NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

DEPLOYMENT OF SHAPED CHARGES BY A SEMI-AUTONOMOUS GROUND VEHICLE

by

John Frederick Herkamp

June 2007

Thesis Advisor:	Richard Harkins
Second Reader:	Peter Crooker

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2007	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE Deployment of Shaped Charges by a Semi-Autonomous Ground Vehicle			**5. FUNDING NUMBERS**
6. AUTHOR(S) John Frederick Herkamp			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

Neutralization of remotely operated Improvised Explosive Devices (IEDs) is a dangerous task risking human lives on a daily basis. BigFoot seeks to replace the local human component by deploying and remotely detonating shaped charges to destroy IEDs. This research developed a platform that can autonomously navigate GPS waypoints, avoid obstacles, and provide remote user controls for an onboard robotic arm to deploy and remotely detonate shaped charges. BigFoot incorporates improved communication range over previous Autonomous Ground Vehicles and an updated user interface that includes controls for the arm and camera by interfacing multiple microprocessors. BigFoot is capable of avoiding static and mobile obstacles as well handling most surfaces with minor slopes. BigFoot continues to be somewhat limited by communications range and GPS availability. However, BigFoot is an ideal platform for relatively short range deployment to neutralize roadside IEDs.

14. SUBJECT TERMS Autonomous, Robot, Shaped Charge, Improvised Explosive Device			**15. NUMBER OF PAGES** 201
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500

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

THIS PAGE INTENTIONALLY LEFT BLANK

DEPLOYMENT OF SHAPED CHARGES BY A SEMI-AUTONOMOUS GROUND VEHICLE

John Frederick Herkamp
Lieutenant, United States Navy
B.S. Electrical Engineering, Old Dominion University, 2000

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN PHYSICS

from the

NAVAL POSTGRADUATE SCHOOL
June 2007

Author:	John Frederick Herkamp
Approved by:	Richard Harkins Thesis Advisor
	Peter Crooker Second Reader
	James Luscombe Chairman, Department of Physics

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Neutralization of remotely operated Improvised Explosive Devices (IEDs) is a dangerous task risking human lives on a daily basis. BigFoot seeks to replace the local human component by deploying and remotely detonating shaped charges to destroy IEDs. This research developed a platform that can autonomously navigate GPS waypoints, avoid obstacles, and provide remote user controls for an onboard robotic arm to deploy and remotely detonate shaped charges. BigFoot incorporates improved communication range over previous Autonomous Ground Vehicles and an updated user interface that includes controls for the arm and camera by interfacing multiple microprocessors. BigFoot is capable of avoiding static and mobile obstacles as well handling most surfaces with minor slopes. BigFoot continues to be somewhat limited by communications range and GPS availability. However, BigFoot is an ideal platform for relatively short range deployment to neutralize roadside IEDs.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF EQUATIONS

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I first must thank my Lord and Savior Jesus Christ. Without God's blessings, none of this or any other aspects of my life would have been possible. I must also thank my patient, loving wife, Raylene, and son Isaac for sacrificing their time and their endless love and support. I thank my family for their continued love and support. I thank my advisor, Professor Richard Harkins, for his guidance and advice. I thank Sam Barone and George Jaksha for their knowledge and assistance. I also thank Todd Williamson and Ben Miller for their invaluable help in BigFoot's development and testing.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Robotics is playing an increasingly important role in military operations. Unmanned Aerial Vehicles (UAVs) and Unmanned Ground Vehicles (UGVs) are used for everything from surveillance to carrying out combat operations and neutralization of Improvised Explosive Devices (IEDs). While generally effective at their specific missions, these platforms are not autonomous and operators must control the platform to handle any unexpected scenarios.

The Department of Defense Joint Ground Robotics Enterprise (JGRE) was founded as the Joint Robotics Program in 1989 by the Secretary of Defense and was reorganized into the JGRE structure in 2006. Its mission is to develop and assess unmanned robotic platforms for military applications [1]. Many platforms have been designed and tested, but autonomous operation has yet to be successfully implemented in any of the current applications.

