2009-06

Characterization of robotic tail orientation as a function of platform position for surf-zone robots

Holland, Courtney L.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

CHARACTERIZATION OF ROBOTIC TAIL ORIENTATION AS A FUNCTION OF PLATFORM POSITION FOR SURF-ZONE ROBOTS

by

Courtney L. Holland

June 2009

Thesis Advisor: Richard Harkins
Second Reader: Peter Crooker

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

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

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 2009	Master's Thesis

4. TITLE AND SUBTITLE: Characterization of Robotic Tail Orientation as a Function of Platform Position for Surf-Zone Robots

5. FUNDING NUMBERS

6. AUTHOR(S) Courtney L. Holland

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

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

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

12b. DISTRIBUTION CODE
A

13. ABSTRACT *(maximum 200 words)* The Naval Postgraduate School Small Robot Initiative is an ongoing effort to develop autonomous robotic platforms for military applications. The latest design in this series, a quadruped robot with a tail for stability and obstacle climbing, is currently under development in collaboration with Case Western Reserve University. Tail orientation as a function of robot platform attitude is tested for angle of bank climbs at 10 and 15 degrees. Data indicate that although the platform induced noise is significant, tail orientation can be successfully managed with proper PID feedback mechanisms, including tail position as a function of platform attitude. Gross control of the tail used as an assist for climbing is validated in this experiment. More sophisticated filter algorithms are indicated for fine tuned tail control, including but not limited to the Kalman filter.

14. SUBJECT TERMS Amphibious, Autonomous, Robotics, WHEGS

15. NUMBER OF PAGES
103

16. PRICE CODE

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

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18-298-

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

CHARACTERIZATION OF ROBOTIC TAIL ORIENTATION AS A FUNCTION OF PLATFORM POSITION FOR SURF-ZONE ROBOTS

Courtney L. Holland
Lieutenant, United States Navy
B.S., United States Naval Academy, 2002

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN APPLIED PHYSICS

from the

NAVAL POSTGRADUATE SCHOOL
June 2009

Author: Courtney L. Holland

Approved by: Richard Harkins
Thesis Advisor

Peter Crooker
Second Reader

James Luscombe
Chairman, Department of Applied Physics

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The Naval Postgraduate School Small Robot Initiative is an ongoing effort to develop autonomous robotic platforms for military applications. The latest design in this series, a quadruped robot with a tail for stability and obstacle climbing, is currently under development in collaboration with Case Western Reserve University. Tail orientation as a function of robot platform attitude is tested for angle of bank climbs at 10 and 15 degrees. Data indicate that although the platform induced noise is significant, tail orientation can be successfully managed with proper PID feedback mechanisms, including tail position as a function of platform attitude. Gross control of the tail used as an assist for climbing is validated in this experiment. More sophisticated filter algorithms are indicated for fine tuned tail control, including but not limited to the Kalman filter.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

Thanks to Professor Harkins, my thesis advisor, for keeping me on the right track, even if I didn’t know where I was headed. Thanks to Professor Crooker for volunteering to help, and for teaching me pretty much everything I know about electronics. Thanks to George Jaksha, without whom, I would have gotten absolutely nowhere. I’d like to thank Sam for the caffeine, and Keith for the helpful hints. And, I’d also like to thank my wife, Ania, for putting up with me while I have been MIA these last few months.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. NPS BACKGROUND

The Naval Postgraduate School Small Robot Initiative (SMART) is an ongoing effort of the Combat Systems Science and Technology Department to develop autonomous robots with potential military applications. Utilizing Commercial Off-the-Shelf (COTS) components, this program seeks to design, build, and operate cost effective robots that are highly mobile, can perform waypoint navigation and dead reckoning, conduct obstacle avoidance, and support a variety of different mechanisms to perceive and interact with their environment.

There is significant interest in developing robots for operation in rugged, unstructured environments. In particular, the ability to deploy autonomous robots in beach, surf-zone, and near-shore environments is highly sought after for both civilian and military applications. Potential applications include the ability to conduct coastal surveys, covert surveillance, mine identification, and mine clearance operations. However, this environment is extraordinarily challenging for a robot to operate in and no current robotic applications have been successfully fielded. The robot would need to be robust enough to navigate a varied and changing environment to include soft and shifting sands, the movement of the tides and waves, and an uneven surface with few level areas. In addition to this, the robot would need to be rugged enough to withstand water to a certain depth, salt water corrosion, and various kinetic stresses from operating in a real world environment.

B. WHEGS PLATFORM DEVELOPMENT

1. Biologically Inspired Movement

Whegs™ (wheel-legs) is a class of robot developed using basic mechanical designs that were inspired by the locomotive principles of biological organisms. Roger Quinn of Case Western Reserve University developed this method for utilization in the university's Biologically Inspired Robotics Laboratory. The motion of Whegs™ concept was inspired by the mobility of hexapods, specifically the cockroach, to take advantage of the insect's inherent stability and mobility over a variety of terrain. The principals of cockroach motion are that it stands and moves using six legs. It walks using a tripod gait, wherein the front and rear legs of the body move in phase with the middle leg of the opposite side of the body. When it encounters a large obstacle, its gait changes by engaging all of its Whegs™ simultaneously to surmount the obstacle [1]. It also flexes its body joint to permit greater vertical reach for its legs and prevents high centering off the obstacle during a climb by bending the front half of its body down.

2. Early WHEGS™ Designs

Whegs™ utilize a unique wheel-leg design to take advantage of both the enhanced mobility of legged platforms and the simplicity of wheel driven robots. The Whegs™ is a simple design that utilizes COTS resources to greatly increase mobility over normal wheeled robots without the need for complicated low level control algorithms. The standard Whegs™ wheel consists of symmetrically spaced spokes attached to a central hub. Passive mechanical adaptation is incorporated through the use of torsional compliance devices in the axles, permitting the opposing Whegs™ to engage for additional torque to surmount significant obstacles. After the obstacle has been surmounted, the opposing Whegs™ will automatically return to its normal gait for continued translation.

The previous Whegs™ design used by the NPS SMART program was the Dayton Area Graduate Studies Institute (DAGSI) Whegs™ prototype called Agbot. The Agbot platform was built as a collaboration between NPS and Case Western Reserve University Biologically Inspired Robots Laboratory. Agbot, pictured in Figure 1, is a six-legged robot with a tripod gait. It utilizes one drive motor to move the Whegs™, which are mechanically linked through a chain and gear system. Passive compliance on Agbot is performed by a limited slip differential which consists of two coaxial axles linked by a spring. Steering is accomplished by turning the front and back leg sets inboard in opposition, accounting for a large turning radius. Agbot did not have body joint flexion incorporated, but the chasis was built in two halves with a body joint. An in-depth review of Agbot can be found in [2].

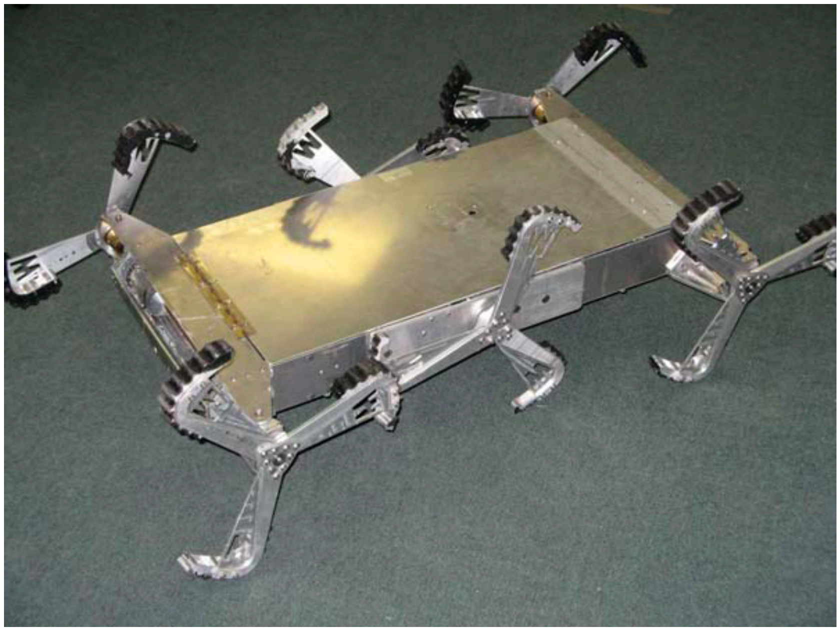

Figure 1. Agbot (From [2])

3. Pelican Whegs™ Development

The Pelican Whegs is the latest design effort under this initiative. Alexander Boxerbaum of Case Western Reserve University is building the Pelican Whegs body as part of the ongoing examination of Whegs-based robotic applications.

The Pelican Whegs™ platform replaces the hexapod locomotion and body joint flexion of the DAGSI Whegs™ with quadruped locomotion and a motorized tail as seen in Figure 2. This robot moves using a diagonal gait instead of the more stable tripod gait of the earlier model. To improve stability, the three spoke wheel-legs have been replaced by four evenly spaced spoke wheel-legs to reduce body roll and vertical translation during motion. The right and left side Whegs™ are driven using separate drive motors, and each set is linked mechanically. This permits Pelican Whegs™ to take advantage of the differential, tank steering common to most wheeled robots. The passive torsionally compliant devices will be incorporated into enclosures in the individual Whegs™ hubs.

The new tail mechanism is intended to perform several functions for the Pelican Whegs™. During normal operation, the tail can be lowered to provide stability during transit across uneven terrain to reduce undesirable body tilt and roll. When the robot seeks to mount an obstacle, the tail will be lowered to provide additional motive force to help boost the robot. It also replaces the body flexion function of the DAGSI Whegs™ by acting as a foot during climbing, preventing the robot from high-centering and falling back off of obstacles [4].

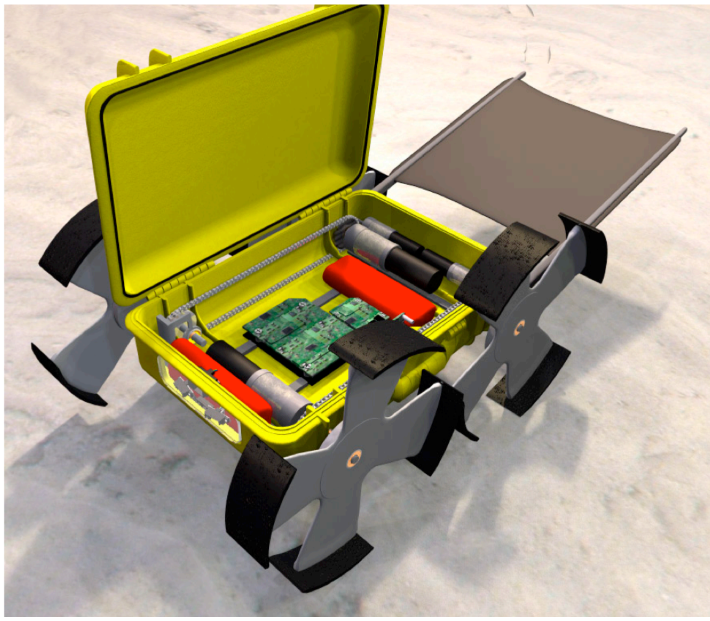

Figure 2. Pelican Whegs™ (From [4])

C. THESIS CONCEPT

A robotic platform named Robster, seen in Figure 3 and Figure 4, was built to emulate the basic functional design considerations and components of the Pelican Whegs™ robot. This thesis is determined to develop control and logic algorithms to operate the tail mechanism implementation in the new Pelican Whegs™ design and evaluate the efficacy of the new design. Passive compliance devices were not built for the Robster due to their complexity and poor reliability and are not the focus of this thesis.

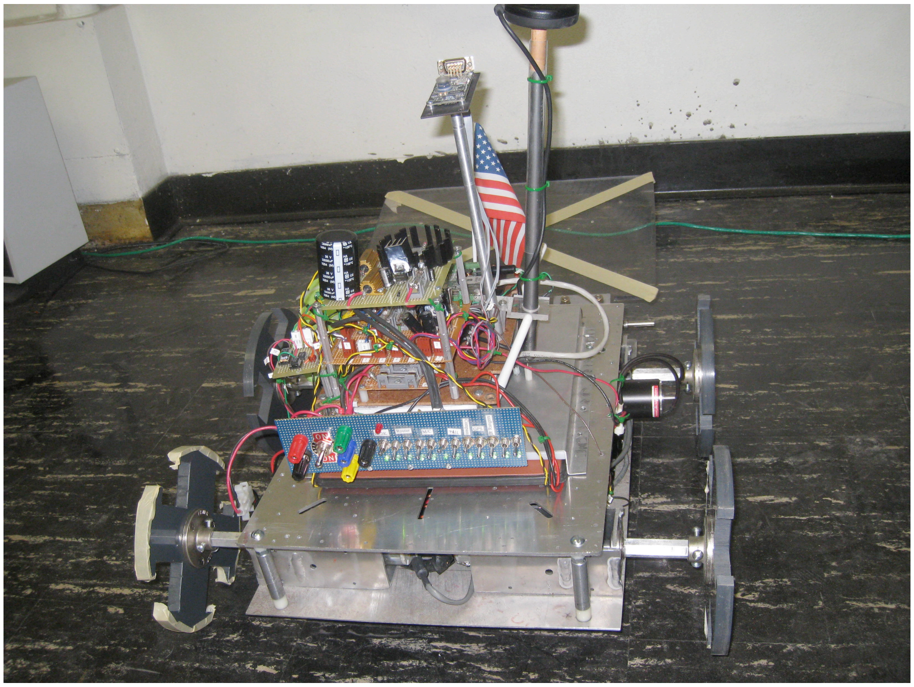

Figure 3. ROBSTER, front view

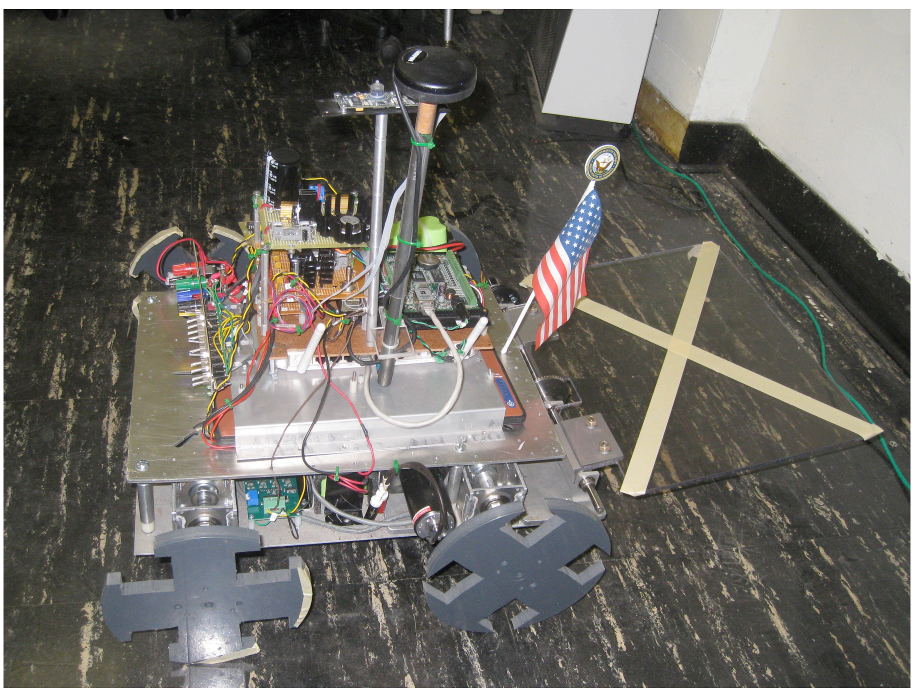

Figure 4. ROBSTER, side view

THIS PAGE INTENTIONALLY LEFT BLANK

II. ROBOT DESIGN

A. MECHANICAL COMPONENTS

1. Platform

The Robster chasis and platform components were chosen to match the functional capabilities of the Pelican Whegs™ chasis. The four drive motors, motor controllers, motor battery, and tail assembly are mounted on a $13''$ wide by $18''$ long $\frac{1}{16}''$ thick aluminum plate. The tail assembly gears, shafts, motor, and paddle are attached to a separate removable platform connected to the base. Above this, a separate aluminum plate is connected using 3” extensions to provide the base for the electronics. This plate also shields sensitive electronic components from large electromagnetic interference generated during drive motor activation. The Robster weighs 32lbs, primarily due to the four drive motors and the tail motor. This weight was much greater than the expected weight of the Pelican Whegs™ and affected some of the results of the experiment.

2. Wheel Legs

Robster has four wheel-legs machined from single pieces of hard, polyvinyl chloride plastic pictured in Figure 5. Each Wheg™ consists of four, 3.5” long spokes around a central hub, evenly separated by 90°. The four-spoke leg has advantages and disadvantages compared to a conventional wheel that are common to all Whegs™ (see Figure 6). In a conventional wheel, climbing ability is limited by the radius of the wheel, no matter how great the traction of the wheel. The Pelican Whegs™ can get a foothold for an obstacle height given by

$$h_1 = 2\left[r \cdot \cos(\theta)\right] = 2\left[(3.5'') \cdot \cos\left(45^\circ\right)\right] = 5.23'' \qquad (1)$$

for a 29.3% greater climbing ability than a wheel. The Pelican Whegs™ also has an advantage over the previous DAGSI Whegs™ model in the vertical translation of its Whegs™ hub during motion, which is given by

$$h_2 = r\left[1 - \cos\left(\tfrac{\theta}{2}\right)\right] = (3.5'')\left[1 - \cos\left(\tfrac{45^\circ}{2}\right)\right] = 0.266'' \qquad (2)$$

Vertical translation is only 7.6% of hub height, which is also 43% less than the 13% of hub height achieved by the three-spoked DAGSI Whegs™. Consequently, the Pelican Whegs™ enjoys a much steadier ride. This advantage helps to offset the reduced stability of the diagonal vs. tripod gait [1].

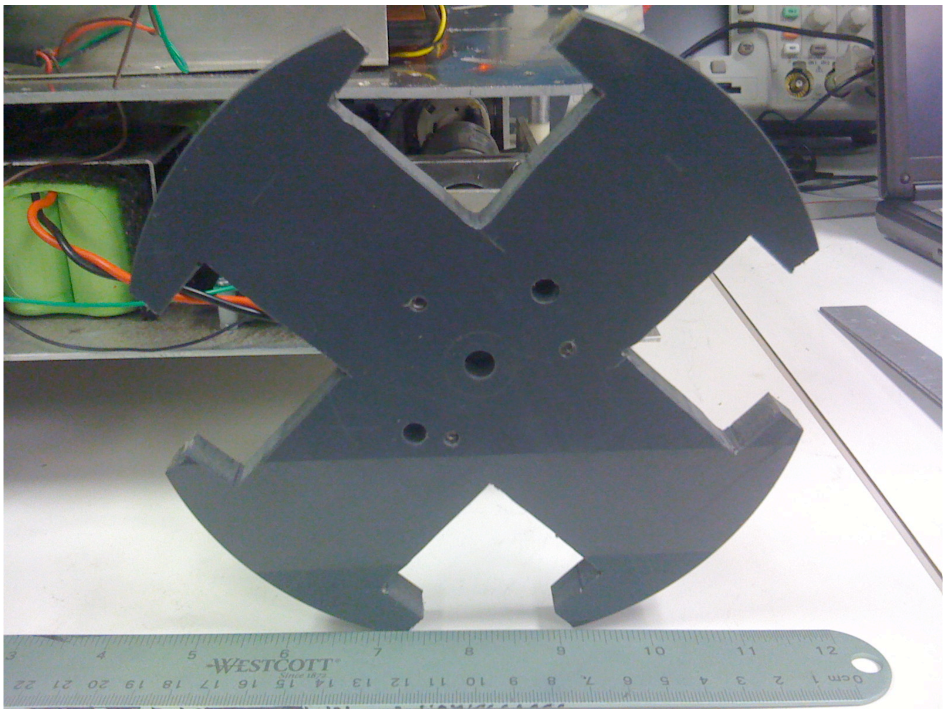

Figure 5. Robster Pelican Whegs™

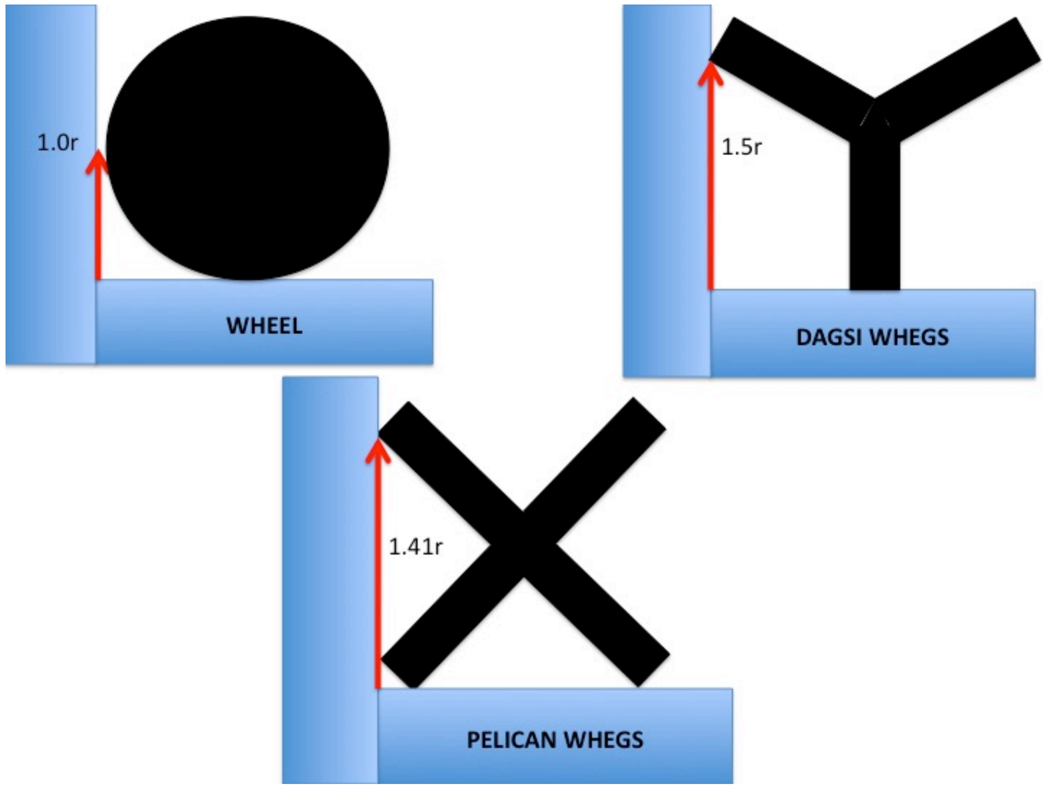

Figure 6. Wheel vs. DAGSI Whegs™ vs. Pelican Whegs™

3. Tail Mechanism

The Robster tail mechanism, pictured in Figure 7, was designed to approximate the desired function of the modeled Pelican Whegs™ tail device. A motor connected to a steel rod running lengthwise along the center rear half of the robot drives the tail. The gears that interlink the motor and shaft have a ratio of 1:1. This shaft, in turn, links to a second rod mounted parallel to the rear of the platform with a gearing ratio of 1:2. The tail is a 12″ wide by 12″ long by 0.5″ thick panel of clear PVC plastic mounted to the rear shaft by two aluminum brackets. All gears are straight, bevel gears fabricated from steel.

