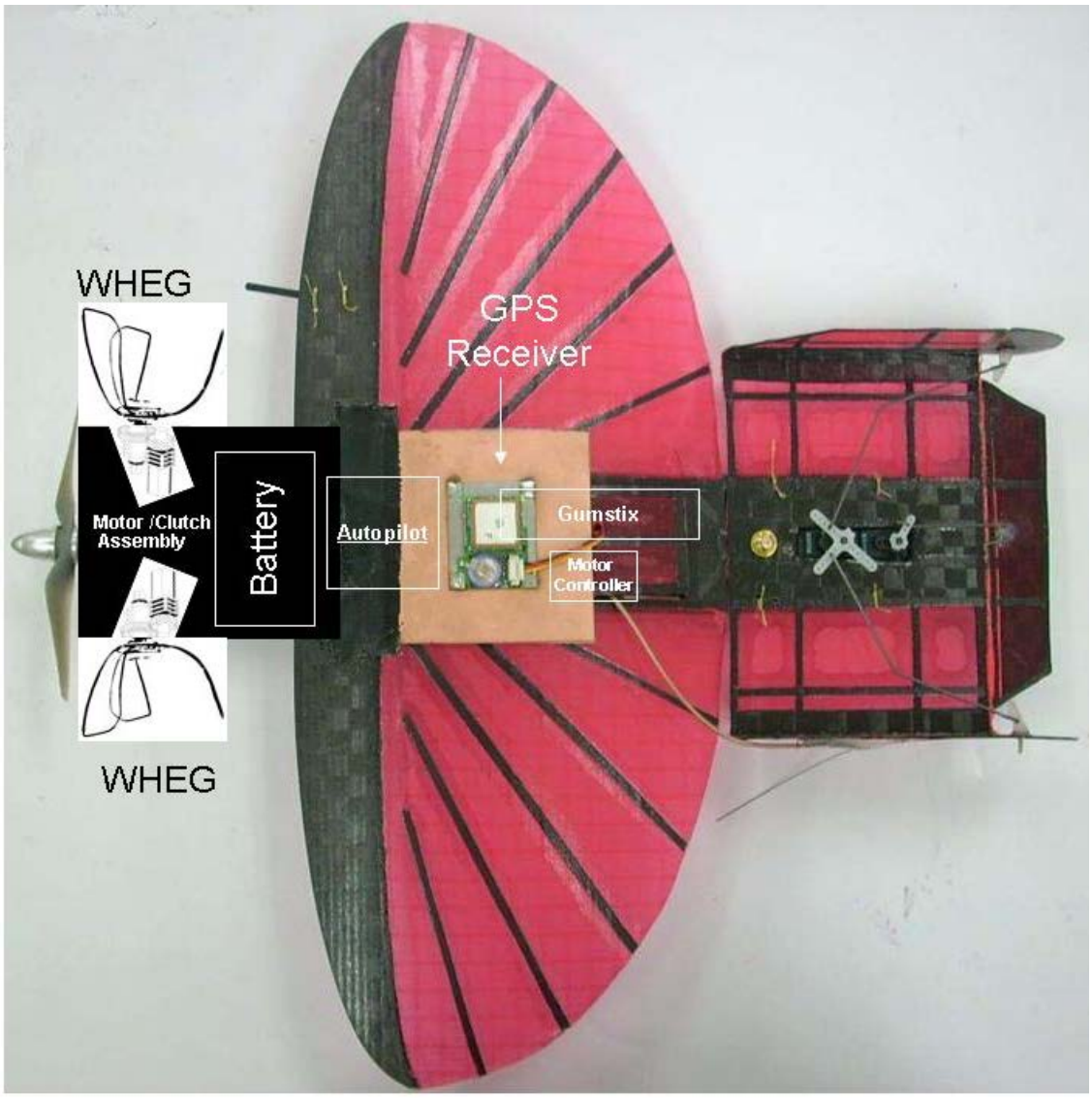

Figure 39. Conceptual MMALV Ground Locomotion Configuration [After 3].