A. UGV

TALON is the most commonly used ground robot currently in the military's inventory. It is used most commonly for IED detection and neutralization, but has been equipped with weapons and sensors for other missions. TALON weighs close to 140 lbs with an arm installed with approximate dimensions of 1' X 2' X 3' and is not autonomous [3]. Since January 2004, TALON has been used to neutralize more than 1000 IEDs that would otherwise have been handled by US EOD personnel [2]. While it is a flexible and effective platform, it is a cumbersome piece of equipment to transport and very expensive at approximately $60,000 per unit. These characteristics make TALON undesirable for large-scale employment. Smaller, lighter, and cheaper robots are desirable in this situation.

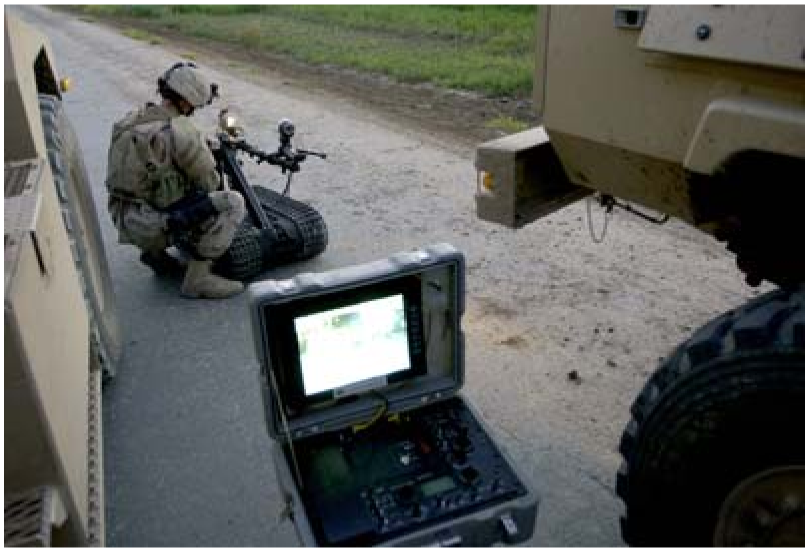


Figure 1. TALON Robot with arm (From [4]).

B. UAV

The most well known UAV program is Predator. Predator is capable of a range of missions from intelligence, reconnaissance and surveillance to launching Hellfire missiles in actual combat operations. Predator utilizes satellite communications thereby permitting remote control from any location in the world [5]. Figure 2 shows a Predator UAV in flight. Figure 3 shows the UAV Tactical Control System for Predator which consists of an entire trailer of computers and electronics.

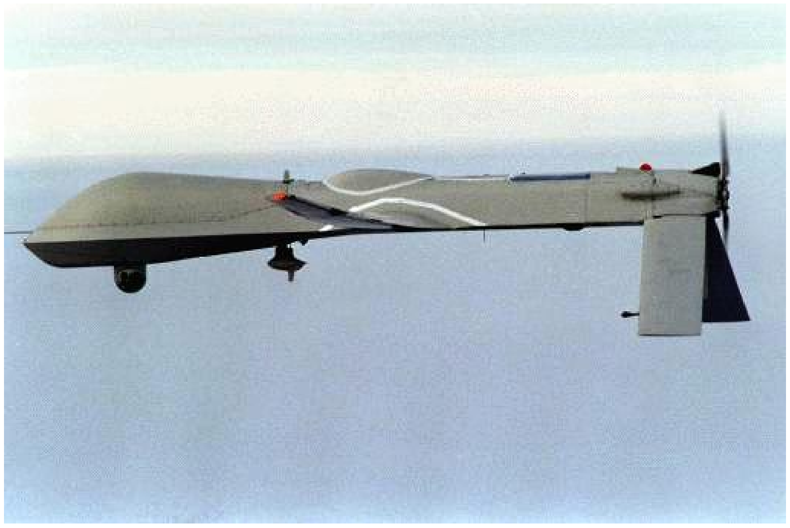


Figure 2. Predator MAE UAV in flight (From [5]).