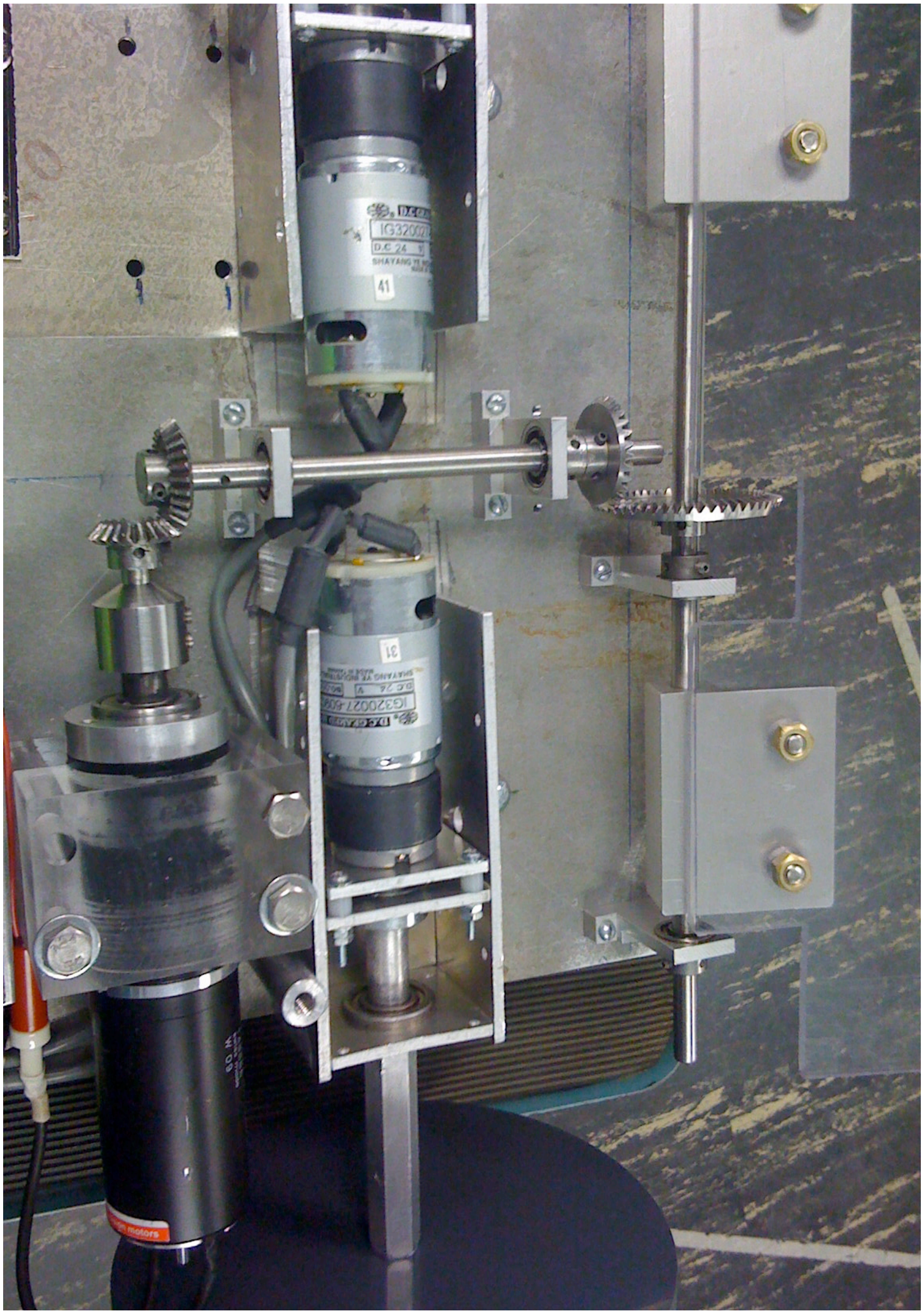

Figure 7. Tail Mechanism

A Maxon RE 40 148866 series motor with a GP 42C 203123 planetary gear head is used to drive the shaft of the tail assembly. This motor can provide 98.7mNm or

0.0728 lb-ft of continuous torque and operates at a maximum permissible speed of 8200RPM. The Maxon is driven by a 12VDC power source with a max continuous current of 6A and a no load current of 241mA. The planetary gear head provides a 74:1 reduction [6]. The motor output to the first bevel gear is therefore given a maximum of 5.387 lb-ft and 110.8 RPM. The output from the motor to the tail shaft through the 1:2 step up gear is 10.774lb-ft

Given the output torque values, a tail height of four inches, and a tail length of 12 inches, the vertical component of the downward force generated by the tail can be calculated as follows:

$$\theta = \sin^{-1}\left(\frac{h_3}{r}\right) = \sin^{-1}\left(\frac{4in}{12in}\right) = 19.5^\circ \quad (3)$$

$$F = \frac{T}{r} = \frac{10.774lb - ft}{1ft} = 10.774lb \quad (4)$$

$$F_y = F\cos(\theta) = (10.774lb)\cos(19.5^\circ) = 10.16lb \quad (5)$$

This would not be sufficient to lift the whole robot body. However, it would provide a large additional positive vertical force during any climbing operation.

B. ELECTRICAL COMPONENTS

1. Drive Control

a. Motor Battery

The battery driving the wheel motors is a rechargeable 24 VDC, 4000 mAhr Nickel Metal Hydride (NiMH) battery pack, shown in Figure 8. The battery pack is a 2 x 10 array of C cell batteries and weighs 3½ pounds. It is mounted between the two sets of wheels and relatively centered in the platform base.

b. Electronics Battery

The electronics battery is the Power Pad 160 rechargeable, 15VDC, 11,000 mAhr Lithium Ion battery, shown in Figure 9. This battery is mounted flat on Robster's chassis and weighs 2½ lbs. This battery can provide 27.4 minutes of power for operation of all Robster electronic components.

Figure 8. 24VDC Motor Battery (From [5])

Figure 9. Power Pad 160 laptop battery (From [5])

c. Motor Controllers

Two MD22 Devantech Dual Motor Drivers, pictured in Figure 10, control the Whegs™ speed and direction. Each motor controller can handle 5A current capacity for motors from 5 to 50V. A 3A fuse is connected in line with the +24V battery terminal to prevent high current draw to the circuit board. Both motor and logic ground are internally connected through the module providing the common ground for the whole robot. The motor controllers have five modes of operation. These motor drivers are set to

Control Mode 1, capable of providing two independent channels for separate motors, but which are instead tied together to electronically link each motor pair. A DC analog voltage provides the control signal from 5V (Full Forward) to 2.5V (Stop) to 0.0V (Full Reverse) [7]. The analog voltages are provided by the BL2000 through a LM6132 Buffer circuit. This protects the BL2000 Digital to Analog Converter (DAC) outputs from high current draws that will destroy the DAC's.

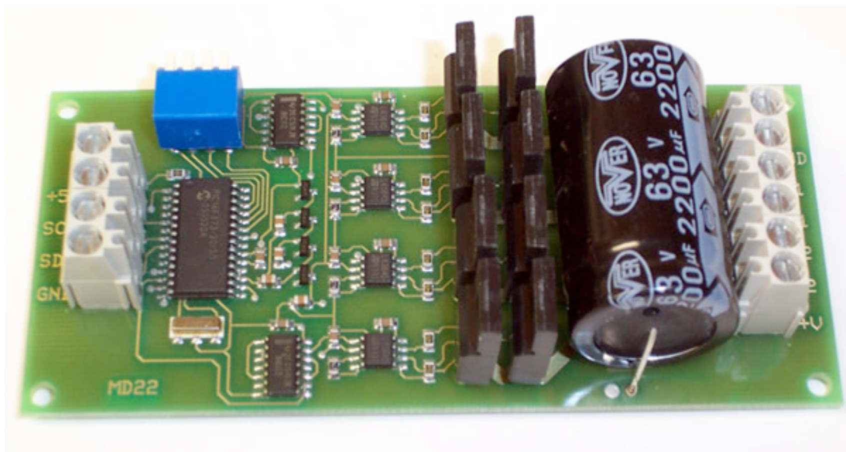

Figure 10. Devantech MD-22 motor controller (From [7])

2. Power Distribution

Robster requires 24V, 15V, 12V, and 5V to run its various electronic and mechanical components. The basic components for Robster's power supply and distribution system were taken from the Bigfoot robot developed by John Herkamp [5]. Mechanical power drawn from the 24V battery is routed directly to the motor controllers driving the wheels. AGC 4A, 250V glass fuses in line with the motor controllers protect against high current from the batteries. The 15VDC laptop battery provides the electronics power through a separate bus. Both 12V and 5V are reduced from the 15VDC battery by means of 7812 and 7805 voltage regulators, respectively. Common ground for all components is provided through the wheel motor controllers, which use both 5VDC and 24VDC (see Figure 11). Voltage requirements and current loads for each component are delineated in Table 1.

3. Tail Control

The tail mechanism is driven by a 12V battery connected through a Devantech MD-22 Motor Controller. The battery is a rechargeable 12VDC NiMH 4000mAh battery pack comprised of a 2 x 5 array of C cell batteries. A 4A fuse is place in line with the battery to protect the motor controller from high current draw from the motor. The MD-22 is driven by a DAC on the microcontroller, which is protected by a LM6132 Buffer circuit. This configuration is shown in Figure 12.

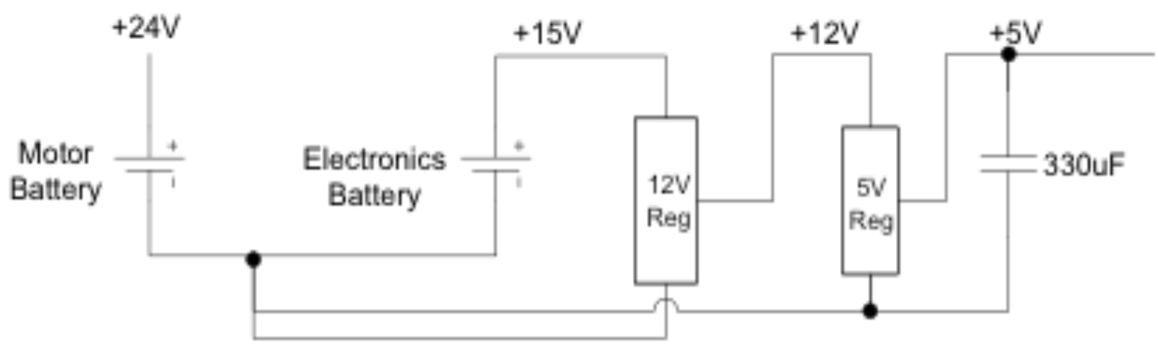

Figure 11. Power Distribution Design (From [5])

Component	Voltage Requirements (V)	Current Requirements (mA)
BL2000	15	60
Router	12	160
Motor Controllers	5	50
Buffers	5	0.36
PWM	5	3
GPS	5	60
Compass	5	35
Potentiometer	5	33
Total Current		401.36
Battery Life		27.4min

Table 1. Power Requirements

Separate motor driver circuits were built to drive a 12V PWM Motor Controller. This motor driver configuration functioned to drive the tail mechanism. However, this arrangement provided insufficient motor torque to lift the robot platform due to voltage droop through the voltage regulator circuit.

a. PWM Motor Controller

The motor controller for the tail mechanism is the SuperDroid Robots PWM Motor Controller (see Figure 13). This motor controller can handle from 12-55VDC and drive 3A continuously with surges up to 6A using the LMD18200H-bridge IC. It has a four pin header which provides inputs for ground, break, PWM input, and direction. Break is used to effectively short the Output terminals when set to logic HIGH. Direction controls the direction of current flow between the two Output leads, determining the direction of motor rotation. PWM input operates from 0-5V at a minimum of 1kHz. These operational parameters are displayed in Table 2 [8].

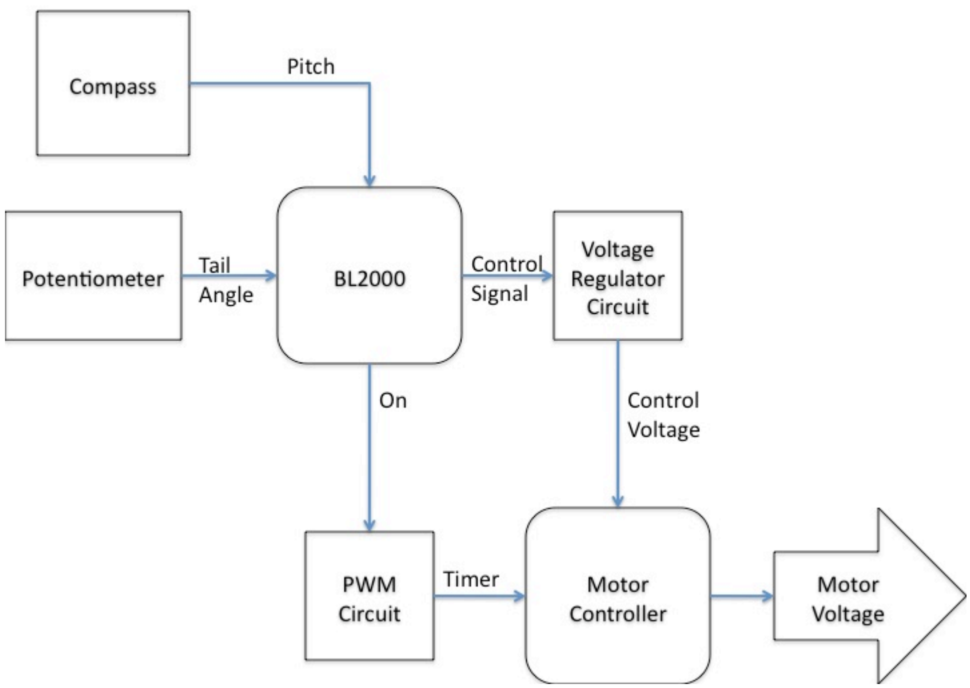

Figure 12. Tail Component Functional Diagram

A simple buffer circuit was built to provide the signal for the DRIVE and BREAK. This buffer was built to protect the digital output ports on the BL2000 from sudden high current pulls generated by the motor controller using a LM6132 Buffer op-amp.

PWM	Dir	Brake	Active Output Drivers
H	H	L	Source 1, Sink 2
H	L	L	Sink 1, Source 2
L	X	L	Source 1, Source 2
H	H	H	Source 1, Source 2
H	L	H	Sink 1, Sink 2
L	X	H	NONE

Table 2. Motor Controller Truth Table (From [8])

b. 1.4kHz PWM Circuit

A Pulse Width Modulation (PWM) circuit drives the motor controller (see Figure 14). This simple PWM circuit is generated by a LM555 timer integrated circuit operating in an astable-oscillator configuration [9]. From the figure below, the resistors are $R_1 = 4.7k\Omega$ and $R_2 = 4.7k\Omega$ with a capacitor $C_1 = 0.1\mu F$.

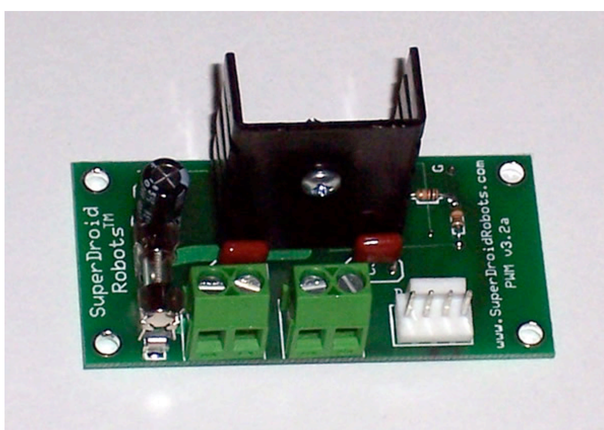

Figure 13. PWM Motor Controller (From [8])

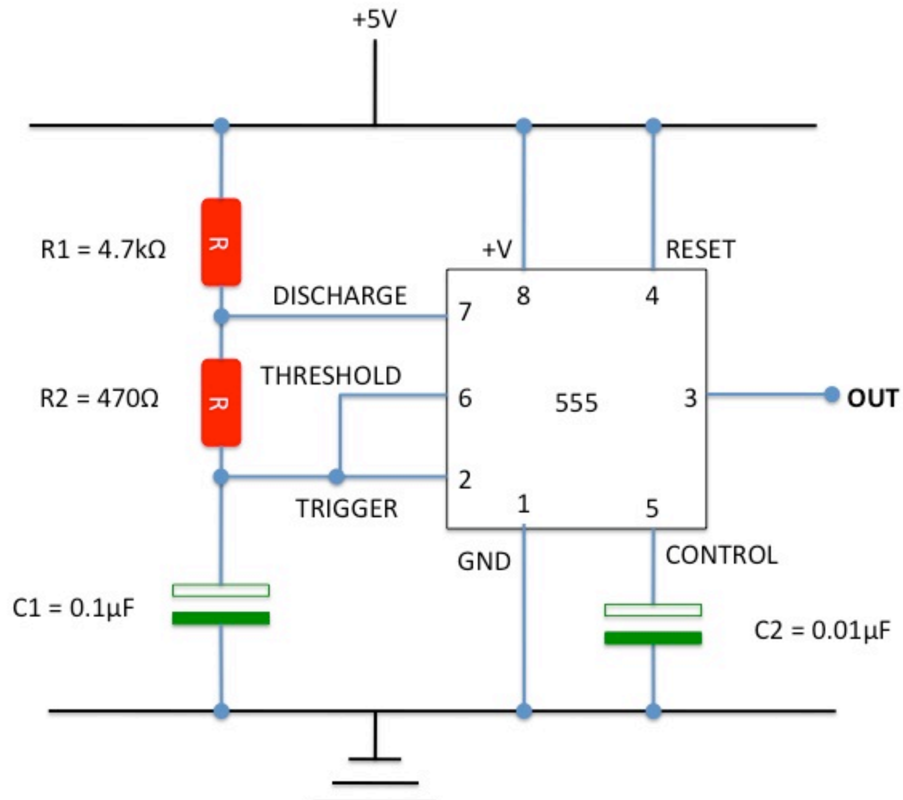

Figure 14. 2.56kHz PWM Circuit

Given the resistance and capacitance values for this circuit, the PWM circuit has the following characteristics (see Figure 15):

$$f = \frac{1}{0.693(R_1 + 2R_2)C_1} = \frac{1}{0.693(4.7k\Omega + 2(470\Omega))0.1\mu F} = 2.56kHz \quad (6)$$

$$T_H = 0.693(R_1 + R_2)C_1 = 0.693(4.7k\Omega + 470\Omega)0.1\mu F = 358\mu s \quad (7)$$

$$T_L = 0.693R_2C_1 = 0.693(470\Omega)0.1\mu F = 33\mu s \quad (8)$$

$$DC = \frac{T_H}{T_H + T_L} = \frac{358\mu s}{358\mu s + 33\mu s} = 91.6\% \quad (9)$$

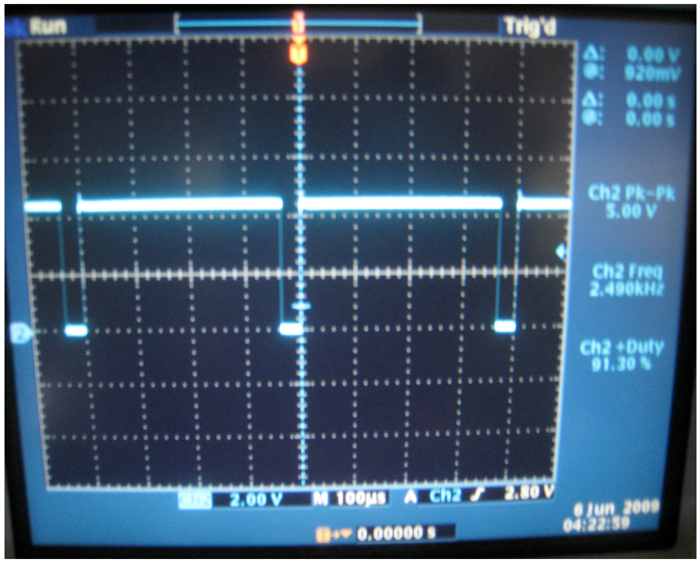

Figure 15. PWM Waveform from Oscilloscope

c. Tail Voltage Regulator Circuit

This circuit converts the 24VDC power of the motor battery and reduces it to 12VDC power for use with the Maxon tail motor, as shown in Figure 16. This circuit uses a standard 7812 voltage regulator IC with a maximum 1A output to provide maintain the +12V voltage [10]. A high current MJ2955 PNP transistor and Dale-RH 4Ω, 25W power resistors function to boost the output current at the regulated voltage. A 1A fuse placed on the output side of the voltage regulator prevents high current from leaking back to the regulator.

d. Potentiometer

A variable resistance potentiometer functioning as a voltage divider determines inclination of the Tail. A Spectrol 536 wire wound precision rotary potentiometer is used in this application. The Spectrol 536 has resistive range from 0.5Ω to $100k\Omega$ with a tolerance of $\pm 5\%$. This ten turn potentiometer can be rotated through 3600° [11].

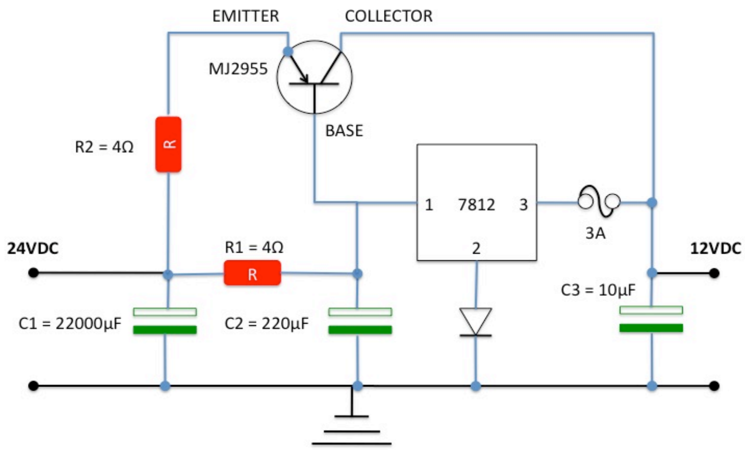

Figure 16. 12VDC Voltage Regulator circuit

This potentiometer is firmly connected to the right end of the axial tail shaft, rotating with the tail shaft, shown in Figure 17. Five volts is applied across the outer terminals of the while the wiper is connected to an ADC input on the BL2000. The baseline voltage for this configuration 2.499V at 100Ω indicating a 0° incline. As the tail moves up or down, the voltage follows a linear relationship inversely proportional to the change in angular position.