THIS PAGE INTENTIONALLY LEFT BLANK

VII. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSION

This thesis has significantly advanced the ground locomotion mode of the MMALV as well as expanded its capabilities. The ground locomotion system design provides the MMALV a method of easily navigating the battlefield to provide the warfighter with a highly mobile sensor and reconnaissance platform. The Gumstix can also provide the MMALV with processing power to collect sensor data and then send the data through the autopilot to the ground station.

The ground locomotion system design presented in this thesis can be easily modified and refined to suit the needs of the MMALV. This is done without any modification of the Kestrel autopilot firmware. Now the ground work laid to bring the MMALV from less of a concept and more a reality.

B. RECOMMENDATIONS

Future thesis work can expand and refine the ground locomotion design in this thesis. Dynamic testing still needs to be completed with the new MMALV airframe. We still don't know how well the MMALV will track the desired heading. Also, the accuracy of the ground GPS navigation has not been completed. The ground locomotion system needs to be flight tested. Data relay will also be needed for the system to work. The data being sent from the ground station and MMALV will need to be routed from an aerial platform due to loss of LOS. Initial research has already been conducted [3].

Additional work can be conducted by expanding the capabilities of the Gumstix. There are many different expansion boards that could be used with the motherboard to do this. The Gumstix could be used to control multiple cameras. It could also be used to collect sensor data. Additionally, the Gumstix could be outfitted with an additional transmitter to function as a wireless sensor node and become part of an ad hoc wireless network. The Gumstix's potential is relatively infinite.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX: GROUND LOCOMOTION SYSTEM C CODE

```
#include <stdio.h>
#include "CommSerialLinux.h"
#include <math.h>
#include <unistd.h>
#include <termios.h>
#include <fcntl.h>
#include <errno.h>

CCommSerialLinux g_CommSerial;
typedef void (*CallBackFuncPtr)(CCommPacket *, PacketType);
pthread_t RequestTelemetryThread;
int Heading,speed,Mode;
float lat1, lon1, lat2, lon2, Heading_Autopilot; //Global Variables

// This is a function that gets called every time a packet from the
autopilot is received
void ReceivePacket(CCommPacket *NewPkt, PacketType Type)
{
      //Casting variables
      CStdTelemPacket *StdTelemPacket;
      CGPSTelemPacket *GPSTelemPacket;
      //Simple error checking
      if(NewPkt == NULL) return;
      //printf("Type =%d",Type);
      switch(Type)
      {
      case STD_TELEMETRY_PACKET:
            StdTelemPacket =
reinterpret_cast<CStdTelemPacket*>(NewPkt);
            Heading_Autopilot = StdTelemPacket->GetHeading()*57.3;
      break;
      case GPS_TELEMETRY_PACKET:
            GPSTelemPacket =
reinterpret_cast<CGPSTelemPacket*>(NewPkt);
            lat2 = GPSTelemPacket->GetGPSLatitude();
            lon2 = GPSTelemPacket->GetGPSLongitude();
      break;
      case 245:
            Heading =   NewPkt-> ReadUnsignedShort(7);
            speed =     NewPkt-> ReadShort(9);
            lat1 =      NewPkt-> ReadFloat(11);
            lon1 =      NewPkt-> ReadFloat(15);
            Mode =      NewPkt->ReadUnsignedChar(19);
      }
}
void *RequestTelemetry(void* parm)
{
      int counter = 0;
      //Create our standard telem request packet
      while(1)
      {
```

```
            //Request the standard telemetry 10 times a second
            CWriteRawPacket
ReqStdTelemPkt(STD_TELEMETRY_REQUEST_PACKET, 0xFFFF);
            ReqStdTelemPkt.SetPacketIDNum(counter++);
            g_CommSerial.SendRaw(&ReqStdTelemPkt);
            usleep(1000000); //Sleep for .1 seconds

            //Request the GPS telemetry
            CWriteRawPacket
ReqGPSTelemPkt(GPS_TELEMETRY_REQUEST_PACKET, 0xFFFF);
            ReqGPSTelemPkt.SetPacketIDNum(counter++);
            g_CommSerial.SendRaw(&ReqGPSTelemPkt);
            usleep(1000000); //Sleep for .1 seconds
      }