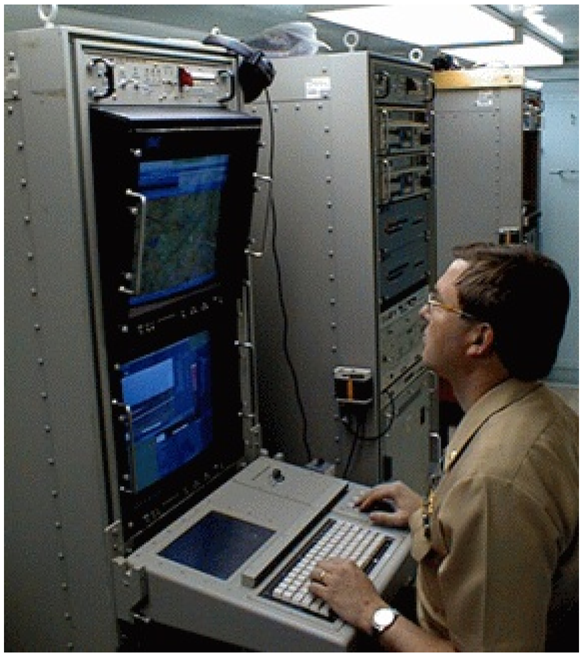


Figure 3. UAV Tactical Control System (From [6]).

C. PREVIOUS NAVAL POSTGRADUATE SCHOOL (NPS) PROJECTS

The NPS Small Robot Technology (SMART) initiative develops prototype robotic platforms for the military. The robots vary in size from inches to yards and their missions are just as extensive. BigFoot's development started with a prototype known as Bender. Bender was not designed with any particular mission objective, but was developed as a platform to investigate autonomous architecture. It had a box shape and utilized a hardened track chassis. Bender utilized ultrasonic ranging sensors to accomplish collision avoidance. An onboard computer (a commercial BL2000) controlled all robot functions using the basic programming language Dynamic C. Bender was equipped with a web-cam to view any contacts in its path. Bender could function autonomously to the extent that it could move from one point to another and avoid large obstacles in the process. Figure 4 shows Bender in its final form. Bender's large size and expensive chassis make the platform unsuitable for BigFoot's mission.