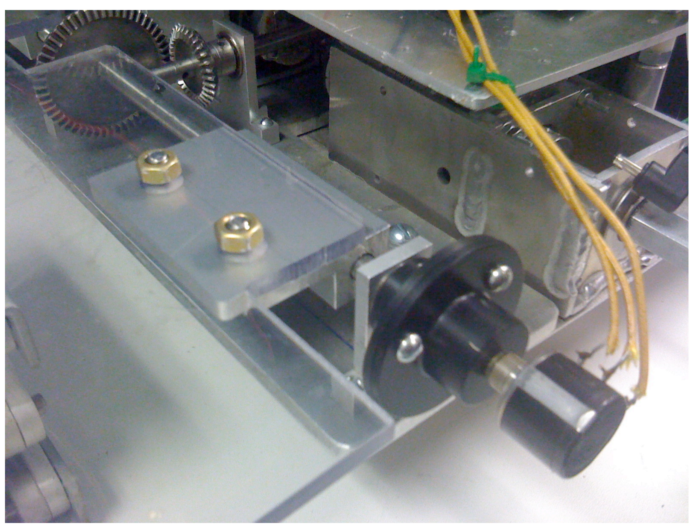

Figure 17. Spectrol 536 Potentiometer

C. ELECTRONIC COMPONENTS

1. Microcontroller

The BL2000 Wildcat microcontroller is a single-board computer that offers high performance in a compact form factor, pictured in Figure 18. The BL2000 incorporates the 22.1MHz Rabbit microprocessor, 256K flash memory, 128K static RAM, 28 digital input/output ports, 9 12-bit analog/digital converter inputs, 2 12-bit digital/analog converter outputs, 4 serial ports, and 1 RJ-45 Ethernet port. It is robust, highly adaptable, and easily programmed using Dynamic C [12].

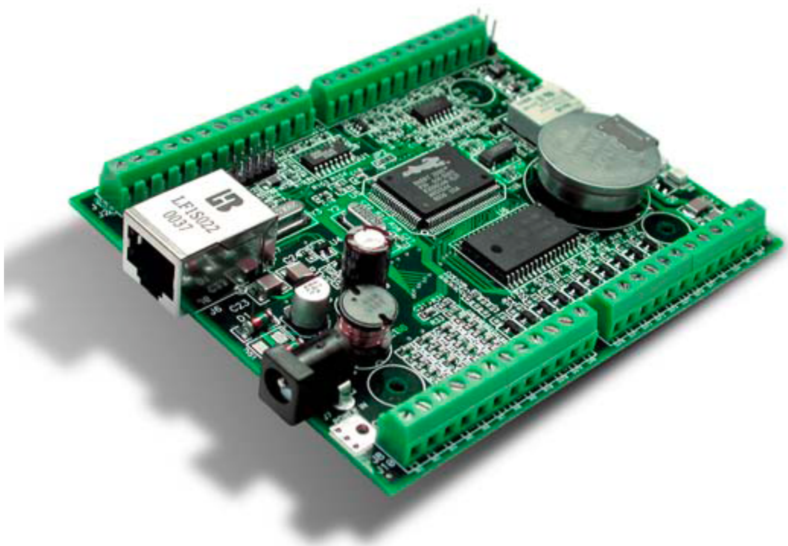

Figure 18. BL2000 Microcontroller (From [12])

2. Router

During operation, all communications with Robster are directed through a Netgear Rangemax 240 Wireless G router installed onboard the platform, pictured in Figure19. This router operates on the IEEE standard 802.11B and G at 2.4GHz with a maximum data rate of 240Mbps and a range of up to 300 ft. It has four built in 10/100 Mbps switch inputs to connect devices, one of which is used to connect the BL2000. The router operates on 12VDC input power and draws 160mA of continuous current [13]. All communication utilizes the standard UPD protocol. The router is connected to the RJ-45 Ethernet port on the BL2000 and provides the communications pathway for the JAVA interface.

Figure 19. Netgear Rangemax 240 Wireless Router (From [13])

3. GPS

Positional data is provided by the Garmin GPS-16HVS antenna and receiver system, shown in Figure 20. The receiver is a 12 channel Wide Area Augmentation System (WAAS) capable of simultaneously tracking 12 satellites to compute a differential GPS fix for a position accuracy of 3-5m. It updates in interval of 1 to 900 seconds in 1 second increments. This GPS unit requires 3.3 to 6VDC-regulated power typically drawing 65mA of current. It communicates using true RS-232 output and asynchronous serial input with RS-232 and TTL voltage levels. This application uses the National Marine Electronics Association (NMEA) 0183 v2.0 ASCII serial format with GPGGA as the primary output sentence [14].

An example GPS NMEA output string is as follows:

$GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47

GGA	Global Positioning System Fix Data	
123519	Fix taken at 12:35:19 UTC	
4807.038,N	Latitude 48 deg 07.038' N	
01131.000,E	Longitude 11 deg 31.000' E	
Fix quality:	0 = invalid	1 = GPS fix (SPS)

2 = DGPS fix 3 = PPS fix

4 = Real Time Kinematic 5 = Float RTK

6 = estimated (dead reckoning) (2.3 feature)

7 = Manual input mode 8 = Simulation mode

08 Number of satellites being tracked

0.9 Horizontal dilution of position

545.4,M Altitude, Meters, above mean sea level

46.9,M Height of geoid (mean sea level) above WGS84 ellipsoid

(empty field) time in seconds since last DGPS update

(empty field) DGPS station ID number

*47 the checksum data, always begins with *

[14]

Figure 20. Garmin GPS 16HVS (From [14])

4. Compass

Robster heading, tilt, and incline are determined using the Honeywell HMR3000 Digital Compass, shown in Figure 21. The HMR3000 uses three magneto resistive magnetic sensors and a liquid filled, two axis tilt sensor to produce accurate compensated heading data for up to 45° of tilt (see Figure 22). The magneto-resistive sensing elements are composed of NiFe thin films deposited on a silicon substrate as a Wheatstone resistor bridge (see Figure 23). The magnetometer has a wide dynamic range of ±2 Gauss

(200μT) compared with 0.65 Gauss for earth's magnetic field and therefore should not saturate. The compass has an accuracy of 0.5° with 0.1° of resolution [15].

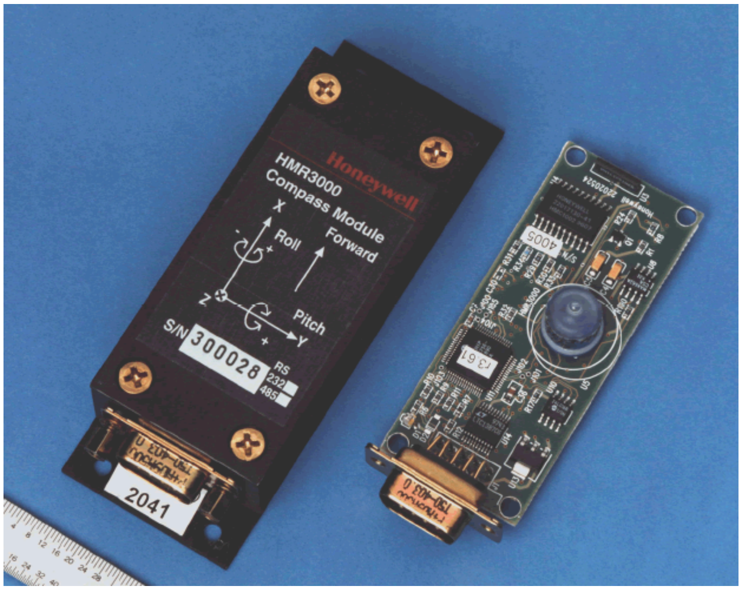

Figure 21. HMR3000 Digital Compass (From [15])

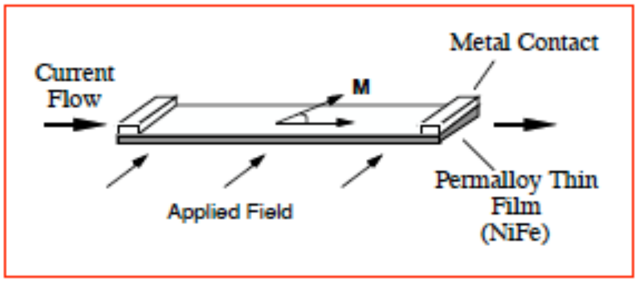

Figure 22. MR Sensor Basics (From [15])

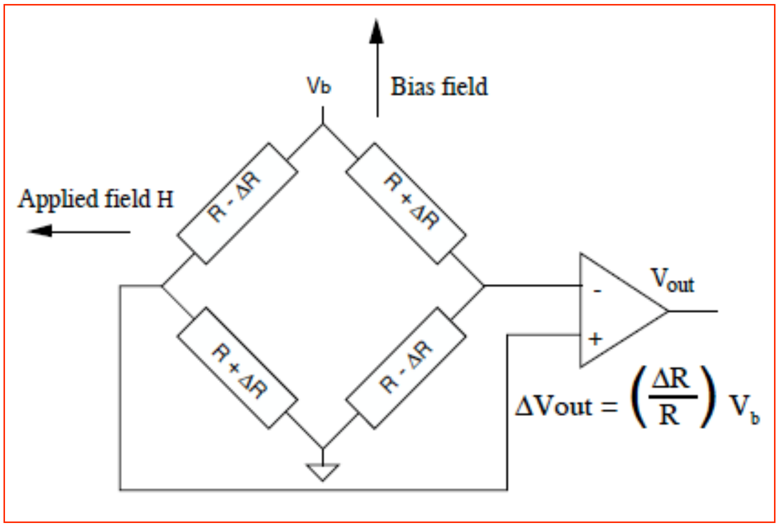

Figure 23. Wheatstone Bridge (From [15])

Compass heading is calculated at 13.75Hz from 5 filtered measurements: TiltX, TiltY, MagX, MagY, and MagZ. The HMR3000 is powered by 5V regulated supply but is capable of operating with 6-15V unregulated power supply. It communicates using standard serial RS-232 connection using an NMEA 0183 output string at 19200 baud [15].

An example compass NMEA output string is as follows:

$PTNTHPR,85.9,N,-0.9,N,0.8,N*2C

HPR	Heading, Pitch, and Roll
85.9	Heading 85.9° magnetic
-0.9	Tilt -0.9° x-axis
0.8	Roll 0.8° y-axis
N*2C	checksum for parity

III. ROBOT CONTROL

A. CONTROL ALGORITHM

A computer algorithm embedded on the BL2000 microprocessor controls Robster. The code is written in Dynamic C and compiled using Dynamic C 7.1.9. The program uses a series of costatements to permit the processor to conduct cooperative multitasking operations. Costatements are a feature of Dynamic C that permits the program to perform several tasks simultaneously by voluntarily releasing processor time to the next function during delays in the individual tasks. The components of this control algorithm were developed for the Bender robot and described in detail in [3]. The basic outlines of this control algorithm are provided in Table 3 and Figure 24.

FUNCTION	PORT
Wheels	DAC1
Tail	DAC0
Compass	Serial C
GPS	Serial B
Potentiometer	ADC0

Table 3. Interface Architecture

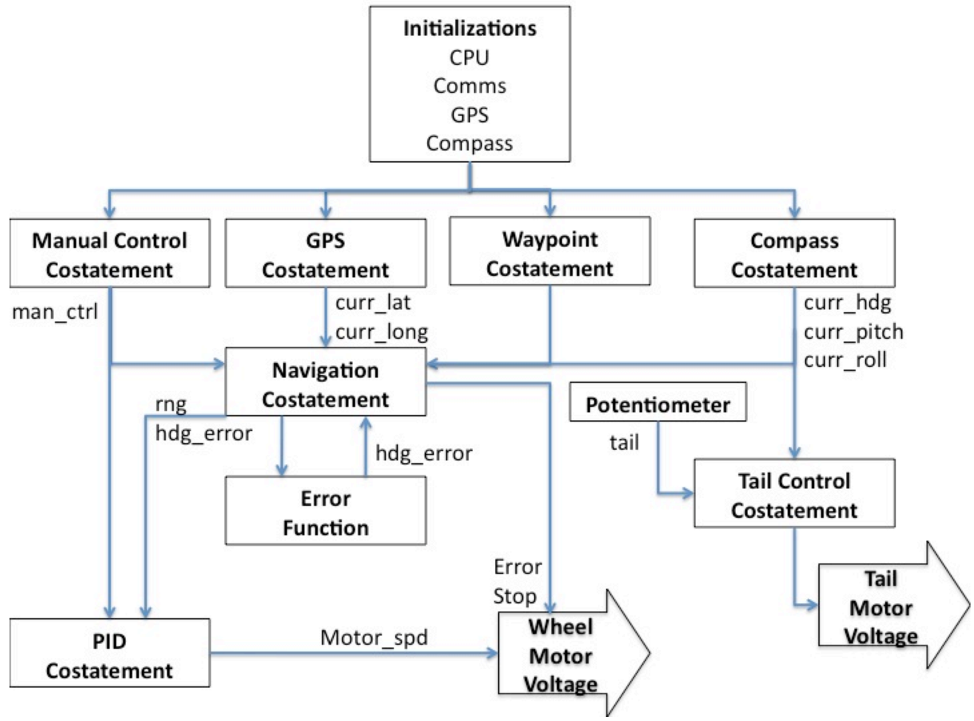

Figure 24. Control Algorithm Flow (After [3])

1. Manual Control

The user initiates the manual control costatement through the JAVA interface by way of port 4001 calls manual control. Manual control overrides all autonomous navigation functions and sets the man_ctrl flag that prevents the robot from entering the navigation and PID costatements. Manual control receives a string from the JAVA application buttons, converts the string to integers, and parses it for control voltages for the left and right motor pairs. The motor control signal for the left motor pair is directed by DAC1 and the right side by DAC0.

2. Waypoint

The user initiates the Waypoint costatement through port 4002. It stores the waypoint coordinate data from the JAVA Application and parses that data into an acceptable form for the Navigation costatement. In addition, this function also resets the man_ctrl flag.

3. Navigation

The Navigation costatement is initiated by the Waypoint costatement from the JAVA interface. It receives the waypoint data and passes heading error and range from waypoint information to the Control costatement. It uses the error function to determine the heading error value from the new_hdg and curr_heading variables.

4. GPS

The GPS costatement triggers the GPS receiver and translates that data for the JAVA GUI through port 4004. The GPS receiver is controlled on the BL2000 on Serial Port C. GPS data is updated in the GUI every one second.

5. Compass

The compass costatement triggers the compass and translates the data for heading, pitch, and roll to the JAVA GUI through port 4003. The compass is controlled by the BL2000 on Serial Port B. The compass is configured to update to the BL2000 five times per second.

B. TAIL CONTROL

The tail costatement takes data from the digital compass and potentiometer inputs, determines the optimal angular position of the tail mechanism, and transmits the drive signal to the tail motor. For this experiment, the tail angle will follow the pitch angle of the robot unless the robot exceeds a predetermined positive pitch angle. At this angle, the tail will immediately lower to provide the boost function described by [4].

The compass input is parsed from the NMEA string output statement of the compass. The ASCII characters representing the pitch data are stored in memory and converted into a floating-point variable, which is interpreted by the BL2000. An offset is added to the Pitch value to correct for error due to imperfect mounting of the compass to the platform.

The tail position is computed from the analog voltage generated by the potentiometer and read into ADC0. This value is initialized upon startup to determine the voltage that corresponds to a 0° angle, indicating that the tail is horizontal. Moving the tail updates the stored analog voltage and converts this to a tail angle based on

$$\theta_t = c_1 * (V_t - V_0) \qquad (10)$$

where V_0 is the initial voltage that corresponds to $\theta = 0^\circ$, V_t is the current voltage, c_1 is an experimentally determined proportionality constant which converts the voltage to an angular value and θ_t is the computed position of the tail with respect to the horizontal (see section IV-A).

A proportional coefficient determines the desired motor drive signal output to DAC1 (see Figure 25). This statement provides a motor drive signal proportional to the difference between the current compass pitch and tail angle values given by

$$V_T = \frac{\theta_c - \theta_t}{c_2} + V_S \qquad (11)$$

where θ_c is the compass pitch, V_S is the tail stop voltage, V_T is the tail motor drive voltage, and c_2 is an experimentally determined proportionality constant which converts the angular measurement given in degrees into a voltage (see section IV-B).

The control algorithm for the tail is a series of *if…then…else* function calls that bound the parameters for tail actuation. The *if…then…else* statement is a C command which performs a task only if defined criteria are met during the function call. If not, it will check whether the parameters of the else statement are met before ignoring the function. A series of nested *if…then…else* provide boundary conditions for different motor commands.

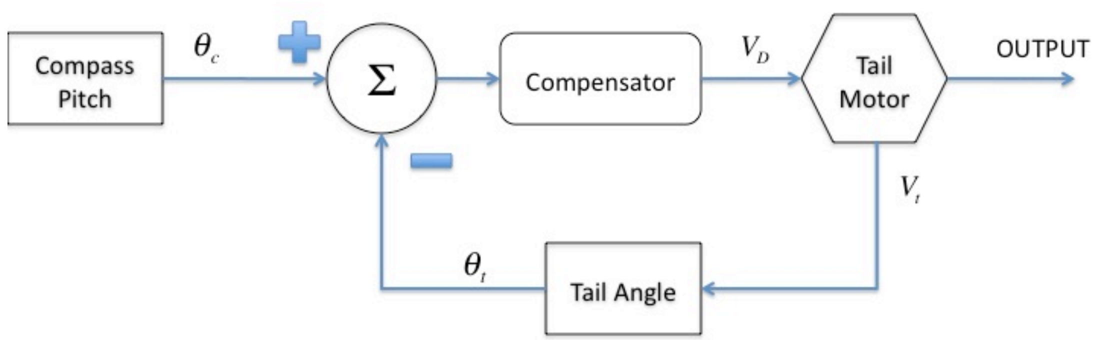

Figure 25. Proportional Control Loop

C. JAVA GRAPHICAL USER INTERFACE

Kubilay Uzan developed the JAVA Graphical User Interface for use in the NPS SMART program. This program takes data input from the GPS receiver and Compass and returns motor control signals for the wheels. All information is passed through the router. The user interface is a JAVA application, which appears as a map on the laptop screen (see Figure 26). GPS data on fix time, available satellites, latitude, and longitude, and compass data on heading, pitch, and roll are parsed to the interface. Error messages are processed to a separate error indicator. Both manual control and waypoint navigation commands can be input by the user and output to the robot [5].

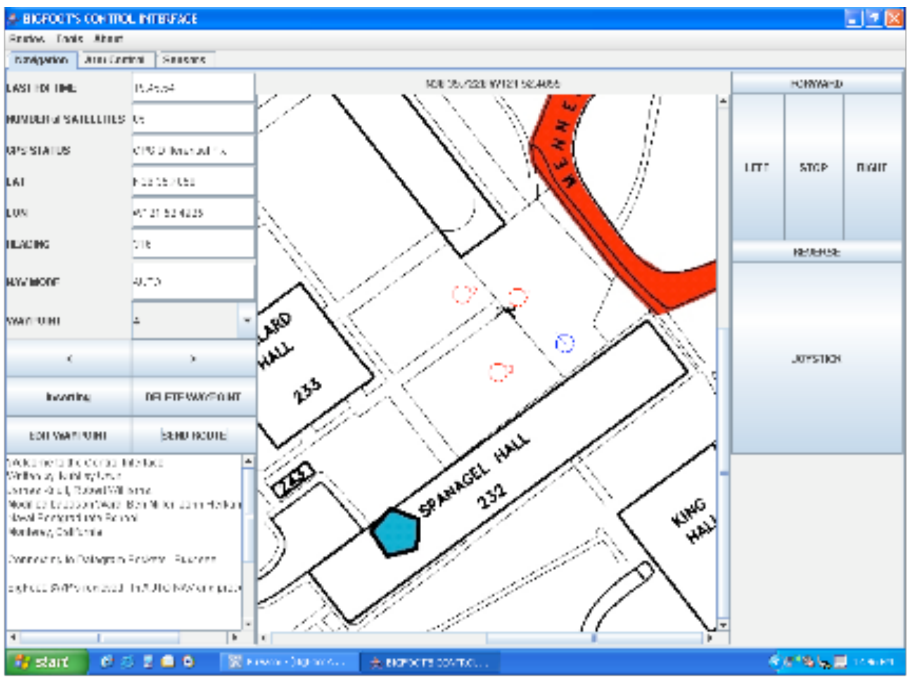

Figure 26. JAVA GUI Screen Capture (From [5])

THIS PAGE INTENTIONALLY LEFT BLANK

IV. RESULTS

A. TAIL POSITION CALIBRATION

To calculate the angle of the tail with respect to the horizontal, the consant, c_1, from Equation 10, had to be determined experimentally. In other words, the potentiometer output to the BL2000 was characterized to generate an accurate, reliable measurement of the angle of the tail as shown in Figure 27.

To do this, the robot was placed on an elevated platform and the embedded Dynamic C program was initiated. The wheel and tail motors were both deactivated. The tail was then manually elevated and lowered using a Cenco-Lerner Lab Jack to maintain a fixed angle. The tail was moved in increments of 5° with each new angular position verified by a protractor and liquid level.