      return NULL;
}
int main()
{
      //Register a callback for the packets
      g_CommSerial.RegisterCallBack(ReceivePacket, ALL_PACKET);

      //Open the serial port on the gumstix
      if(!g_CommSerial.Open("/dev/ttyS2")) return -1;

      if (0 != pthread_create(&RequestTelemetryThread, NULL,
RequestTelemetry, NULL))
      {
            printf("Could not create request telemetry thread.
Exiting\n");
            return -1;
      }
      while(1)
      {
            int fd, nbytes;
            struct termios options;
            char buf[1024];
            char cmd[6];
            //Open the commport and parameters
            fd =open("/dev/ttyS1", O_RDWR | O_NOCTTY | O_NDELAY);

            if (fd == -1) {
                  perror("Could not open port! - ");
                  return 1;
            } else {
                  fcntl(fd, F_SETFL, FNDELAY);
            }

            tcgetattr(fd, &options);
            cfsetispeed(&options, B9600);
            cfsetospeed(&options, B9600);
            options.c_cflag |= (CLOCAL | CREAD);
            options.c_cflag &= ~PARENB;//No parity
            options.c_cflag &= ~CSTOPB;//No Stop Bit
            options.c_cflag &= ~CSIZE;
            options.c_cflag |= CS8;//8 bits at a time
```

```
            options.c_cflag &= ~CRTSCTS;
            options.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);
            options.c_iflag &= ~(IXON | IXOFF | IXANY);
            options.c_oflag &= ~OPOST;
            tcsetattr(fd, TCSANOW, &options);

                  float R = 6378.7;//radius of earth
                  float tc1,tc2,error,lat_t,lon_t,lat_a,lon_a;
                  int direction1=0,direction2=2;

                  if(Mode == TRUE){//bearing to target calculation
                        lat_t = lat1*M_PI/180;
                        lon_t = lon1*M_PI/180;
                        lat_a = lat2*M_PI/180;
                        lon_a= lon2*M_PI/180;
                        tc1=atan2(sin(lon_t-
lon_a)*cos(lat_t),cos(lat_a)*sin(lat_t)-
sin(lat_a)*cos(lat_t)*cos(lon_t-lon_a));
                        tc2 = tc1*180/M_PI;
                        float theta = fmodf(tc2+360, 360);
                        float D =
R*acos(sin(lat_a)*sin(lat_t)+cos(lat_a)*cos(lat_t)*cos(lon_t - lon_a));
                        error = Heading_Autopilot-theta;
                        if(D*1000 <= 1){
                              speed = 0;
                        }
                  }
                  else if(Mode ==FALSE){
                        error = Heading_Autopilot-Heading;
                  }
                        float r_rate=1.27;  //full speed*10
                        float l_rate=1.27;

                        if (error<-180){
                              error = error+360;
                        }
                        if (error>180){
                              error = error-360;
                        }
                        printf("Error = %f.\n",error);

                        if (error<0){
                              r_rate = 1.27 + error/180;
                              }
                        if (error>0){
                              l_rate = 1.27 - error/180;
                        }
                        printf("%f, %f\n",r_rate,l_rate);
                        float right = r_rate*abs(speed)*10;
                        float left = l_rate*abs(speed)*10;
                        int rgt = right;
                        int lft = left;


                        if (speed<0){           //reverse for motors
```

```
                              direction1 = 1;
                              direction2 = 3;
                        }

// 4 byte motor commands

  bzero(cmd, 4);
  cmd[0] = 128;         /* Unit ID */
  cmd[1] = 0;           /* Unit ID */
  cmd[2] = direction1;  /* Right Motor 0 = fwd, 1 = rvs */
  cmd[3] = rgt;   /* Speed 0 - 127 */

 write(fd, cmd, 4);

  bzero(cmd, 4);
  cmd[0] = 128;         /* Unit ID */
  cmd[1] = 0;           /* Unit ID */
  cmd[2] = direction2;  /* Left Motor 2 = fwd, 3 = rvs */
  cmd[3] = lft;   /* Speed 0 - 127 */

  write(fd, cmd, 4);

        close(fd);

      }

      return 0;
}
```