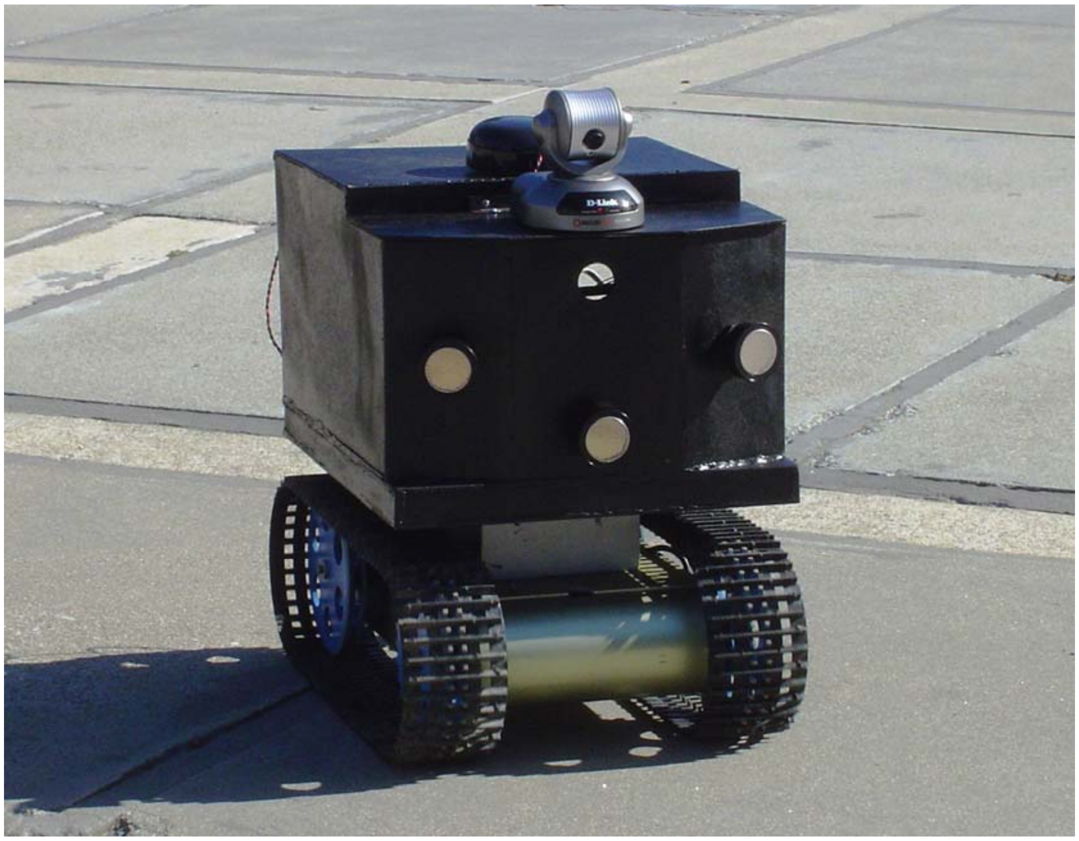


Figure 4. Bender.

The second generation of autonomous robots was directed toward naval specific applications. LT Jason Ward created Lopez as a prototype for a surf-zone robot to conduct reconnaissance and surveillance on a beachhead. These platforms will be launched from surface ships or submarines in the future. Figure 5 shows the working Lopez model.

The third generation of the robots from the SMART program was created by ENS Tom Dunbar from NPS in collaboration with Case Western University. Agbot is a more powerful version of Lopez constructed from aluminum. This platform suffers from structural problems, but it is a working prototype that has been successfully tested on sand, grass, and concrete [7]. Figure 6 shows Agbot in its final state. Both Lopez and Agbot run autonomously from a Java interface. These platforms both incorporate the same basic components (GPS, onboard computer, compass, camera, and router) as BigFoot.

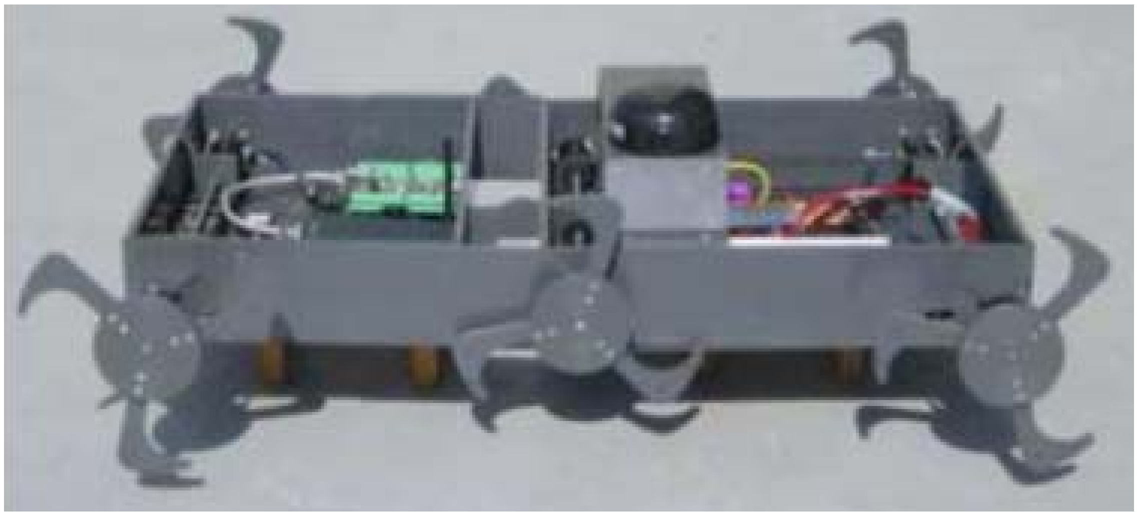


Figure 5. Lopez with all components installed (From [7]).