The result was the linear relationship between the voltage and the indicated tail angle. This data was plotted in Octave GNU and returned a linear regression plot of the resulting data pictured in Figure 27. The slope of the line generated is determined to be:

$$c_1 = \frac{\Delta\theta_t}{\Delta V} = \frac{74.37^\circ}{74.336mV} = 1000\,\tfrac{\deg}{V}\; or\, 1\,\tfrac{mV}{\deg} \qquad (12)$$

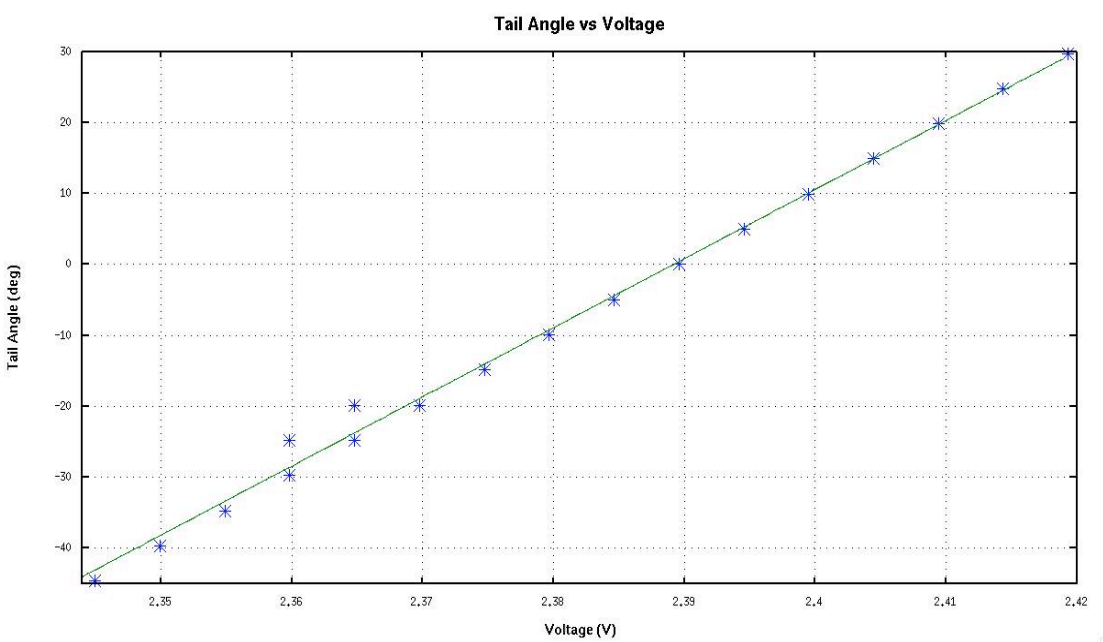

Figure 27. Tail Angle vs. Voltage

The proportionality constant, c_1, is utilized to determine the actual tail angle. During dynamic evaluation, the computed tail angle became highly erratic due to noise generated by other loads on the source voltage. Offsets in the horizontal base voltage also developed due to friction between the potentiometer and tail shaft. To reduce these effects, the proportionality constant was adjusted several times to test the response of the tail position indication. For the proportionality constant, a value $c_1 = 500\frac{deg}{V}$ generated a properly damped observed response. The applied c_1 was only 50% of the determined value.

B. TAIL POSITION VS. ROBSTER PITCH – STATIC

Tail position response to compass pitch angle was characterized using a static demonstration of the tail control algorithm. For this experiment, the robot was placed on an elevated platform, and the pitch of the robot was manually manipulated by tilting the robot body, depicted in Figure 28. The tail motor controls the position of the tail angle in accordance with parameters defined in the robot control algorithm.

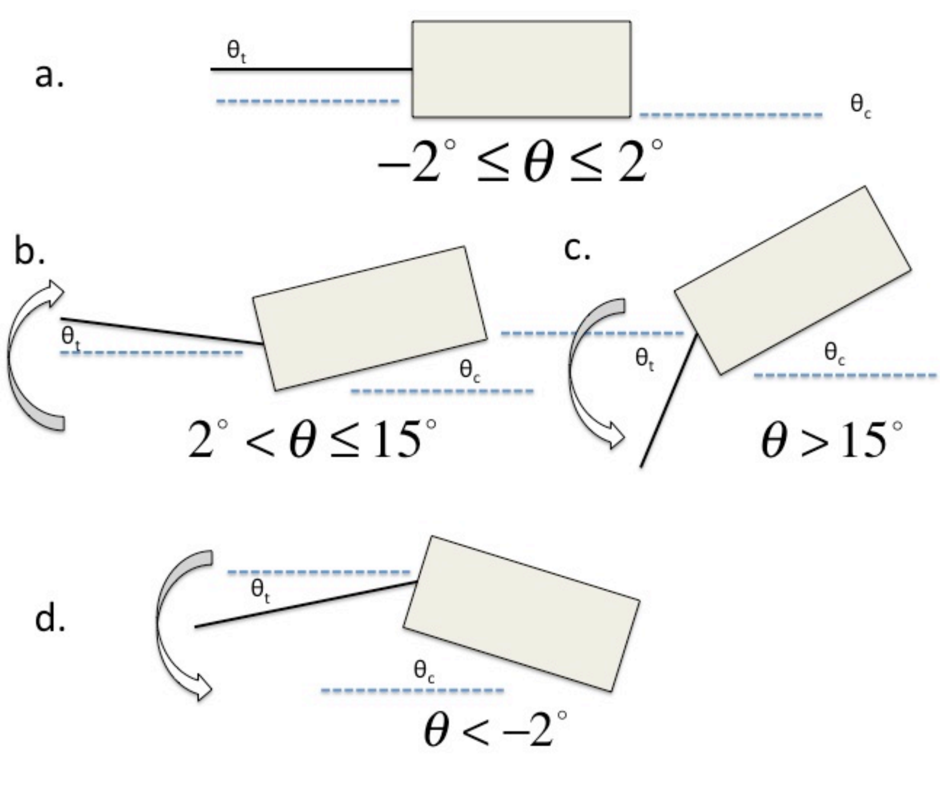

Figure 28. Static Test Concept

The results for the static demonstration were plotted in Figure 29 and were in agreement with the desired results. For the plot, CPU run time starting at i = 200, the robot pitch angle was steadily increased and the tail responded by increasing with the increasing tail angle. When the pitch was increased beyond the critical angle, set at 15^{o}, the tail reacted by rapidly changing its attitude downward to stop at the negative stop, set at -35^{o} for this test. When the pitch angle declined below the critical angle, the tail rapidly returned to its position and continued following the pitch angle. This event can be seen between i = 5000 and i = 5800. After this event, the tail attitude continues to follow the pitch angle, from a positive angle through negative angles and back to the horizontal.

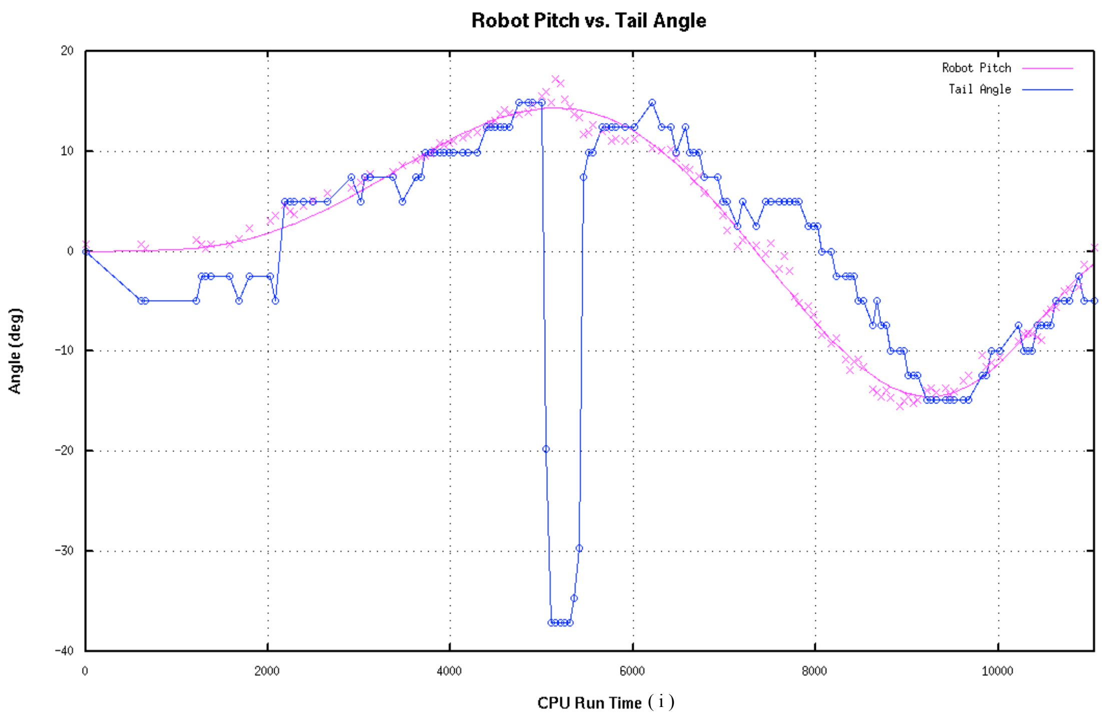

Figure 29. Robot Pitch and Tail Angle

The proportionality constant c_2 from Equation 11, used to generate the output control voltage, was determined analytically and through experimental testing. The limits for the tail angle, θ_t, provide a maximum allowable differential of 70^{o}. The stop voltage for the tail motor controller is V_S=2.5V with a voltage range between 0V for maximum reverse voltage and 5V for maximum forward voltage. Using Equation 11, a minimum value for c_2 can be derived.

$$V_T = \frac{\theta_c - \theta_t}{c_2} + V_S \quad (11)$$

$$0V < \frac{30^\circ - {}^{-}40^\circ}{c_2} + 2.5V < 5V \Rightarrow \qquad c_2 \geq \left|28\,\tfrac{\text{deg}}{V}\right|$$

These bounds correspond to the voltage bandwidth of the motor controller. In the case where motor controller voltage is outside these bounds, the motor signal becomes erratic and cannot be properly characterized. The robot rapidly approaches this limit in the case where pitch exceeds 15^o and the tail moves down to drive the rear of the robot. To avoid this situation, $c_2 = 35\frac{\text{deg}}{V}$ was chosen to provide a buffer for the motor controller voltage. This limited maximum drive signal voltage to $0.5V<V_T<4.5V$.

Physical upper and lower bounds for the tail angle were encountered where the tail struck the lower base of the robot at its maximum negative angle and where the plastic tail plank impacted the longitudinal shaft of the tail mechanism at its maximum positive angle. The physical limits of the tail were measured to be $\theta_t^{\max} = 75^\circ$ and $\theta_t^{\min} = -60^\circ$. For this test, the tail would only actuate between limits $-40^\circ \leq \theta_t \leq 30^\circ$ in order to avoid slamming into these stops and possibly damaging the assembly. If the tail overshoots these bounds, motor control voltage is set to stop and then a small drive voltage is sent to reverse the direction of the tail and return it to the desired limits.

C. TAIL POSITION VS. ROBSTER PITCH – DYNAMIC

1. Experimental Setup

Dynamic tests were performed to evaluate the performance of the tail algorithm during robot operation. This test was used to characterize the performance of the compass pitch sensor under the non-ideal conditions of yaw and vertical translation during robot motion. The test characterized the response of the tail control algorithm to this unpredictable environment and its ability to generate a desired tail angle.

The dynamic test entailed driving the robot forward over a ramp elevated to different slope angles and recording the response of the compass and potentiometer with time. The ramp was a simple platform made of two $60" \times 40"$ wooden boards meeting at vertices to form an isosceles triangle, see Figure 30. Wooden blocks support the apex of this ramp from beneath and are adjusted to provide the desired elevation. Rubber strips were attached to the surfaces of the Whegs™ to increase dynamic friction between the plastic wheels and the smooth wooden surface.

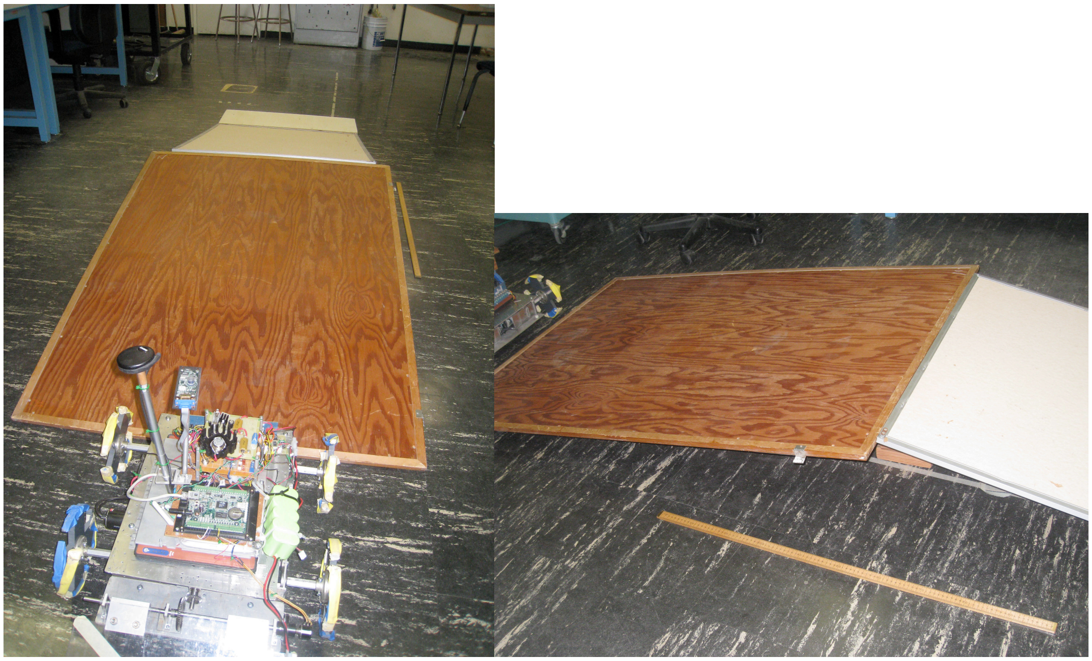

Figure 30. Dynamic Test Ramp

Figure 31 is a pictorial representation of the desired dynamic test characteristics. The desired performance of the tail control, as a function of the robot pitch angle, was identical to that of the static test of tail control. On level ground, the robot pitch should indicate approximately zero angle and the tail should be parallel to the surface (see Figure 31 a). As the robot began to ascend the ramp, the pitch should indicate this increasing positive angle and the tail angle should increase directly proportional to the increasing pitch (see Figure 31 b). As the pitch exceeded a critical value for each ramp slope, the tail should immediately descend and remain in that position until the pitch fell below the critical slope angle (see Figure 31 c). As the robot descended the ramp, the pitch angle should become negative and the tail should continue to follow the pitch (see Figure 31 d). As the robot returned to level ground, the pitch angle and the robot tail incline should both indicate this by resuming the original condition of zero angle for both pitch and tail (see Figure 31 e).

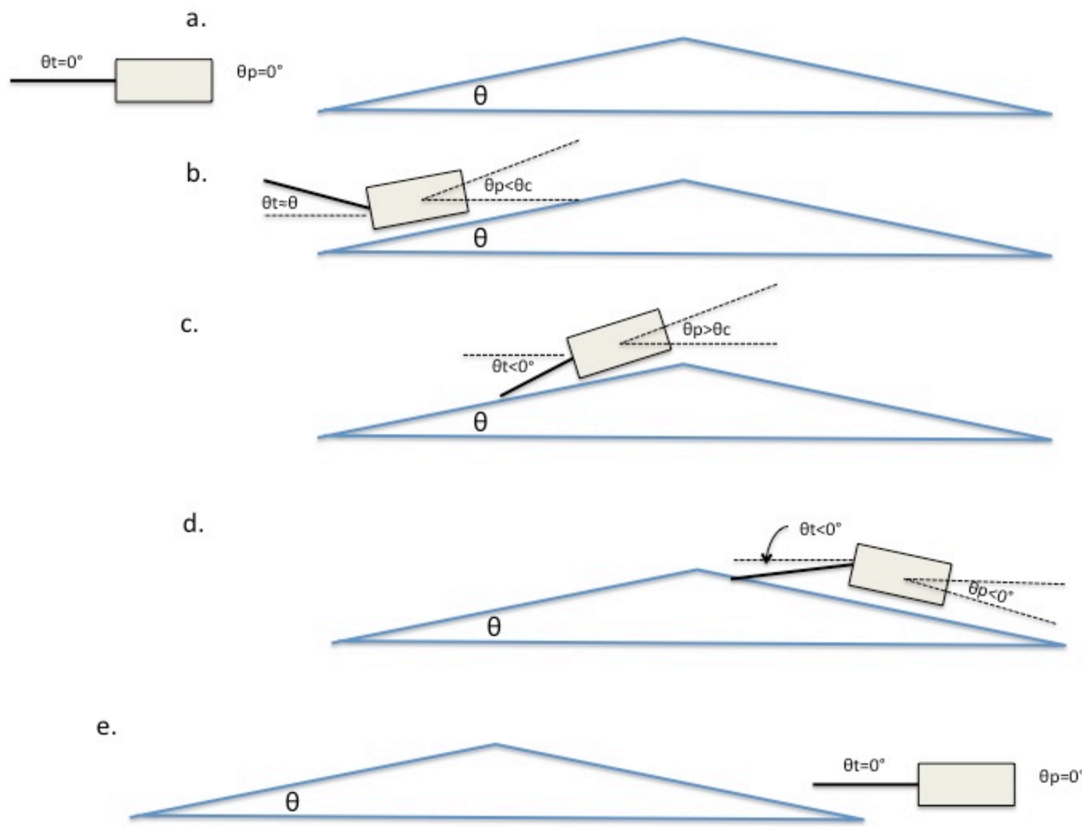

Figure 31. Dynamic Test Concept

The tail control algorithm was optimized during testing to generate a smoother, more consistent response as the robot crossed the ramp. To record data, the robot remained directly connected to the laptop by means of the BL2000 programming cable and the data was obtained by means of a printf terminal output command. Output would only be collected for recorded changes of the compass pitch of $\theta_c > |0.2^\circ|$. Ten data sets were recorded utilizing this method, five for the platform elevated to a 10^o slope and five for the platform elevated to a 15^o slope. An example of an ideal data set is depicted in Figure 32. Letter labels in Figure 32 correspond to the letter labels given to the steps depicted in Figure 31.

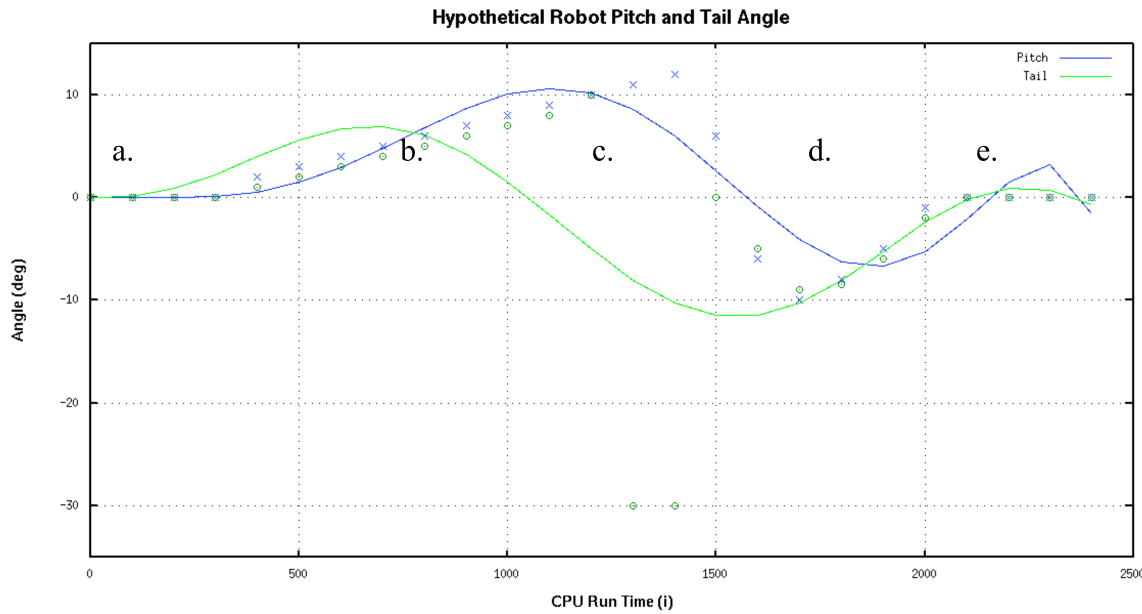

Figure 32. Desired Robot Pitch and Tail Angle Plot

2. Experimental Results

The data from the experimental trials were plotted in MATLAB™ to evaluate the performance of the pitch indication and tail sensor. For all plots, both individual data points and seventh order linear regression fit lines of each data set are displayed with different colors. The plots for average robot pitch and tail angle for each set of trials includes error bars characterizing the standard deviation of these results (see Table 4).

Trial (Average)	Standard Deviation
Robot Pitch 10°	5.77
Tail Angle 10°	9.41
Robot Pitch 15°	7.32
Tail Angle 15°	10.06

Table 4. Average Standard Deviations

Large data scatter was a significant factor in the resulting pitch and tail incline data. Motion of the Whegs™ platform is subject to a vertical translation, pitch in the direction of motion, and yaw perpendicular to the direction of motion during normal horizontal movement. A Whegs™ platform is analogous to driving on square wheels. Though the ends are curved to the arc of a circle, they cannot obviate this innate limitation. In addition, the compass mounting was jostled and vibrated due to these shocks. The nonlinear characteristics of the platform and the pitch sensor resulted in the large scatter observed in the recorded data points from the dynamic experiment.

Results for the 15° ramp were generally much worse than the shallower ramp due to mechanical limitations of the robot. Climbing the steeper ramp appeared to approach the limits of the torque capabilities of the driving motors and the dynamic friction achievable by the Whegs™. During each run, the robot slowed dramatically as it climbed the ramp, and accelerated rapidly on the downward side of the ramp. The resulting data was very chaotic, particularly for the tail angle, even though the robot appeared to perform its desired tasks based on visual observations.

a. Robot Pitch

Figure 33 represents the robot pitch angle for a single run of the robot over the ramp. The plot for individual data points indicates a great deal of noise in the pitch indication as the robot proceeded over the ramp. However, the regression line indicates that the recorded data generally corresponded with our desired results. The large pitch measurement at i = 1600 was likely produced by the impact of the front Whegs™ on the flat ground as the robot rapidly descended the slope.