LIST OF REFERENCES

[1] R. J. Bachmann, R. D. Quinn, R. Vaidyanathan, and J. Zupan, "A Morphing Micro Air-Land Vehicle," presented as a technical proposal to the United States Air Force, Eglin AFB, FL, 2004. For a copy, contact author at rvaidyan@nps.edu.

[2] J. Kiihne, "Autonomous Flight and Ground Locomotion for a Biologically Inspired Micro Morphing Air-Land Vehicle (MMALV)," MS Thesis, Naval Postgraduate School, September 2006.

[3] R. Bledsoe, "ZigBee/802.15.4Applied to a Biologically Inspired Micro Morphing Air-Land Vehicle (MMALV) and MiniWhegs Vehicles for Autonomous Control and Ad-hocNetworking," MS Thesis, Naval Postgraduate School, June 2007.

[4] B. Pater, "Improved Airframe Design and Altitude Measurement for a Biologically Inspired Micro Morphing Air-Land Vehicle (MMALV)," MS Thesis, Naval Postgraduate School, June 2008.

[5] Procerus Technologies, "Kestrel Users Guide," September 2006.

[6] Procerus Technologies, "Application Note 101: Interfacing the Kestrel Autopilot to a Gumstix SBC," 2008.

[7] Gumstix Inc., "Home Page," [http://www.gumstix.com/index.html] (last reviewed on June 5, 2008).

[8] Gumstix Developers Site, "Getting Started" [http://www.gumstix.net/Software/cat/Getting-started/111.html] (last reviewed on June 5, 2008).

[9] Gumstix Developers Site, "Build System Overview," [http://www.gumstix.net/Software/view/Build-system-overview/Hello-world-tutorial/111.html] (last reviewed on June 5, 2008).

[10] Pololu, "Micro serial servo controller user's guide", (last reviewed on June 5, 2008).

[11] Pololu, "Low-voltage dual serial motor controller user's guide," (last reviewed on June 5, 2008).

[12] Pololu, "Micro Dual Serial Motor Controller SMC02B User's Guide," 2004.

[13] Pololu, "Solarbotics Gearboxes," [http://www.pololu.com/catalog/category/40] (last reviewed on June 5, 2008).

[14] R. J. Bachmann, Re: clutch schematic, Email available: from richard.bachmann@gmail.com.

[15] Procerus Technologies, “Communications Protocol Communications Protocol Version 6,” October 2007.

[16] Institute for Geophysics, “Calculate Distance, Bearing and More Between Two Latitude/Longitude Points,” [http://www.ig.utexas.edu/outreach/googleearth/latlong.html] (last reviewed on June 5, 2008).

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Fort Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Marine Corps Representative
   Naval Postgraduate School
   Monterey, California

4. Professor Xiaoping Yun
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

5. Professor Ravi Vaidyanathan
   Department of Systems Engineering
   Naval Postgraduate School
   Monterey, California

6. David E. Green, SL, USMC
   Chief Technology Advisor, C4
   Headquarters Marine Corps
   Washington, DC

7. Director, Training and Education, MCCDC, Code C46
   Quantico, Virginia

8. Director, Marine Corps Research Center, MCCDC, Code C40RC
   Quantico, Virginia

9. Jayson Durham
   Science and Technology, C4I
   SPAWAR SSC, San Diego