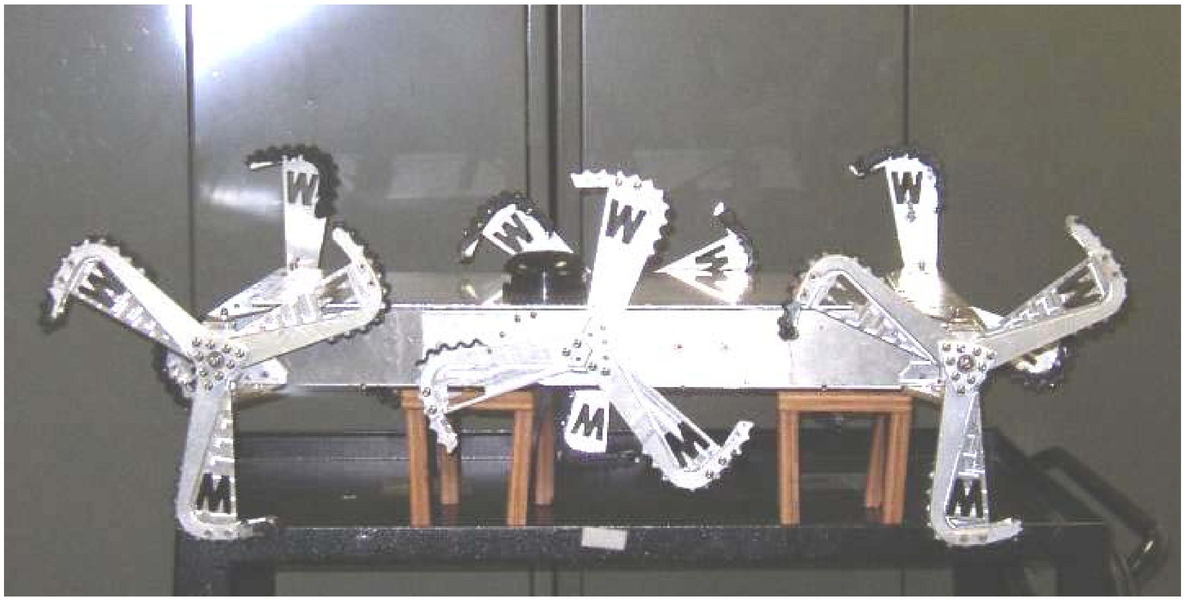


Figure 6. Agbot with all components installed (From [7]).

The fourth generation of SMART robots was created by MAJ Ben Miller. Autonomous Ground Vehicle (AGV) employs motion sensors and a web camera to detect, investigate, and report suspicious activity to a remote monitoring station. It also adapts previous work to a wheeled platform. AGV will be deployed in the future along roadways in Iraq to detect IED emplacement to protect troop and civilian movements. Figure 7 shows AGV on a test mission. The AGV platform is too small and slow to perform BigFoot's mission, but is a basis for BigFoot's development.