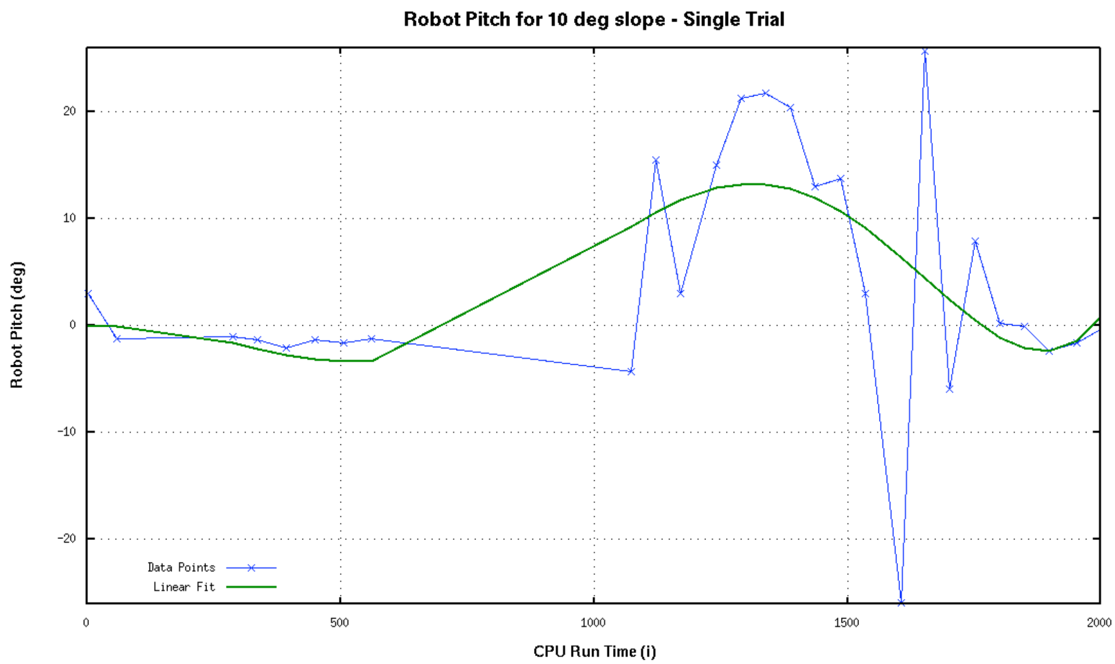

Figure 33. Robot Pitch vs. Time - Data Points and Linear Fit for Single 10° Trial

Robot pitch data for both ramp angles was plotted in Figure 34 and Figure 35 to illuminate trends in the recorded pitch data over the course of ten tests. From all trials, the compass data proved to be a rough but useful estimate for characterizing the robot's pitch during actual operation. The change in angle over time is apparent through the majority of the regression plots. At a given time, the pitch angle was observed to steadily increased to a maximum in most trials. At this point, a transition from positive incline to negative decline was observed. Both ramp angles showed a transition of pitch angle from positive to negative as the robot body continued along the downward slope. All plots indicated the change in slope of the pitch from negative back toward zero, indicating the robot returning to a level surface.

However, all trials produced a significant amount of spurious angular data that affected the response of the tail control algorithm. Both plots recorded large amounts of scatter in the results. For the shallow slope, the individual linear fit curves were fairly consistent, but all of the runs significantly overshot the ramp angle near the tip-over point. On the downward slope of the steeper ramp, the individual data points alternated between positive and negative values for several of the runs, flattening out the regression plots. No explanation could be found because these results could not be reproduced consistently through all trials and were sometimes absent in others.

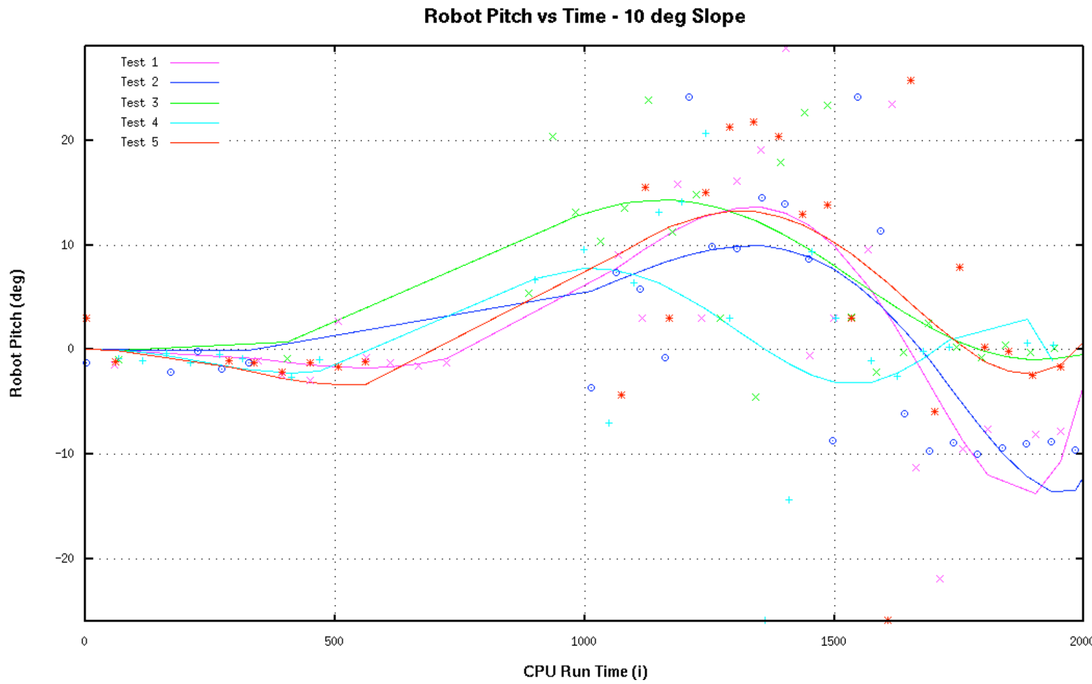

Figure 34. Robot Pitch vs. Time - 10°

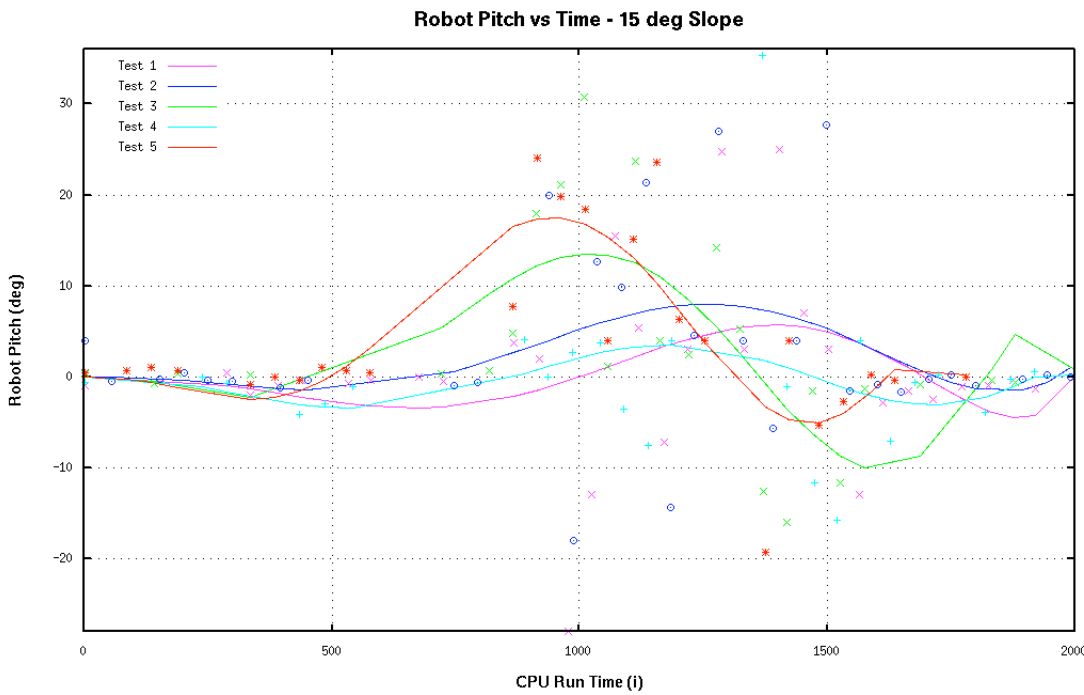

Figure 35. Robot Pitch vs. Time - 15°

The final plot, Figure 36, compared average pitch results for both ramp heights. The pitch results for both trials produced average values less than the actual amount, an overdamped condition. However, the 10° slope was much closer to representing the correct result than the higher angle slope. In addition, the standard deviation for the 15° slope was 27% larger than that of the shorter slope, indicating that the results were far less consistent over the data set.

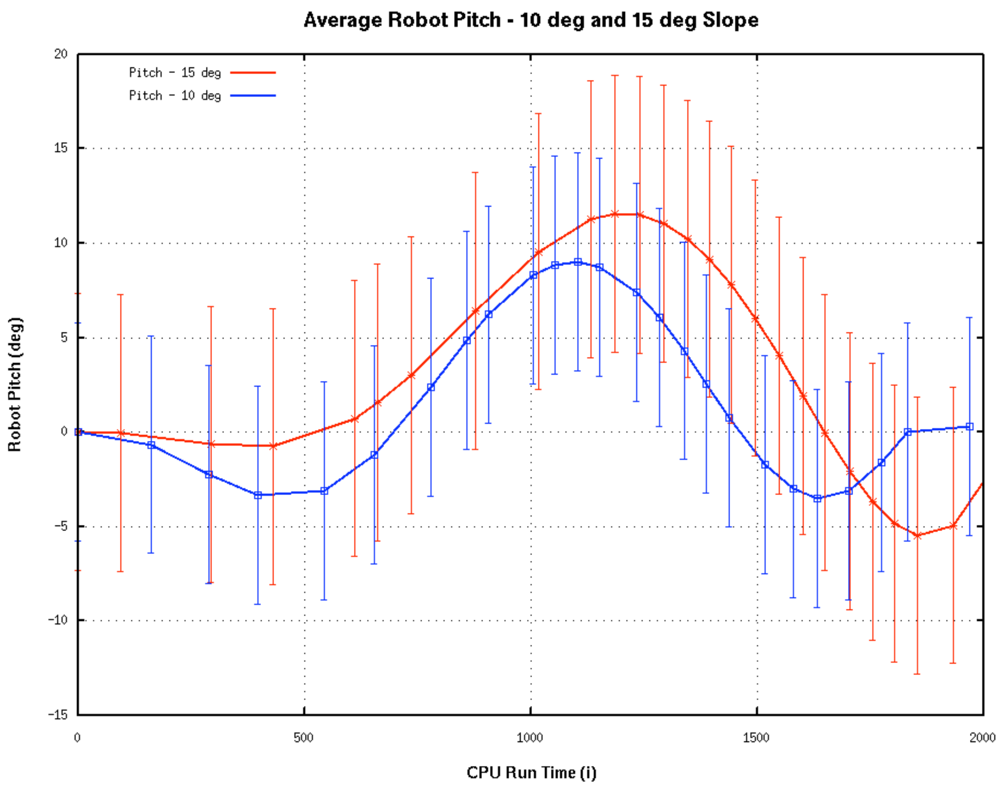

Figure 36. Average Robot Pitch - 10° and 15°

b. Tail Angle

Figure 37 represents the plot of a single run of the robot over the ramp. As in the plot of pitch data, the recorded tail angle data was very noisy, but the regression line generally corresponded with the desired results. The large and rapid swings in tail angle indicate that the tail response (see Equation 11) could be further damped. However, several attempts to do this resulted in very sluggish tail response for a variety of different proportionality constants.

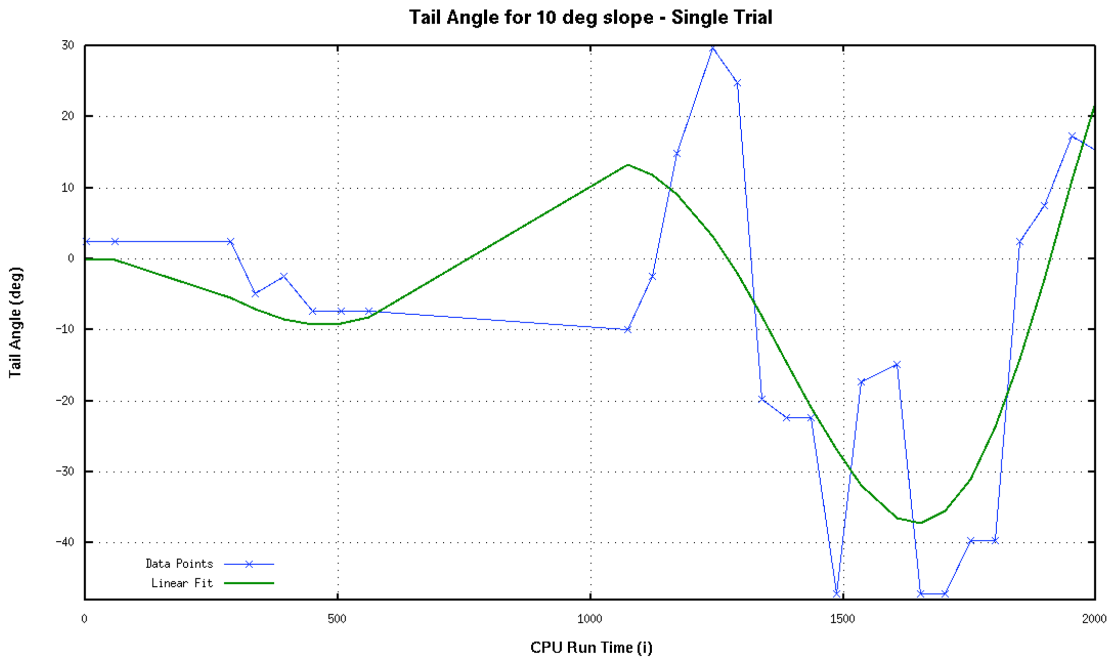

Figure 37. Tail Angle vs. Time - Data Points and Linear Fit for Single 10° Trial

Results for the tail angle, plotted in Figure 38 and Figure 39, were not as conclusive as the pitch results. For the majority of trials, the tail was observed to perform the desired characteristics of holding a fixed horizontal position on flat terrain, holding a positive angle during the initial ascent of the positive slope, lowering rapidly when pitch angle exceeded a set value, maintaining a negative angle on the down slope, and returning to a horizontal position following the ramp. This was especially true of the trials across the 10° slope. However, the regression plots showed that most of the output data was not consistent for the 15° slope, and individual trials produced vastly different results.

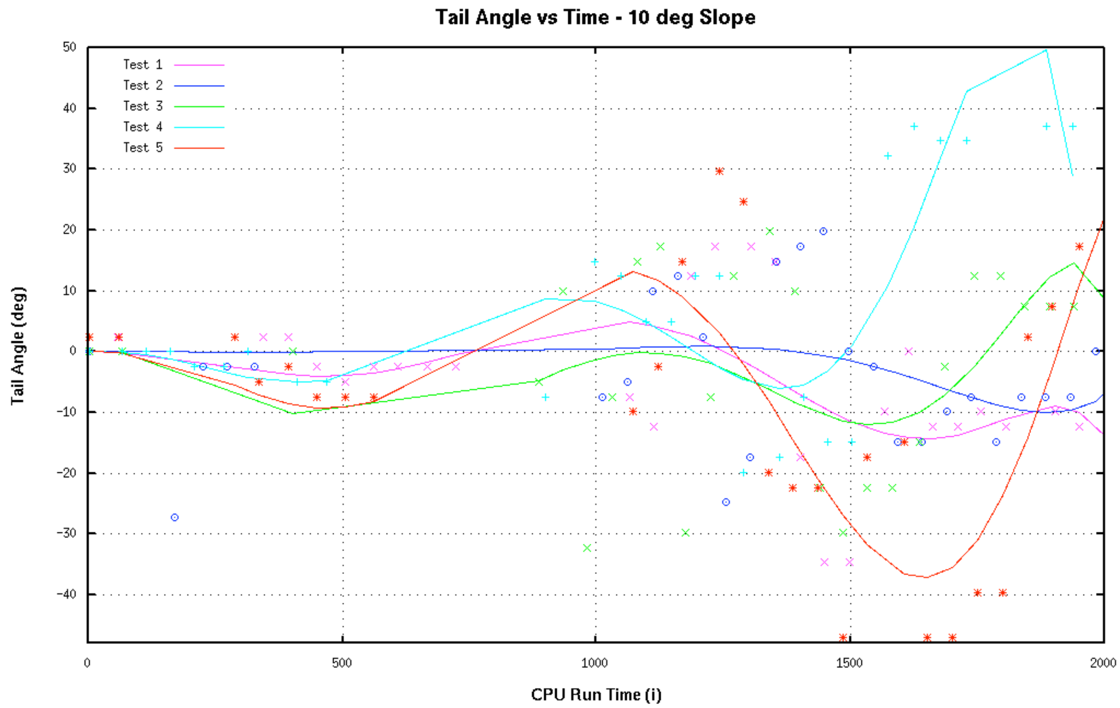

Figure 38. Tail Angle vs. Time - 10°

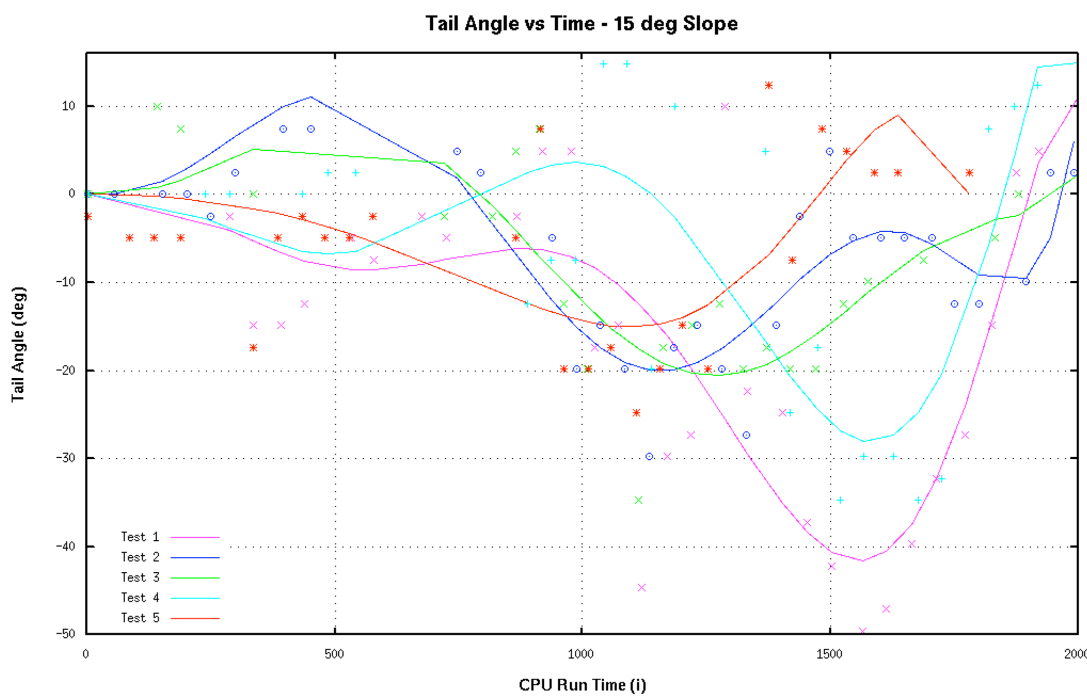

Figure 39. Tail Angle vs. Time - 15°

Comparing average results for both ramps in Figure 40, the 15° slope produced better results than the shallower ramp. This fact appears to be more an artifact of the method for averaging the data, and not representative of the results of the individual trials for that set which produced a wide variation in results for each trial. However, the standard deviation for the 15° data set was only 6% larger than that derived from the 10° data set.

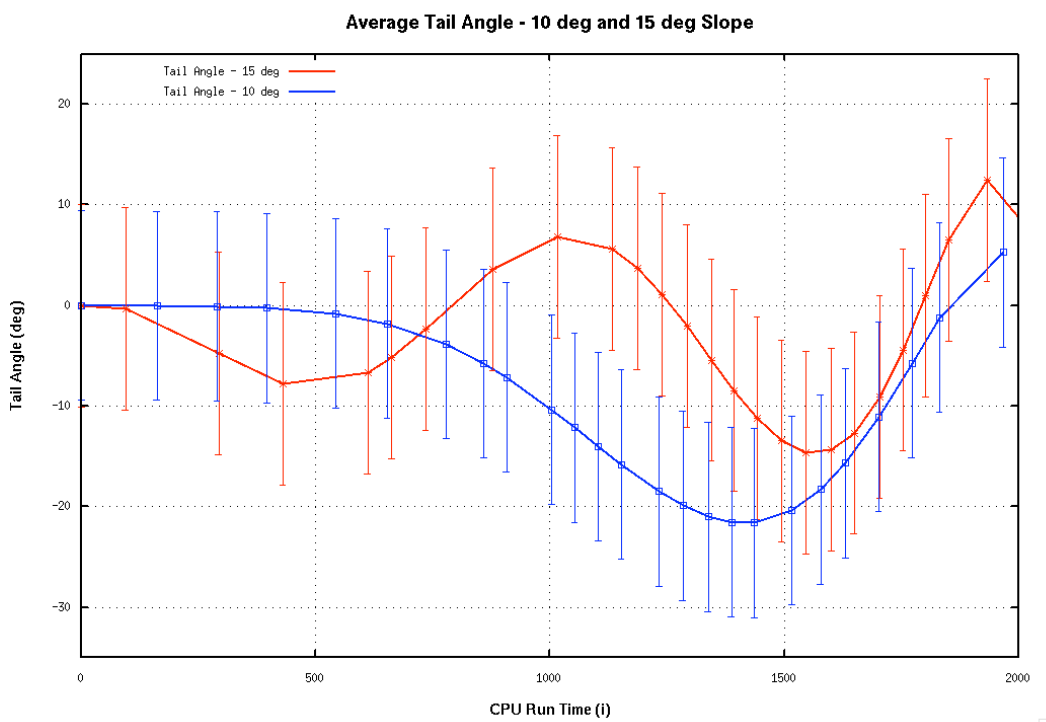

Figure 40. Average Tail Angle – 10° and 15°

c. *Robot Pitch vs. Tail Angle*

The final plot sets, Figure 41 and Figure 42, compare average values for each trial set. This produced some expected and some surprising results. Both plots illustrated the sudden change in tail angle as the robot pitch exceeded the critical value, exactly the desired result. This operation occurred near the maximum pitch angles for each slope. One unexpected result was that the average tail response for the 10° slope did not reproduce the desired operation as well as the 15° slope for the positive side of the incline. During the ascent of the 15° slope, the robot slowed significantly due to the greater torque requirements necessary to surmount the greater incline, giving it additional time to generate a response. However, both reacted correctly to angles in excess of their respective critical angles, lowering the tail to the boost position, and at identical rates. On the downward side of the ramp, the 15^{o} slope produced a significant overshoot of the horizontal in comparison to the 10^{o} slope. This was likely caused by the rapid and very large change in the difference between the pitch and tail angle, producing a high tail voltage in the difference function (see Equation 11) for a brief period. For both the pitch and tail angles, standard deviation for the 15^{o} slope was much larger than that found in the 10^{o} slope.

$$V_T = \frac{\theta_c - \theta_t}{c_2} + V_S = \frac{30^\circ - {}^{-}40^\circ}{35\frac{\text{deg}}{V}} + 2.5V = 4.5V \qquad (11)$$

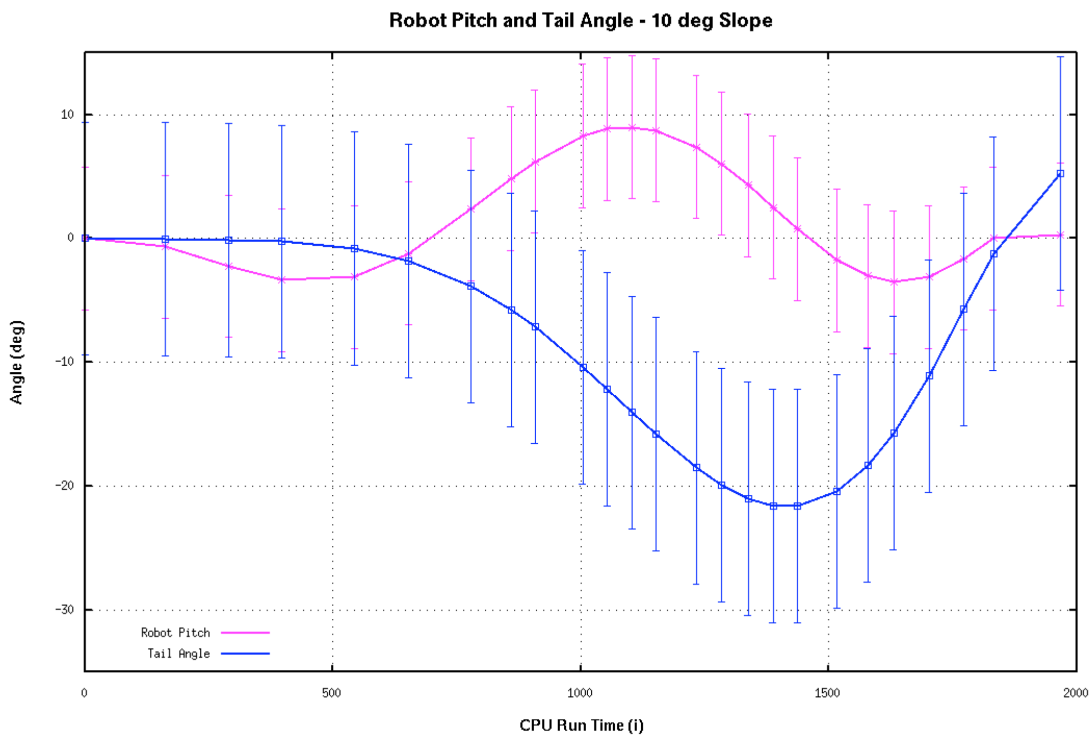

Figure 41. Average Robot Pitch and Tail Angle - 10°

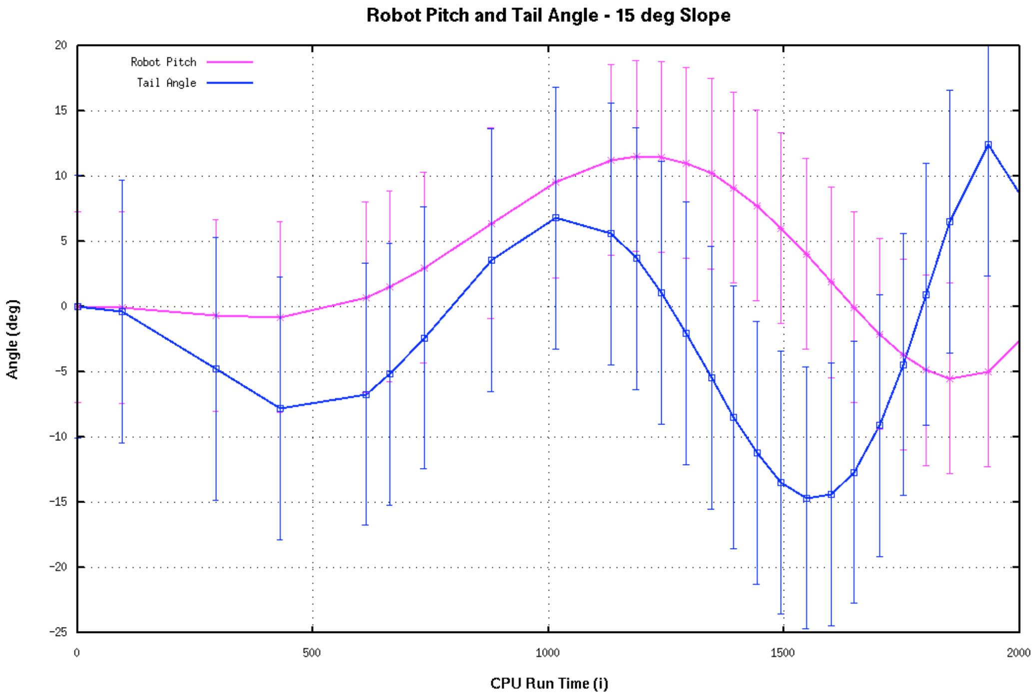

Figure 42. Average Robot Pitch and Tail Angle - 15°

V. RECOMMENDATIONS

The purpose of this thesis was to evaluate sensor and control components for integration into future work on the Pelican Whegs™ prototype currently under development at Case Western Reserve University. Robster proved that a tail mechanism could be incorporated into the Pelican Whegs™ design, and that this system could be controlled using a digital compass with tilt sensor and a variable resistance potentiometer to indicate tail position. Robster could effectively keep track of its pitch and maintain an appropriate attitude for its tail during walking and climbing operations. However, performance was better when climbing a shallower platform than a steeper one due to the underdamped response of the feedback loop. After evaluating this demonstration platform, some specific considerations were recommended for implementation in the prototype.

A. CONTROL ALGORITHM IMPROVEMENTS

Further work is needed to develop the specific control algorithm to be implemented in the Pelican Whegs™. The primary improvement to the tail control algorithm will need to come in damping or eliminating response to spurious sensor data. This can be accomplished by filtering sensor input data for transient results that might fall outside the surrounding data. A set of data would need to be stored in memory and the results compared for outliers. However, this would take a certain amount of computer time and would be highly reliant on the repetition frequency of the sensor queries. Tail reaction time to pitch changes would be protracted, but much improved position reliability would be obtained.

A new critical angle needs to be found for the tail "boost" response. Currently, this angle was chosen to match the parameters of the experimental setup. However, the critical angle for this response should occur when the robot's center of gravity causes it to high-center and flip over. This angle would need to be characterized specifically for the Pelican Whegs™ platform.

B. INCLINOMETER

A more responsive pitch indicator might be utilized to improve the positional awareness of the robot. The HMR 3000 compass was employed in this robot because it was already integrated into the navigation function of the robot and had a readily available two-axis tilt sensor. Yet, inconsistent pitch data during the dynamic tests was the primary source of error in the results. A solid-state MEMS inclinometer would likely provide better fidelity and a higher sampling rate than this. However, a different inclinometer was not acquired due to cost and lack of time to implement in this thesis.

C. TAIL ANGLE SENSOR

An optical shaft encoder might be used as an alternative to the variable resistance potentiometer. The potentiometer was very responsive to changes in inclination and results were easily interpreted by the BL2000. However, the measurement was subject to nonlinear errors due to noise in the source voltage. These variations introduced unpredictable results into several of the testing trials. A shaft encoder would output a fixed value for each given unit of rotation. Therefore, variations in the input voltage would not affect the results. This might reduce some of the inaccurate results that were recorded during the dynamic tests.

APPENDIX A. TAIL CONTROL DYNAMIC C CODE

```
/*----------------------------------------------------------------------
---

Courtney Holland
                        29MAY2009
ROBSTER THESIS RESULTS
Demonstration of Walking and Tail Over Obstacle
      1. Before starting, place Tail at 0 deg

      2. Tail Control - Turn ON:
            a. Compass
            b. Potentiometer
            c. Tail
            d. Motor Controller

      3. Drives forward for XX seconds

----------------------------------------------------------------------
--*/

#define READDELAY 15

#define MAX_SENTENCE 100

#memmap xmem
/*----------------------------------------------------------------------
-

   Serial Port Settings

--------------------------------------------------------------------

#define BINBUFSIZE 127

#define BOUTBUFSIZE 127

#define CINBUFSIZE 127

#define COUTBUFSIZE 127

/*-------------------------------------------------------------------

  Compass variables

-------------------------------------------------------------------*/

char dir_string[2];
int string_pos;
char input_char;
float curr_hdg;

char compass_sentence[MAX_SENTENCE];
```

```
int compass_error;

//Tilt test variables

char *first, *second, *third, *fourth;
float tilt;


const int compass_delay = 50; //mili-seconds to delay between compass
readings
const char init_str[] = "#BAD=11*7A\r\n";       //5 times per second
//const char init_str[] = "#BAD=15*7E\r\n";     //200 times per second

unsigned long compass_wait_time;

const int compass_timeout = 1;

int Compass_update;

/*-------------------------------------------------------------------

  New PID Variables

-------------------------------------------------------------------*/

int compconv;

const float P = 1;              // proportional coefficient (concrete)
const float I = 5;              // Integral coefficient     (concrete)
const float D = 3;              // differential coefficient (concrete)

int flag;                       // determines left or right turn or stop
int flagint;                    // integral counter

float insidevolts;              // voltage on side to which robot turns
float pScale;                   // proportional scaling term
float dScale;                   // differential scaling term
float iScale;                   // integral scaling term

int Error;                      // heading error +/- 180
int prevError;                  // heading error previous sample

/*-------------------------------------------------------------------

      CTRL  bools

-------------------------------------------------------------------*/

int man_ctrl;

/*-------------------------------------------------------------------

   Control Variables

-------------------------------------------------------------------*/
```

```
const float ERR_INNER_STOP = 90.0;        //Error(deg) that makes inner
track stop

const float ERR_INNER_REV = 180.0;        //Error(deg) that makes inner
track rev

const float PW_STOP = 2.50;                     //Pulse    width    that
results in stop command

const float PW_REV = 1.50;                            //Pulse      width
that results in max reverse (old 4.00)

const float PW_FWD = 3.25;                            //Pulse      width
that results in max forward (old 0.80)

float LeftSide, RightSide;                      //  wheel  control  for
manual control

const int Motor = 1;                                        //wheel
drive

const int Tail = 0;                                         //tail
drive

const int rt_ch = 0;                            //right side
const int lt_ch = 1;                            //left side

float rot, Tale;
      //Potentiometer measurements

const float T_FWD = 3.00;
const float T_STOP = 2.55;
const float T_REV = 1.00;
float T_MOVE;

/*-------------------------------------------------------------------

   Function Prototypes

-------------------------------------------------------------------*/

int compass_get_hdg(char sentence[MAX_SENTENCE]);

void msDelay (long sd);

unsigned long t0;

#define time 5

/*********************************************************************
********

                                 Main Function

**********************************************************************
*******/
```

```
main()

{

      int i, t;

      float diff, tilt1, level;

/*  -------------------------------------------------------------------
--

                                     Initializations

---------------------------------------------------------------------
*/

     brdInit();

/*  -------------------------------------------------------------------
--

                            Motor Initialization

---------------------------------------------------------------------
*/

     anaOutVolts(Motor, PW_STOP);
     anaOutVolts(Tail, PW_STOP);

     iScale=0;
     pScale=0;
     dScale=0;

     tilt1 = 10;
     t = 0;
     i = 0;

/*  -------------------------------------------------------------------
--
       Set flags

---------------------------------------------------------------------
*/

            man_ctrl = 1;
            Compass_update = 0;

/*--------------------------------------------------------------------
--

       Initialize Compass

---------------------------------------------------------------------
*/
```

```
            serBopen(9600);
            serBwrFlush();
            serBputs(init_str);

            rot = anaInVolts(0);
            msDelay(100);
            level = rot;

        while (1)
            {
//  ------------------------------------------------------------------
--

//                                    Compass Costatement

//

//  this is where we transmit the compass report to the GUI

//  ------------------------------------------------------------------
--

            costate
            {
                  waitfor (DelayMs(compass_delay));

                  serBrdFlush();

                  string_pos = 0;

                  input_char = serBgetc();
                  //find begining of sentence

                  compass_wait_time  =  SEC_TIMER  +  compass_timeout;
//timeout if compass not working

                  while (input_char != '$')
                  {
                        if (SEC_TIMER > compass_wait_time) abort;
                        input_char = serBgetc();
                        msDelay(READDELAY);
                  }
                  //read the sentence

                  while (input_char != '*' )
                  {
                        compass_sentence[string_pos] = input_char;
                        string_pos++;

                        if(string_pos == MAX_SENTENCE)
                              string_pos  =  0;  //reset  string  if  too
large

                        input_char = serBgetc();
                        msDelay(READDELAY);
                  }
```

```
                  compass_sentence[string_pos] = 0; //add null

                  //Tilt string parse
                  first = strtok(compass_sentence, ",");
                  second = strtok(NULL, ",");
                  third = strtok (NULL, ",");
                  fourth = strtok (NULL, ",");

            }//end of compass

//  ------------------------------------------------------------------
--
//                                  Tail Control Costatement
//  ------------------------------------------------------------------
--

costate
{
      i++;
      tilt = atof(fourth)+ 4.0;
      rot = anaInVolts(0);

      Tale = 500*(rot - level);
      diff = tilt - Tale;
      T_MOVE = diff/45 + T_STOP;

//Move only with tail angle between +30 and -40
      if(Tale <= 30 && Tale >= -40){
            if (tilt >= -1.0 && tilt <= 1.0){
//For Pitch within 1 deg of 0 and tail within 5, STOP
                  if(Tale <= 5 && Tale >= -5)
                        anaOutVolts(Tail, T_STOP);
                  else anaOutVolts(Tail, T_MOVE);
            }
//For Pitch < 15 deg, Tail follows pitch
            if (tilt > 3.0 && tilt <= 12.0)
                        anaOutVolts(Tail, T_MOVE);
//For Pitch > 15 deg, Tail moves DOWN
            if (tilt > 15.0)
                  anaOutVolts(Tail, T_REV);
      //For Pitch < -3 deg, Tail follows pitch
            if (tilt < -3.0)
                  anaOutVolts(Tail, T_MOVE);
      }
      else if (Tale > 30){                      //Move   Tail   DOWN   at
stop
                  anaOutVolts(Tail, 2.3);
                  }
      else if (Tale < -40){                     //Move Tail UP at stop
                  anaOutVolts(Tail, 2.7);
                  }
//Print only for different Pitch
      if (tilt1 >= tilt + 0.2 || tilt1 <= tilt - 0.2){
            printf("%d \t %.4f \t %.4f \n", i, tilt, Tale);
      }
                  tilt1 = tilt;
} //end of tail statement
```

```
costate
{
      anaOutVolts(Motor, PW_STOP);
      waitfor (DelayMs(10000));
      anaOutVolts(Motor, PW_FWD);
      waitfor (DelayMs(10000));
} //end of motor drive statement
}//while(1)

}//main

/*               START                FUNCTION                DESCRIPTION
********************************************
compass_get_hdg

SYNTAX:      int compass_get_data();
KEYWORDS:       compass
DESCRIPTION:   Parses a sentence to extract heading data.
            This function is able to parse HPR data from a
            HMR3000 Digital Compass
PARAMETER1: sentence - a string containing a line of HPR data
RETURN VALUE:      0 - success
            -1 - parsing error
            -2 - heading marked invalid

SEE ALSO:
END                                                           DESCRIPTION
*******************************************************/
int compass_get_hdg(char sentence[MAX_SENTENCE])
{
      auto int i;
      char *err,*hdg,*type;
      char error;

      if(strlen(sentence) < 4)
            return -1;

      if(strncmp(sentence, "$PTNTHPR", 8) == 0)
      {
            //parse hpr sentence
            type = strtok(sentence, ",");
            hdg = strtok(NULL, ",");
            err = strtok (NULL, ",");
            if(hdg == NULL)
                  return -2;
            //pull out data
            curr_hdg = atof(hdg);

            error = (int)err;
            if (strncmp(&error, "N", 1) == 0)
                  return -2;
      }
      else
            return -1;
      return 0;
}
```

```
void msDelay (long sd)
{
      unsigned long t1;
      t1 = MS_TIMER;

      for (t1 = MS_TIMER; MS_TIMER < (sd + t1); );
}
```