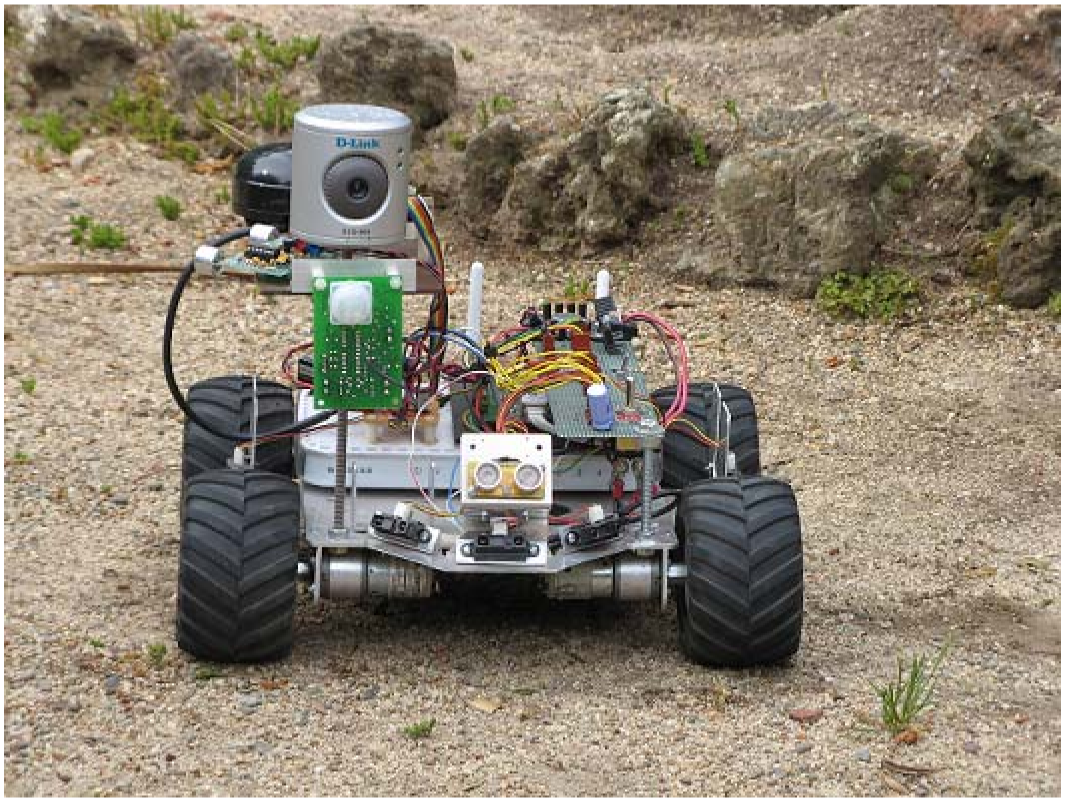


Figure 7. AGV on a test mission (From [8]).

D. PROJECT MOTIVATION

The IED problem in Iraq continues to kill soldiers and civilians alike. TALON robots are being used in the place of human with hands on efforts to remove or detonate IEDs that have been identified. As shown in Figure 8, these are not always completely successful. The TALON robots cost approximately $60,000 each. Figure 8 represents almost a quarter of a million dollars for 4 IEDs. Secondly, TALON is relatively large. This makes it inconvenient or even impossible for all units to take a TALON unit with them. As a result, smaller units must wait for a TALON to be delivered to them, extending the time the IED is in place and risking lives while personnel are stationary. By employing smaller, lighter, cheaper robots for IED neutralization, they can be made available to all units at a minimal cost. If the smaller cheaper robot is destroyed, it can be replaced at a significantly lower cost than TALON.

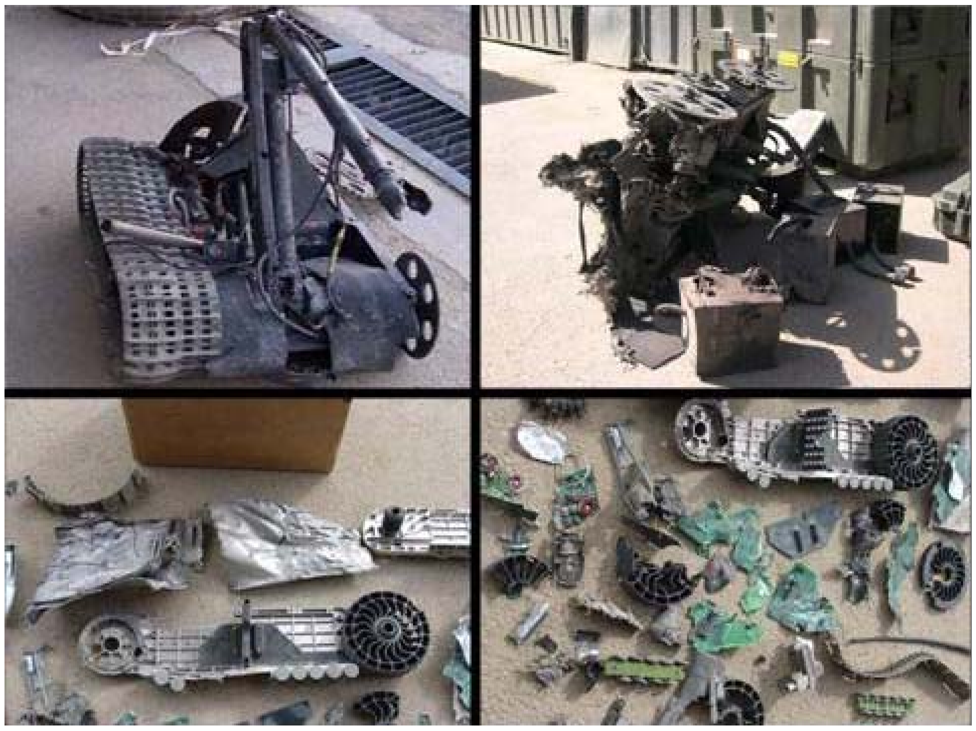


Figure 8. Four destroyed TALONs (From [2]).