APPENDIX B. EMBEDDED DYNAMIC C CODE

```
/**********************************************************************
LT Courtney Holland

I.  AGBOT Code v2.0 - 18MAR2009
    Changes:
    Eliminated all references to Bigfoot Arm and Thermopile
    Commented out sonar

II. AGBOT Code v2.1 - 20MAR2009
    Changes:
    Added accelerometer costatement

III. ROBSTER Code v2.2 - 09APR2009
    Working Code for Comms, GPS, Compass, IR, Navigation
    Changes:
    Deleted accelerometer costatement
    Added Serial compass

IV. ROBSTER Code v2.3 - 13APR2009
    Good working code for all mechanical components
    Changes:
    Added function for tail
    Working on manual tail control
    To Do:
    Navigation improvements

V. ROBSTER Code v2.4 - 18MAY2009
    Changes:
    Both wheel set now on DAC1 under Motor
    Tail now on DAC0 under Tail
    Added:
    Potentiometer for ADC0 as Rotation Sensor

**********************************************************************/
/**********************************************************************
BL2000 CONNECTIONS

   Motor Controllers
   DAC1    <--->       //left side wheels
   DAC0    <--->       //right side wheels

   Tail Controller
   OUT0           BREAK             BLACK
   OUT1           FWD/REV           YELLOW

   GPS Serial Communicataions
   TX2            BROWN
   RX2            BROWN WITH RED
   GROUND   BLACK

   Compass Serial Communicataions
   TX1            GREY
   RX1            GREY
```

```
   GROUND   BLACK

   IR Ranger
   ADC3           WHITE       //center
*********************************************************************/

#define READDELAY 15

#define MAX_SENTENCE 100

/*------------------------------------------------------------------
  Network Settings
------------------------------------------------------------------*/

#define MY_IP_ADDRESS         "192.168.1.2"                 //BL2000
adress
#define INTERFACE_ADDRESS     "192.168.1.3"                 //Laptop
address
#define MY_NETMASK                "255.255.255.0"
#define MY_GATEWAY                "192.168.1.1"             //Router
address

#define MAN_PORT 4001        // receives manual control data
#define WP_PORT 4002         // receives waypoint data
#define GPS_PORT 4003        // sends gps data
#define COMPASS_PORT 4004    //sends compass data
#define ERROR_PORT 4005      // sends error reports

#use "dcrtcp.lib"

#memmap xmem

/*---------------------------------------------------------------------
   Serial Port Settings
---------------------------------------------------------------------*/

#define BINBUFSIZE 127
#define BOUTBUFSIZE 127
#define CINBUFSIZE 127
#define COUTBUFSIZE 127
/*-----------------------------------------------------------------
  GPS Variables
------------------------------------------------------------------*/

double curr_lat;
double curr_lon;

const int xmit_delay = 100;

char sentence[MAX_SENTENCE];
char dir_string[2];

typedef struct {
    int lat_degrees;
    int lon_degrees;
    double lat_minutes;
    double lon_minutes;
```

```
    char lat_direction;
    char lon_direction;
} GPSPosition;

GPSPosition current_pos;   // Declare new GPSPosition variable

int gps_error, gps_error_count;

const float pi = 3.14159;

const char GPS_Reset[]="$PGRMI,,,,,,,R\r\n";       // Reset Unit
const char GPS_Sent_Clr[]="$PGRMO,,2\r\n";          // Clear all output
sentences
const char GPS_GGA_Enable[]="$PGRMO,GPGGA,1\r\n";   // Enable the GGA
sentence

unsigned long gps_wait_time;
const int gps_timeout = 1;

int string_pos;
char input_char;

/*-------------------------------------------------------------------
  Compass variables
-------------------------------------------------------------------*/

float curr_hdg;
char compass_sentence[MAX_SENTENCE];
int compass_error;

//Tilt test variables
char *first, *second, *third, *fourth;
float tilt;

const int compass_delay = 50; //mili-seconds to delay between compass
readings
const char init_str[] = "#BAD=11*7A\r\n";       //5 times per second
//const char init_str[] = "#BAD=15*7E\r\n";     //200 times per second

unsigned long compass_wait_time;
const int compass_timeout = 1;

int Compass_update;

/*-------------------------------------------------------------------
  New PID Variables
-------------------------------------------------------------------*/

int compconv;
const float P = 1;             // proportional coefficient (concrete)
const float I = 5;             // Integral coefficient     (concrete)
const float D = 3;             // differential coefficient (concrete)
int flag;                      // determines left or right turn or stop
int flagint;                   // integral counter
float insidevolts;             // voltage on side to which robot turns
float pScale;                  // proportional scaling term
float dScale;                  // differential scaling term
```

```
float iScale;                    // integral scaling term
int Error;                       // heading error +/- 180
int prevError;                   // heading error previous sample

/*--------------------------------------------------------------------
   Communications
--------------------------------------------------------------------*/

word status, port;
longword host;

udp_Socket compass_data, gps_data, error_data;
sock_type wp_data, man_data;

char cmdBuf[2048];
char cmdstr[20], *cmdptr;

char wptBuf[2048];
char wptstr[500], *wptptr, *wpttmp;

char error_buf[200];

/*--------------------------------------------------------------------
  Nav Variables
--------------------------------------------------------------------*/

const float brg_error = 5.0;  //Allowable Bearing Error
const float rng_error = 5.0;  //Allowable range error (in yards)

float lat_diff, lon_diff;     //The amount of Lat/Long (in Seconds and
                                  //         Decimal   Seconds   between
Bender's current
                                  //    position and the next waypoint

float theta;                      //Angle (deg) from True North to next
waypoint
float hdg_error;                //Angle (deg) from current heading to next
                               //    waypoint

float new_hdg;                //The Desired heading in degrees

double rng, temp_rng;           // Range and temporary range (in yards)

double brg;                   //Don't know what this is for

/*--------------------------------------------------------------------
  Waypoint Variables
--------------------------------------------------------------------*/

typedef struct
{
    double lat;
    double lon;
    char  action;
}WP;                              // Define WP structure

WP waypoints[10];                 // stores the list of waypoints
```

```
char passed_waypoint[10];           //  Stores action value for passed
waypoints
int curr_wp;                    // current wp
char *temp;
char *temp_lat, *temp_lon;
char *temp_action;

double lat, lon, wlat, wlon;

/*-------------------------------------------------------------------
    CTRL    bools
-----------------------------------------------------------------*/

int man_ctrl;
int GPS_updated;

/*-------------------------------------------------------------------
   Control Variables
-----------------------------------------------------------------*/

const float ERR_INNER_STOP = 90.0; //Error(deg) that makes inner track
stop
const float ERR_INNER_REV = 180.0; //Error(deg) that makes inner track
rev

const  float  PW_STOP  =  2.50;     //Pulse  width  that  results  in  stop
command
const float PW_REV = 1.5;     //Pulse width that results in max reverse
(old 4.00)
const float PW_FWD = 3.5;           //Pulse  width  that  results  in  max
forward (old 0.80)

float LeftSide, RightSide;      // wheel control for manual control
const int Motor = 1;          //wheel drive

const int Tail = 0;                 //tail drive
const int rt_ch = 0;       //right side
const int lt_ch = 1;       //left side
float rot, Tale, level;                  //Potentiometer measurements
float T_MOVE;                            //Tail Proportional equation

const float T_FWD = 3.00;
const float T_STOP = 2.55;
const float T_REV = 1.50;

/*-------------------------------------------------------------------
   Function Prototypes
-----------------------------------------------------------------*/

int compass_get_hdg(char sentence[MAX_SENTENCE]);
int gps_get_position(GPSPosition *newpos, char *sentence);
int gps_parse_coordinate(char *coord, int *degrees, float *minutes);
int ERROR_function(float new_hdg);
void msDelay (long sd);

void CommStart(void);
```

```
unsigned long t0;
#define time 5

/**********************************************************************
                                Main Function
**********************************************************************/

main()
{

    int i, t;
    float diff;

/* -------------------------------------------------------------------
                                     Initializations
 -------------------------------------------------------------------*/
     brdInit();
     CommStart();

/* -------------------------------------------------------------------
                            Motor Initialization
---------------------------------------------------------------------*/

      anaOutVolts(Motor, PW_STOP);
     anaOutVolts(Tail, PW_STOP);

      new_hdg=0;
      iScale=0;
      pScale=0;
      dScale=0;

/* -----------------------------------------------------------------
     Set flags
---------------------------------------------------------------------*/

      man_ctrl = 1;
      GPS_updated = 0;
      Compass_update = 0;


/*------------------------------------------------------------------
     Initialize Compass
---------------------------------------------------------------------*/

      serBopen(9600);
      serBwrFlush();
      serBputs(init_str);

/*-------------------------------------------------------------------
     Initialize GPS
-------------------------------------------------------------------*/

      serCopen(9600);            // Open serial port C
      serCwrFlush();             // Flush serial port C Buffer
      serBputs(GPS_Reset);       // Send Reset signal to GPS Receiver
      serBputs(GPS_Sent_Clr);    // Send Clear signal to GPS Receiver
      serBputs(GPS_GGA_Enable); // Send GGA Sentence enable signal
```

```
                                    //    (position info)
/*-------------------------------------------------------------------
     Initialize Tail
--------------------------------------------------------------------*/

      rot = anaInVolts(0);
      msDelay(100);
      level = rot;


  while (1)
    {
      tcp_tick(NULL);
//-------------------------------------------------------------------
//                              Receive Manual Control Data
// ------------------------------------------------------------------

   costate
    {
          waitfor(sock_recv( &man_data, cmdstr, (word)sizeof(cmdstr)));

            //Tokenize the string and convert to integers
          LeftSide = atof(strtok(cmdstr, " "));
          RightSide = atof(strtok(NULL, "/n"));

         anaOutVolts(Motor, RightSide);
//         anaOutVolts(Tail, LeftSide);

         if (!man_ctrl)
            {
            sprintf(error_buf,  "$Manual  control  data  recieved...IN
MANUAL CTRL\n", curr_wp);
             sock_puts(&error_data, error_buf);
           }
         //Update the flags
         man_ctrl = 1;

    }       // man data costate

// -----------------------------------------------------------------
//                                     Compass Costatement
//  this is where we transmit the compass report to the GUI
// ------------------------------------------------------------------

      costate
      {
            waitfor ( DelayMs(compass_delay));

            serBrdFlush();
            string_pos = 0;

            input_char = serBgetc();

            //find begining of sentence
            compass_wait_time = SEC_TIMER + compass_timeout; //timeout
if compass not working
```

```
			while (input_char != '$')
			{
				if (SEC_TIMER > compass_wait_time) abort;
				input_char = serBgetc();
				msDelay(READDELAY);
			}

			//read the sentence
			while (input_char != '*' )
			{
				compass_sentence[string_pos] = input_char;
				string_pos++;

				if(string_pos == MAX_SENTENCE)
					string_pos = 0; //reset string if too large

				input_char = serBgetc();
				msDelay(READDELAY);
			}

			compass_sentence[string_pos] = 0; //add null
			sock_puts(&compass_data, compass_sentence);

			//Tilt string parse
			first = strtok(compass_sentence, ",");
			second = strtok(NULL, ",");
			third = strtok (NULL, ",");
			fourth = strtok (NULL, ",");
			if((compass_error =compass_get_hdg(compass_sentence)) != 0)
			{
			sprintf(error_buf, "$Compass Error: %d\n",compass_error);
				sock_puts(&error_data, error_buf);
			}
			else
			{
				Compass_update = 1;
			}
		}//end of compass

// ---------------------------------------------------------------------
//                                  Tail Control Costatement
// ---------------------------------------------------------------------

		costate
		{
			i++;
			tilt = atof(fourth)+ 4.0;
			rot = anaInVolts(0);

			Tale = 500*(rot - level);
			diff = tilt - Tale;
			T_MOVE = diff/45 + T_STOP;

			//Move only with tail angle between +30 and -30
			if(Tale <= 30 && Tale >= -40){
				if (tilt >= -1.0 && tilt <= 1.0){
//For Pitch within 1 deg of 0 and tail within 5, STOP
```

```
                        if(Tale <= 5 && Tale >= -5)
    anaOutVolts(Tail, T_STOP);
                        else anaOutVolts(Tail, T_MOVE);
                  }
                  if (tilt > 3.0 && tilt <= 12.0){
//For Pitch < 15 deg, Tail follows pitch
                        anaOutVolts(Tail, T_MOVE);
                  }
                  if (tilt > 12.0){
    //For Pitch > 15 deg, Tail moves DOWN
                        anaOutVolts(Tail, T_REV);
                  }
                  if (tilt < -3.0){
    //For Pitch < -3 deg, Tail follows pitch
                        anaOutVolts(Tail, T_MOVE);
                  }
            }
            else if (Tale > 30){
    //Move Tail DOWN at stop
                  anaOutVolts(Tail, 2.3);
            }
            else if (Tale < -40){
    //Move Tail UP at stop
                  anaOutVolts(Tail, 2.7);
            }
            tilt1 = tilt;
      } //end of tail statement


// ----------------------------------------------------------------------
//                              Receive Waypoint Data
// ----------------------------------------------------------------------

        costate
      {
      waitfor(sock_recv( &wp_data, wptstr, (word) sizeof(wptstr)));

            //find begining of string
            wptptr = wptstr;        //assign a pointer

         while (*wptptr != '$') //Step through until begining of string
         wptptr++;
            wptptr++;
            //tokenize
            temp_lat = strtok(wptptr, " ");
            temp_lon = strtok(NULL, " ");
            temp_action = strtok(NULL, " ");

         for (i = 0; i < 10; i++)
            {
              if ((temp_lat == 0 && temp_lon ==0) ||
                   waypoints[i].action != "P")
                 {
                   waypoints[i].lat = strtod(temp_lat, NULL);
                      waypoints[i].lon = strtod(temp_lon, NULL);
                      waypoints[i].action = *temp_action;
                 } //End if Statement
```

```
              temp_lat = strtok(NULL, " ");
                 temp_lon = strtok(NULL, " ");
                 temp_action = strtok(NULL, " ");
               }//End for loop

            curr_wp = 0;
// Resets current WP to 1st waypoint. If this is an
// update to waypoints, Nav will increment curr_wp until
// a good waypoint is there.
            //update the flags
            man_ctrl = 0;

            sprintf(error_buf,
                    "$WP's recieved.  In AUTO NAV and preceeding to WP
%d\n",
                    curr_wp);
            sock_puts(&error_data, error_buf);

         // these commands make the robot start moving forward before
         // trying to find the heading to avoid em surge near compass
         anaOutVolts(rt_ch, PW_FWD);
         anaOutVolts(lt_ch, PW_FWD);
         msDelay(500);

      }//End Waypoint Costatement


// -----------------------------------------------------------------------
//                                        GPS
// -----------------------------------------------------------------------
     costate
        {
//          waitfor (DelaySec(gps_delay));

            serCrdFlush();
            string_pos = 0;
         input_char = serCgetc();

         //timeout if gps not sending data
         gps_wait_time = SEC_TIMER + gps_timeout;
         while (input_char != '$')
            {
                  if (SEC_TIMER > gps_wait_time) abort;
                  input_char = serCgetc();
                  msDelay(READDELAY);
            }
         //find begining of sentence
         while ((input_char != '\r') && (input_char !='\n'))
            {
                  sentence[string_pos] = input_char;
                  string_pos++;
                  if(string_pos == MAX_SENTENCE)
                        string_pos = 0; //reset string if too large

                  input_char = serCgetc();
                  msDelay(READDELAY);
```

```
            }

            sentence[string_pos] = 0;

            sock_puts(&gps_data, sentence);
            //tcp_tick(NULL);

            gps_error = gps_get_position(&current_pos, sentence);

            if ((gps_error == 0) || (gps_error == -1))
                  gps_error_count = 0;
            else
            {
                  gps_error_count ++;
            }
                  GPS_updated = 1;
                  curr_lat=(current_pos.lat_degrees                    +
(current_pos.lat_minutes/60));
                  curr_lon=(current_pos.lon_degrees                    +
(current_pos.lon_minutes/60));

//          }
      }//GPS

// ---------------------------------------------------------------------
//                                    Navigation
//   Passes heading error and range to CTRL costatement and uses error
function
//  to determine error from new_hdg and curr_heading
// ---------------------------------------------------------------------
    costate
       {
         if (man_ctrl)
           {
            abort;
           }
         if (GPS_updated)     //Navigates to new waypoint
           {
            //if(1)            {
            // give fake lat/long
            //curr_lat = 36.595;
            //curr_lon = 121.8753;

            lat = 60 * curr_lat;              // converts latitude into
                                         // Minutes and decimal minutes
            lon = 60 * curr_lon;             // converts longitude into
                                         // Minutes and decimal minutes
            wlat = 60 * waypoints[curr_wp].lat;
// Converts waypoint values
            wlon = 60 * waypoints[curr_wp].lon;  // to decimal minutes
// replaced by following line for simplicity
            rng =sqrt((4000000*(wlat-lat)*(wlat-lat))+
                      (2560000*(wlon-lon)*(wlon-lon)));
            if (rng <= rng_error)
// When close enough to waypoint, action
//   code takes effect and next waypoint is loaded
              {
```

```
                switch (waypoints[curr_wp].action)
                        {
                     case 'T':                  //Go to next waypoint
                       {
                         passed_waypoint[curr_wp] = 'T';
                                    // Stores action code in temp array
                         waypoints[curr_wp].action = 'P';
                                      // Changes action code to indicate
                                              // WP has been passed
                         sock_puts(&error_data,  "$Proceeding  to  next
WP\n");

                         curr_wp++;
                         while ((waypoints[curr_wp].lat == 0) &&
                                (waypoints[curr_wp].lon == 0))
                            {     //checks for valid WP
                              curr_wp++;
                                 if (curr_wp == 10)
                                {
                                 sock_puts(&error_data,  "$No  Valid  WP
Found\n");

                                 tcp_tick(NULL);
                                 man_ctrl = 1;
                                 abort;
                                }//End if
                            }//End while
                         break;
                       } //End case 'T'

                     case 'H':               //Start from beginning again
                                {
                       for (i = 0;i < 10;i++)
//Reloads prior action codes
                            {
                              waypoints[i].action = passed_waypoint[i];
                            }
                       sock_puts(&error_data,"$Proceeding  back  to  home
WP. \n");
                                curr_wp = 0;

                                while ((waypoints[curr_wp].lat == 0) &&
                                                (waypoints[curr_wp].lon
== 0))
                            {                  //checks for valid WP
                              curr_wp++;

                                            if (curr_wp == 10)
                                {
                                sock_puts(&error_data,  "$No  Valid  WP
Found\n");

                                tcp_tick(NULL);
                                man_ctrl = 1;
                                abort;
                                }//End if
                               }//End while

                                   break;
                       }//End case 'H'
```

```
                    case 'S':              //Stop
                      {
                        anaOutVolts(rt_ch, PW_STOP);
                                          anaOutVolts(lt_ch,  PW_STOP);
//Stops Bigfoot

                                 for (i = 0; i < 10; i++)
//Clears the Waypoint array
                        {
                             waypoints[i].lat = 0;
                             waypoints[i].lon = 0;
                             waypoints[i].action='T';
                         }//End for loop

                         sock_puts(&error_data,
                                         "$Destination        Achieved,
Waypoints cleared\n");
                         tcp_tick(NULL);
                         man_ctrl = 1;
                         abort;
                               }//End case 'S'

                    case 'C':  //Turn in a circle then proceed to next
                               // WP?????   this could be a problem
                               {
                                   curr_wp++;
                         while ((waypoints[curr_wp].lat == 0) &&
                                         (waypoints[curr_wp].lon == 0))
                                     {            //checks for valid WP
                              curr_wp++;
                     if (curr_wp == 10)
                     {
                       sock_puts(&error_data, "$No Valid WP Found\n");
                                tcp_tick(NULL);
                                man_ctrl = 1;
                                abort;
                              }// End if
                               }// End while
                                  break;
                            }//End case 'C'
                       case 'P': // Check for passed waypoints
                      {
                         curr_wp++;     // Bigfoot ignores this point
                                        // and goes to next one
                         while ((waypoints[curr_wp].lat == 0) &&
                                         (waypoints[curr_wp].lon == 0))
                           {                  // Checks for valid WP
                              curr_wp++;
                      if (curr_wp == 10)
                      {
                       sock_puts(&error_data, "$No Valid WP Found\n");
                                tcp_tick(NULL);
                                man_ctrl = 1;
                                abort;
                                }//End if
                           }//End while
```

```
                        break;
                      }//End case 'P'

              default: //Indicates and invalid action code
             {
             sprintf(error_buf, "$Invalid action for WP # %d\n",
                          curr_wp);
                             sock_puts(&error_data, error_buf);
                   tcp_tick(NULL);

                    anaOutVolts(rt_ch, PW_STOP);
                   anaOutVolts(lt_ch, PW_STOP);
                       man_ctrl = 1; abort;
                         }//End default case
                  }//End Switch
          if (curr_wp > 9)                    // Should not be here.
                                      // Action for last WP invalid.
         {
             anaOutVolts(rt_ch, PW_STOP);
             anaOutVolts(lt_ch, PW_STOP);
                 man_ctrl = 1;
           sock_puts(&error_data, "$Invalid action for wp 9\n");
             tcp_tick(NULL);
             abort;
           }//End if (curr_wp>9)
      }//End if (rng < rng_error)
      // Calculate new heading if range not within error
        // 3600 converts lat_diff and lon_diff to decimal seconds
          lat_diff = 3600 * (waypoints[curr_wp].lat-curr_lat);
          lon_diff = 3600 * (curr_lon - waypoints[curr_wp].lon);
          // determine theta in degrees
          theta = atan((lat_diff) / (lon_diff)) * (180 / pi);
          // waypoint located in positive y-axis
          if ((lon_diff == 0) && (lat_diff > 0))
              new_hdg = 0;
          // waypoint is located in negative y-axis
          else if ((lon_diff == 0) && (lat_diff < 0))
              new_hdg = 180;
          // waypoint is located in positive x-axis
          else if ((lon_diff > 0) && (lat_diff == 0))
              new_hdg = 90;
          // waypoint is located in negative x-axis
          else if ((lon_diff < 0) && (lat_diff == 0))
              new_hdg = 270;
          // waypoint is located in the first or fourth quadrant
          // (0-90 or 270-0)
          else if (lon_diff > 0)
              new_hdg = 90 - (int)(theta);
          // waypoint is located in the second or third quadrant
          // (90-180 or 180-270)
          else if (lon_diff < 0)
              new_hdg = 270-(int)(theta);
          hdg_error = ERROR_function(new_hdg);
          tcp_tick(NULL);
         }// End if (GPS_updated)
        }// End NAV costate
//*********************************************************************
```

```
//                              PID Controls
//*********************************************************************

      costate
         {
          waitfor(!man_ctrl);

       if (hdg_error <= 5.0 && hdg_error >= -5.0)
            {
           hdg_error = 0.0;
           flag = 2;
         }

       if ((hdg_error >= 180.0) ||
          ((hdg_error > -180.0) && (hdg_error < 0.0)))
         {
           flag = 1;
         }
       else
         {
           flag = 0;
         }
       //calculate proportional scale constant
       Error = (int)(fabs(hdg_error));

       if (Error > 180)
         {
           Error = 360 - Error;
         }

       pScale = (Error*(0.008333));

       dScale = ((Error-prevError)*(0.00833));

       iScale = pScale + iScale;

       if(flagint > 40)
         {
          iScale = 0.0;
         }

       flagint++;

       prevError = Error;

     if(!(hdg_error == 0.0))
           {
          insidevolts = (P * pScale) + (I * iScale) + (D * dScale);
                     //Do not send more than we put out
          if(insidevolts > PW_STOP)
            {
              insidevolts = 2.35;
            }

          if(flag == 0) // turn right
               {
                 anaOutVolts(rt_ch, insidevolts);
```

```
                  anaOutVolts(lt_ch, PW_FWD);
                }

           if(flag == 1) // turn left
                {
                  anaOutVolts(rt_ch, PW_FWD);
                  anaOutVolts(lt_ch, insidevolts);
                }
             }//ends if for heading error not = 0
             else
             {
      // send the right voltages to the wheels if no heading error
      //  and the range is greater than the delta
            anaOutVolts(rt_ch, PW_FWD);
            anaOutVolts(lt_ch, PW_FWD);
             }
     }//end PID costate

}//while(1)


}//main

/*                START                 FUNCTION                DESCRIPTION
********************************************
compass_get_hdg

SYNTAX:     int compass_get_data();

KEYWORDS:      compass

DESCRIPTION:   Parses a sentence to extract heading data.
      This function is able to parse HPR data from a
      HMR3000 Digital Compass

PARAMETER1: sentence - a string containing a line of HPR data

RETURN VALUE:      0 - success
      -1 - parsing error
      -2 - heading marked invalid

SEE ALSO:

END                                                             DESCRIPTION
**********************************************************/

int compass_get_hdg(char sentence[MAX_SENTENCE])
{
    auto int i;
    char *err,*hdg,*type;
    char error;

    if(strlen(sentence) < 4)
      return -1;

    if(strncmp(sentence, "$PTNTHPR", 8) == 0)
    {
```

```
      //parse hpr sentence
      type = strtok(sentence, ",");
      hdg = strtok(NULL, ",");
      err = strtok (NULL, ",");
      if(hdg == NULL)
            return -2;

      //pull out data
      curr_hdg = atof(hdg);

      error = (int)err;
      if (strncmp(&error, "N", 1) == 0)
            return -2;

    }
    else
      return -1;

    return 0;
}

//*********************************************************************
//                        gps_parse_coordinate
//
// Parses GPS position data
//
//PARAMETERS:     coord - contains N/S, E/W
//             degrees, minutes - positional information
//
//RETURN VALUE:    0 - success (xxxxx.xxxx minutes)
//                      -1 - parsing error
//
//*********************************************************************

nodebug  int  gps_parse_coordinate(char  *coord,  int  *degrees,  float
*minutes)
{
    auto char *decimal_point;
    auto char temp;
    auto char *dummy;

    decimal_point = strchr(coord, '.');
    if(decimal_point == NULL)
      return -1;
    temp = *(decimal_point - 2);
    *(decimal_point - 2) = 0; //temporary terminator
    *degrees = atoi(coord);
    *(decimal_point - 2) = temp; //reinstate character
    *minutes = strtod(decimal_point - 2, &dummy);
    return 0;
}

/*               START                FUNCTION                 DESCRIPTION
********************************************
gps_get_position
```

```
SYNTAX:                  int  gps_get_position(GPSPositon  *newpos,  char
*sentence);

KEYWORDS:      gps

DESCRIPTION:   Parses a sentence to extract position data.
                         This  function  is  able  to  parse  any  of  the
following
                         GPS sentence formats: GGA

PARAMETER1:    newpos - a GPSPosition structure to fill
PARAMETER2:       sentence - a string containing a line of GPS data
                         in NMEA-0183 format

RETURN VALUE:  0 - success
                         -1 - not differential
                         -2 - sentence marked invalid
                         -3 - parsing error

SEE ALSO:

END                                                              DESCRIPTION
***********************************************************/

//can parse GGA
nodebug int gps_get_position(GPSPosition *newpos, char *sentence)
{
    auto int i;

    if(strlen(sentence) < 4)
      return -3;
    if(strncmp(sentence, "$GPGGA", 6) == 0)
    {
      //parse GGA sentence
      for(i = 0;i < 11;i++)
      {
            sentence = strchr(sentence, ',');
            if(sentence == NULL)
                  return -3;
            sentence++; //first character in field
            //pull out data
            if(i == 1) //latitude
            {
                  if( gps_parse_coordinate(sentence,
                            &newpos->lat_degrees,
                          &newpos->lat_minutes)
                    )
                  {
                        return -3; //get_coordinate failed
                  }
            }
            if(i == 2) //lat direction
            {
                  newpos->lat_direction = *sentence;
            }
            if(i == 3) // longitude
            {
```

```
                  if( gps_parse_coordinate(sentence,
                        &newpos->lon_degrees,
                        &newpos->lon_minutes)
                    )
                  {
                        return -3; //get_coordinate failed
                  }
            }
            if(i == 4) //lon direction
            {
                  newpos->lon_direction = *sentence;
            }
            if(i == 5) //link quality
            {
                  if(*sentence == '0')
                        return -2;
                  if(*sentence == '1')
                        return -1;
            }
      }
    }
    else
    {
      return -3; //unknown sentence type
    }
    return 0;
}

/*                START                FUNCTION                DESCRIPTION
********************************************
ERROR_function

SYNTAX:        int ERROR_function(new_hdg);

KEYWORDS:      nav, control

DESCRIPTION:    Determines heading error for use by Nav and Control
costatements

PARAMETER1:     new_hdg - latest update of bearing to next waypoint or
direction
                    to drive based upon sonar contact

RETURN VALUE:  hdg_error

SEE ALSO:

END                                                             DESCRIPTION
***********************************************************/

int ERROR_function(float new_hdg)
{
    hdg_error = new_hdg - curr_hdg;

    if (hdg_error <= 6 && hdg_error >= -6)
            {
            hdg_error = 0;
```

```
            }

     return(hdg_error);
}

/*              START                 FUNCTION                 DESCRIPTION
********************************************
gps_get_position

SYNTAX:                  void msDelay(long sd);

KEYWORDS:      delay, wait

DESCRIPTION:   introduces a defined ms delay loop

PARAMETER1:    sd - number of ms to wait

SEE ALSO:

END                                                            DESCRIPTION
**********************************************************/

void msDelay (long sd)
{
    unsigned long t1;

    t1 = MS_TIMER;
    for (t1 = MS_TIMER; MS_TIMER < (sd + t1); );
}

//
***********************************************************************
*****
//
//                            Communication Start
//
//
***********************************************************************
*****

void CommStart()
{
    sock_init();
    if (!(host = resolve(INTERFACE_ADDRESS)))
    {
      exit(3);
    }

/*---------------------------------------------------------------------
   open outgoing error port
---------------------------------------------------------------------*/

    if   (!udp_open(&error_data,   ERROR_PORT,   0xffffffff,   ERROR_PORT,
NULL))
    {
      exit(3);
    }
```

```
    sock_mode( &error_data, TCP_MODE_ASCII);
    sock_mode( &error_data, UDP_MODE_NOCHK);

/*------------------------------------------------------------------
   open incoming waypoint port
  -------------------------------------------------------------------*/

    if (!udp_open(&wp_data, WP_PORT, 0xffffffff, WP_PORT, NULL))
    {
      sock_puts(&error_data, "$Unable to open WP UDP session\n");
      exit(3);
    }
    sock_mode( &wp_data, UDP_MODE_NOCHK);

/*-------------------------------------------------------------------
   open incoming manual control port
  -----------------------------------------------------------------*/

    if (!udp_open(&man_data, MAN_PORT, 0xffffffff, MAN_PORT, NULL))
    {
      sock_puts(&error_data, "$Unable to open MANUAL UDP session\n");
      exit(3);
    }
    sock_mode( &man_data, UDP_MODE_NOCHK);

/*-----------------------------------------------------------------
   open outgoing compass port
  -------------------------------------------------------------------*/

    if      (!udp_open(&compass_data,     COMPASS_PORT,      0xffffffff,
COMPASS_PORT, NULL)) {
      sock_puts(&error_data, "$Unable to open COMPASS UDP session\n");
      exit(3);
    }
    sock_mode( &compass_data, TCP_MODE_ASCII);
    sock_mode( &compass_data, UDP_MODE_NOCHK);


/*------------------------------------------------------------------
   open outgoing GPS port
  -------------------------------------------------------------------*/

    if (!udp_open(&gps_data, GPS_PORT, 0xffffffff, GPS_PORT, NULL))
    {
      sock_puts(&error_data, "$Unable to open GPS UDP session\n");
      exit(3);
    }
    sock_mode( &gps_data, TCP_MODE_ASCII);
    sock_mode( &gps_data, UDP_MODE_NOCHK);

   sock_puts(&error_data, "$Sockets are established\n");

    if (sock_recv_init( &wp_data, wptBuf, (word)sizeof(wptBuf)))
    {
      sock_puts(&error_data, "$Could not enable WP buffer.\n");
      exit(3);
    }
```

```
    if (sock_recv_init( &man_data, cmdBuf, (word)sizeof(cmdBuf)))
     {
       sock_puts(&error_data, "$Could not enable MAN buffer.\n");
       exit(3);
     }
}     // end Comm Start
```

APPENDIX C. EXPERIMENTAL DATA

Experiment 1

Voltage	Tail Angle	Voltage	Tail Angle
2.344976	-44.61956	2.374723	-14.873028
2.344976	-44.61956	2.374723	-14.873028
2.344976	-44.61956	2.374723	-14.873028
2.344976	-44.61956	2.374723	-14.873028
2.344976	-44.61956	2.374723	-14.873028
2.344976	-44.61956	2.37968	-9.915352
2.344976	-44.61956	2.37968	-9.915352
2.349934	-39.661884	2.37968	-9.915352
2.349934	-39.661884	2.37968	-9.915352
2.349934	-39.661884	2.37968	-9.915352
2.349934	-39.661884	2.374723	-14.873028
2.354892	-34.704208	2.37968	-9.915352
2.354892	-34.704208	2.37968	-9.915352
2.354892	-34.704208	2.374723	-14.873028
2.354892	-34.704208	2.37968	-9.915352
2.349934	-39.661884	2.37968	-9.915352
2.354892	-34.704208	2.389596	0
2.354892	-34.704208	2.384638	-4.957676
2.354892	-34.704208	2.384638	-4.957676
2.354892	-34.704208	2.384638	-4.957676
2.354892	-34.704208	2.384638	-4.957676
2.349934	-39.661884	2.384638	-4.957676
2.354892	-34.704208	2.389596	0
2.354892	-34.704208	2.384638	-4.957676
2.354892	-34.704208	2.384638	-4.957676
2.354892	-34.704208	2.389596	0
2.359849	-29.746532	2.384638	-4.957676
2.354892	-34.704208	2.384638	-4.957676
2.354892	-34.704208	2.384638	-4.957676
2.354892	-34.704208	2.384638	-4.957676
2.354892	-34.704208	2.389596	0
2.354892	-34.704208	2.384638	-4.957676
2.359849	-29.746532	2.389596	0
2.359849	-29.746532	2.389596	0
2.359849	-29.746532	2.389596	0
2.359849	-29.746532	2.389596	0
2.359849	-29.746532	2.389596	0
2.359849	-29.746532	2.389596	0
2.359849	-29.746532	2.384638	-4.957676
2.359849	-29.746532	2.389596	0
2.359849	-29.746532	2.394553	4.957676
2.359849	-29.746532	2.394553	4.957676
2.359849	-24.788857	2.394553	4.957676
2.359849	-24.788857	2.394553	4.957676
2.359849	-24.788857	2.394553	4.957676
2.359849	-24.788857	2.399512	9.915829
2.359849	-24.788857	2.399512	9.915829
2.364807	-19.830704	2.399512	9.915829
2.359849	-24.788857	2.399512	9.915829
2.359849	-24.788857	2.399512	9.915829
2.364807	-19.830704	2.404469	14.873505
2.359849	-24.788857	2.404469	14.873505
2.359849	-24.788857	2.404469	14.873505
2.364807	-19.830704	2.399512	9.915829
2.364807	-19.830704	2.399512	9.915829
2.359849	-24.788857	2.404469	14.873505
2.364807	-19.830704	2.404469	14.873505
2.359849	-24.788857	2.404469	14.873505
2.364807	-19.830704	2.404469	14.873505

2.359849	-24.788857	2.399512	9.915829
2.364807	-19.830704	2.409427	19.831181
2.364807	-19.830704	2.409427	19.831181
2.359849	-24.788857	2.409427	19.831181
2.359849	-24.788857	2.409427	19.831181
2.359849	-24.788857	2.409427	19.831181
2.359849	-24.788857	2.414385	24.788857
2.359849	-24.788857	2.414385	24.788857
2.369765	-19.830704	2.414385	24.788857
2.369765	-19.830704	2.419342	29.746532
2.364807	-24.78838	2.414385	24.788857
2.369765	-19.830704	2.419342	29.746532
2.369765	-19.830704	2.414385	24.788857
2.369765	-19.830704	2.414385	24.788857
2.369765	-19.830704	2.419342	29.746532
2.369765	-19.830704	2.414385	24.788857
2.369765	-19.830704	2.419342	29.746532
2.369765	-19.830704	2.419342	29.746532
2.374723	-14.873028	2.419342	29.746532
2.374723	-14.873028	2.419342	29.746532
2.374723	-14.873028	2.419342	29.746532
2.374723	-14.873028	2.419342	29.746532
2.374723	-14.873028	2.414385	24.788857
2.374723	-14.873028	2.419342	29.746532
2.364807	-24.78838	2.419342	29.746532
2.374723	-14.873028		
2.374723	-14.873028		
2.374723	-14.873028		

Experiment 2

Time	Pitch	Tail	Time	Pitch	Tail
1	0.7	0	6468	9.3	9.9154
610	0.7	-4.9579	6569	8.4	12.3942
659	0.3	-4.9579	6619	8.2	9.9154
1212	1.1	-4.9579	6670	7	9.9154
1270	0.7	-2.4788	6721	7.5	9.9154
1319	0.3	-2.4788	6773	5.8	7.4365
1369	0.7	-2.4788	6927	4.7	7.4365
1578	0.7	-2.4788	6979	3.6	4.9577
1677	1.2	-4.9579	7029	2.1	4.9577
1794	2.3	-2.4788	7146	0.5	2.4788
2028	3	-2.4788	7196	1.1	4.9577
2086	3.6	-4.9579	7353	0.6	2.4788
2188	4.7	4.9577	7455	-0.3	4.9577
2239	4	4.9577	7505	0.8	4.9577
2290	3.7	4.9577	7603	-1.8	4.9577
2393	4.6	4.9577	7660	-0.5	4.9577
2495	5.1	4.9577	7709	-2	4.9577
2653	5.8	4.9577	7768	-4.5	4.9577
2909	6.4	7.4365	7820	-5.2	4.9577
3011	6.9	4.9577	7923	-5.5	2.4788
3062	7.3	7.4365	7973	-6.3	2.4788
3112	7.7	7.4365	8022	-7.3	2.4788
3368	8	7.4365	8071	-8.4	0
3472	8.6	4.9577	8173	-9.2	0
3624	9.1	7.4365	8223	-8.7	-2.4788
3675	9.3	7.4365	8323	-10.8	-2.4788
3725	9.6	9.9154	8372	-11.9	-2.4788
3776	9.8	9.9154	8421	-11	-2.4788
3827	10.1	9.9154	8471	-10.8	-4.9579
3878	10.8	9.9154	8520	-11.6	-4.9579
3929	10.6	9.9154	8621	-13.8	-7.4368
3981	10.8	9.9154	8670	-14.1	-4.9579
4032	11	9.9154	8721	-14.5	-7.4368
4137	11.4	9.9154	8770	-13.9	-7.4368
4188	11.7	9.9154	8820	-14.7	-9.9156

4290	11.9	9.9154	8920	-15.5	-9.9156
4392	12.4	12.3942	8969	-15	-9.9156
4442	12.8	12.3942	9020	-14.6	-12.3944
4492	13.1	12.3942	9069	-15.2	-12.3944
4541	13.7	12.3942	9119	-14.9	-12.3944
4592	14.1	12.3942	9219	-13.9	-14.8733
4642	13.8	12.3942	9268	-13.7	-14.8733
4744	13.7	14.8733	9319	-14.1	-14.8733
4847	13.9	14.8733	9420	-13.7	-14.8733
4898	14.7	14.8733	9469	-14.3	-14.8733
4998	15.5	14.8733	9519	-14	-14.8733
5048	16	-19.8309	9621	-13	-14.8733
5099	14.9	-37.183	9671	-12.4	-14.8733
5151	17.2	-37.183	9823	-10.4	-12.3944
5202	16.8	-37.183	9872	-11.6	-12.3944
5253	15.2	-37.183	9922	-11.1	-9.9156
5303	14.5	-37.183	10022	-10.6	-9.9156
5355	13.7	-34.7042	10226	-9	-7.4368
5406	13.4	-29.7465	10276	-8.3	-9.9156
5457	11.7	7.4365	10327	-8.1	-9.9156
5508	11.9	9.9154	10376	-8.3	-9.9156
5558	12.6	9.9154	10427	-8.6	-7.4368
5659	12	12.3942	10476	-8.9	-7.4368
5710	12.2	12.3942	10527	-6.2	-7.4368
5759	11.1	12.3942	10576	-5.8	-7.4368
5809	11.3	12.3942	10626	-5.6	-4.9579
5911	11.1	12.3942	10728	-4	-4.9579
6011	11.3	12.3942	10779	-3.8	-4.9579
6214	10.3	14.8733	10882	-3.6	-2.4788
6316	10.1	12.3942	10935	-1.3	-4.9579
6417	10.2	12.3942	11053	0.4	-4.9579

Experiment 3a - 10 deg

	Trial 1			Trial 2	
Time	Pitch	Tail	Time	Pitch	Tail
1	-0.9	0	1	4	0
287	0.5	-2.4788	56	-0.5	0
335	-2	-14.8733	152	-0.2	0
391	-0.9	-14.8733	201	0.5	0
439	-0.4	-12.3944	249	-0.3	-2.4788
534	-0.7	-4.9579	299	-0.5	2.4788
580	-0.4	-7.4368	397	-1.2	7.4365
676	0	-2.4788	452	-0.4	7.4365
725	-0.5	-4.9579	747	-0.9	4.9577
867	3.7	-2.4788	795	-0.6	2.4788
919	2	4.9577	938	20	-4.9577
977	-28	4.9577	987	-18	-19.8309
1025	-12.9	-17.3521	1035	12.7	-14.8733
1072	15.5	-14.8733	1085	9.9	-19.8309
1120	5.4	-44.6198	1135	21.3	-29.7465
1171	-7.2	-29.7465	1184	-14.3	-17.3521
1218	3	-27.2677	1231	4.6	-14.8733
1287	24.8	9.9154	1281	27	-19.8309
1334	3	-22.3098	1331	4	-27.2677
1404	25	-24.7889	1390	-5.7	-14.8733
1453	7	-37.183	1439	4	-2.4788
1504	3	-42.141	1498	27.7	4.9577
1565	-12.9	-49.5775	1546	-1.6	-4.9577
1614	-2.8	-47.0986	1603	-0.8	-4.9577
1664	-1.6	-39.6619	1649	-1.7	-4.9577
1715	-2.5	-32.2254	1705	-0.2	-4.9577
1772	-1.1	-27.2677	1752	0.2	-12.3944
1827	-0.9	-14.8733	1800	-1	-12.3944
1875	-0.3	2.4788	1896	-0.2	-9.9156

1922	-1.3	4.9577	1944	0.2	2.4788
2035	-1	12.3942	1993	0	2.4788

Trial3			Trial 4		
Time	Pitch	Tail	Time	Pitch	Tail
1	0.4	0	1	-0.6	0
142	-0.7	9.9154	239	0	0
189	0.3	7.4365	289	-0.7	0
335	0.2	0	435	-4.1	0
722	0.3	-2.4788	485	-2.9	2.4788
818	0.7	-2.4788	542	-1.1	2.4788
865	4.8	4.9577	888	4.1	-12.3942
914	18	7.4365	937	0	-7.4365
962	21.1	-12.3944	986	2.7	-7.4365
1009	30.8	-19.8309	1042	3.7	14.8733
1058	1.2	-17.3521	1090	-3.5	14.8733
1113	23.7	-34.7042	1139	-7.5	-19.8309
1162	4	-17.3521	1187	4	9.9156
1221	2.5	-14.8733	1370	35.3	4.9577
1276	14.2	-12.3944	1419	-1.1	-24.7886
1324	5.3	-19.8309	1474	-11.6	-17.3521
1372	-12.6	-17.3521	1521	-15.7	-34.7042
1420	-16	-19.8309	1569	4	-29.7463
1470	-1.6	-19.8309	1629	-7.1	-29.7463
1526	-11.6	-12.3944	1678	-0.6	-34.7042
1576	-1.3	-9.9156	1725	0.2	-32.2254
1689	-0.8	-7.4368	1821	-3.9	7.4368
1833	-0.3	-4.9577	1871	-0.2	9.9156
1880	-0.6	0	1919	0.6	12.3944
2075	-0.4	4.9577	2164	0.3	14.8733

Trial 5		
Time	Pitch	Tail
1	0.5	-2.4788
86	0.7	-4.9577
134	1.1	-4.9577
190	0.7	-4.9577
336	-0.8	-17.3521
384	0	-4.9577
434	-0.3	-2.4788
481	1	-4.9577
530	0.7	-4.9577
577	0.5	-2.4788
865	7.8	-4.9577
915	24	7.4365
963	19.8	-19.8309
1011	18.4	-19.8309
1058	4	-17.3521
1108	15.1	-24.7886
1155	23.6	-19.8309
1202	6.3	-14.8733
1253	4	-19.8309
1376	-19.3	12.3944
1424	4	-7.4365
1484	-5.3	7.4365
1534	-2.7	4.9577
1590	0.2	2.4788
1637	-0.4	2.4788
1782	0	2.4788

Experiment 3b - 15 deg

Trial 1

Time	Pitch	Tail
1	3	2.4788
58	-1.4	2.4788
345	-1	2.4788
393	-2.4	2.4788
450	-2.9	-2.4788
506	2.7	-4.9579
562	-0.7	-2.4788
610	-1.3	-2.4788
667	-1.5	-2.4788
723	-1.3	-2.4788
1066	9.1	-7.4368
1114	3	-12.3944
1186	15.8	12.3942
1234	3	17.3521
1305	16.1	17.3521
1353	19.1	14.8733
1402	28.8	-17.3521
1450	-0.6	-34.7042
1498	3	-34.7042
1568	9.6	-9.9156
1616	23.5	0
1663	-11.3	-12.3944
1711	-21.9	-12.3944
1758	-9.5	-9.9156
1807	-7.6	-12.3944
1904	-8.1	-9.9156
1952	-7.8	-12.3944
2001	-7.6	-12.3944

Trial 2

Time	Pitch	Tail
1	-1.3	0
169	-2.1	-27.2677
225	-0.1	-2.4788
272	-1.8	-2.4788
327	-1.3	-2.4788
1013	-3.6	-7.4368
1062	7.4	-4.9579
1111	5.8	9.9154
1161	-0.7	12.3942
1209	24.2	2.4788
1255	9.9	-24.7889
1304	9.7	-17.3521
1354	14.5	14.8733
1401	13.9	17.3521
1448	8.7	19.8309
1496	-8.7	0
1546	24.2	-2.4788
1593	11.4	-14.8733
1641	-6.1	-14.8733
1690	-9.7	-9.9156
1739	-8.9	-7.4368
1787	-10	-14.8733
1837	-9.4	-7.4368
1884	-9	-7.4368
1934	-8.8	-7.4368
1983	-9.6	0
2082	-9.7	-4.9579

Trial3

Time	Pitch	Tail
1	3	0
66	-0.9	0
403	-0.8	0
887	5.4	-4.9579
935	20.4	9.9154
982	13.1	-32.2254
1031	10.4	-7.4368
1080	13.5	14.8733
1127	23.9	17.3521
1175	11.3	-29.7465
1224	14.8	-7.4368
1271	3	12.3942
1342	-4.5	19.8309
1392	17.9	9.9154
1440	22.7	-22.3098
1487	23.4	-29.7465
1534	3.1	-22.3098
1583	-2.1	-22.3098
1638	-0.3	-14.8733
1687	2.5	-2.4788
1745	0.2	12.3942
1795	-0.7	12.3942
1843	0.4	7.4365
1893	-0.2	7.4365
1941	0.1	7.4365
2040	-0.8	7.4365

Trial 4

Time	Pitch	Tail
1	3	0
66	-0.8	0
113	-1	0
162	-0.4	0
209	-1.3	-2.4788
267	-0.5	-2.4788
315	-0.8	0
412	-2.6	-4.9577
468	-0.9	-4.9577
899	6.7	-7.4365
998	9.6	14.8733
1048	-7	12.3944
1098	6.4	4.9579
1148	13.1	4.9579
1195	14.1	12.3944
1242	20.7	12.3944
1290	3	-19.8309
1361	-25.9	-17.3519
1408	-14.4	-7.4365
1455	9.4	-14.873
1503	3	-14.873
1573	-1	32.2254
1625	-2.5	37.1833
1678	-0.1	34.7042
1729	0.2	34.7042
1887	0.6	37.1833
1938	0.4	37.1833

Trial 5

Time	Pitch	Tail
1	3	2.4788

59	-1.2	2.4788
287	-1	2.4788
336	-1.3	-4.9579
393	-2.1	-2.4791
449	-1.3	-7.4368
506	-1.6	-7.4368
561	-1.2	-7.4368
1073	-4.3	-9.9156
1122	15.5	-2.4791
1170	3	14.873
1242	15	29.7463
1290	21.3	24.7886
1339	21.8	-19.8309
1388	20.4	-22.31
1437	13	-22.31
1486	13.8	-47.0986
1534	3	-17.3521
1606	-25.9	-14.8733
1653	25.7	-47.0986
1701	-5.9	-47.0986
1751	7.9	-39.6621
1801	0.2	-39.6621
1849	-0.1	2.4788
1897	-2.4	7.4365
1952	-1.6	17.3521
2008	-0.1	14.873

LIST OF REFERENCES

1. Quinn, R.D., et al. "Improved Mobility Through Abstracted Biological Principles," *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002, pp. 2652-2657.

2. Dunbar, T.W. *"Demonstration of Waypoint Navigation for a Semi-Autonomous Prototype Surf-Zone Robot."* M.S. thesis, Naval Postgraduate School, 2006.

3. Ward, J.L. *"Design of a Prototype Autonomous Amphibious Whegs™ Robot for Surf-Zone Operations."* M.S. thesis. Naval Postgraduate School, 2005.

4. Boxerbaum, A.S., et al. "Design of a Semi-Autonomous Amphibious Surf-Zone Robot," Technical Report. Case Western Reserve University, 2009.

5. Herkamp, J.F., *"Deployment of Shaped Charges by a Semi-Autonomous Ground Vehicle,"* M.S. thesis, Naval Postgraduate School, Monterey, California, 2007.

6. Maxon. Planetary Gearhead 42C. http://www.treffer.com.br/produtos/maxon/redutores/pdf/245.pdf (accessed June 2009).

7. Superdroid Robots. MD22 Devantech Dual Motor Driver. http://www.superdroidrobots.com/product_info/MD22_info.htm (accessed June 2009).

8. Superdroid Robots. PWM Motor Controller 3A 12-55V. http://www.superdroidrobots.com/shop/item.asp?itemid=583 (accessed June 2009).

9. National Semiconductor Corporation. *LM555 Timer Specification Sheet*. http://www.national.com/mpf/LM/LM555.html (accessed June 2009).

10. Fairchild Semiconductor. *LM78xx Fixed Voltage Regulator.* http://www.datasheetcatalog.com/datasheets_pdf/L/M/7/8/LM7812.shtml (accessed June 2009).

11. Spectrol. 536 Series Potentiometer. http://www.nteinc.com/pot_web/pdf/536_series.pdf (accessed June 2009).

12. Z-World. BL-2000.
http://www.zworld.com/products/bl2000/
(accessed June 2009).

13. Netgear. Rangemax 240 Wireless Router.
http://www.netgear.com/Solutions/HomeNetworking/WirelessNetworking/RangeMax240.aspx
(accessed June 2009).

14. Garmin. GPS 16 HVS.
http://www.garmin.com/manuals/425_TechnicalSpecifications.pdf
(accessed June 2009).

15. Honeywell. HMR3000 Digital Compass.
http://www.ssec.honeywell.com/magnetic/datasheets/hmr3000.pdf
(accessed June 2009).

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Physics Department
   Naval Postgraduate School
   Monterey, California

4. Richard Harkins
   Department of Applied Physics
   Naval Postgraduate School
   Monterey, California

5. Peter Crooker
   Department of Applied Physics
   Naval Postgraduate School
   Monterey, California

6. Alexander Boxerbaum
   Case Western Reserve University
   Cleveland, Ohio

7. Dr. Ravi Vaidyanathan
   Bristol University
   Bristol, United Kingdom