THIS PAGE INTENTIONALLY LEFT BLANK

II. ROBOTIC FUNCTIONAL DESIGN

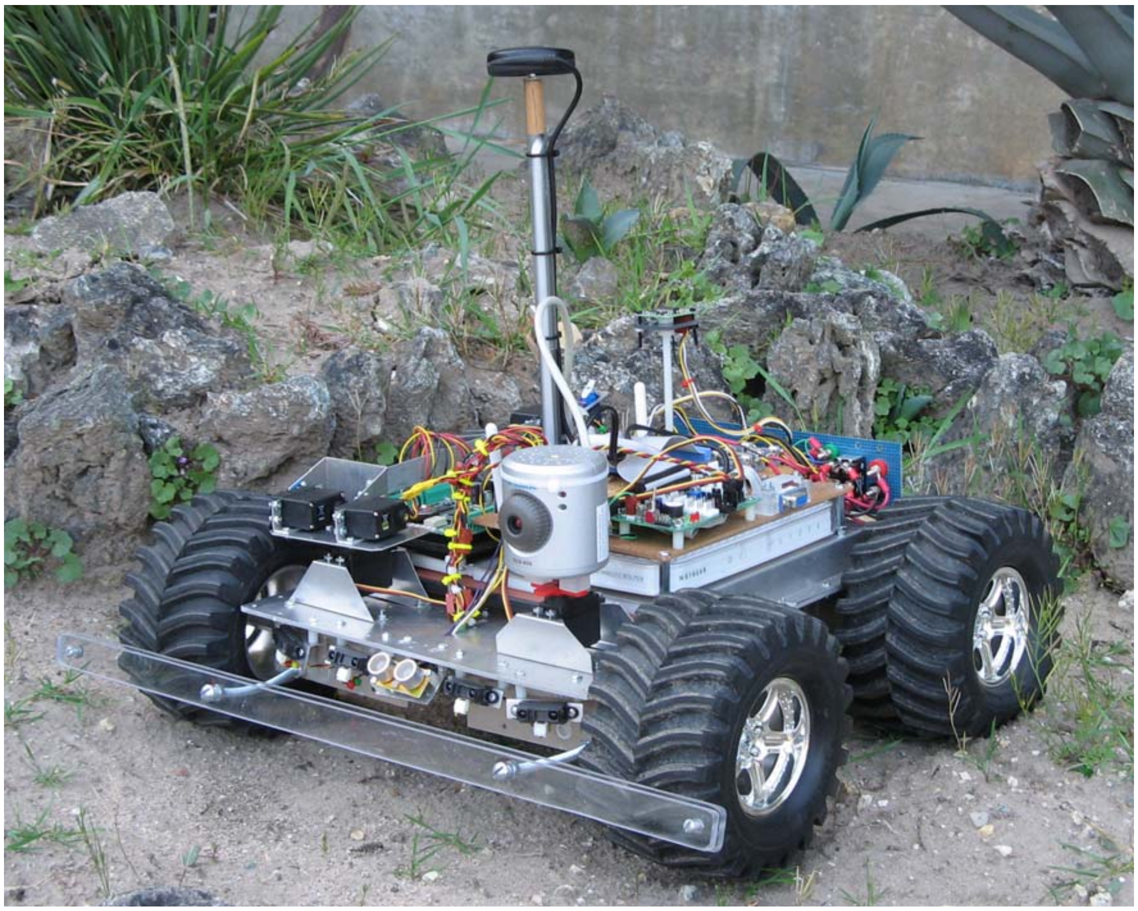


Figure 9. BigFoot.

A. PROBLEM SOLUTION

Figure 9, above, is the autonomous ground vehicle, known as BigFoot, designed to supplement or replace TALON. BigFoot is significantly smaller and cheaper than TALON. It is also capable of semi autonomous operation. BigFoot can navigate via GPS waypoints to a location of a suspected IED and avoid major obstacles on the way. Once at the threat location, BigFoot will be able to mix the previously inert chemical to arm a small shaped charge (once BigFoot is complete). Its arm can then be controlled remotely to place the shaped charge on the suspected IED. BigFoot can then navigate back to its origin and detonate the shaped charge once it is a safe distance away. BigFoot is equipped with a visual camera to send real time digital photos and streaming video as well as a thermal sensor for locating the IED when close. The Graphical User Interface (GUI) is interactive with real time pictures and graphical representations of robot condition.

1. Block Diagram

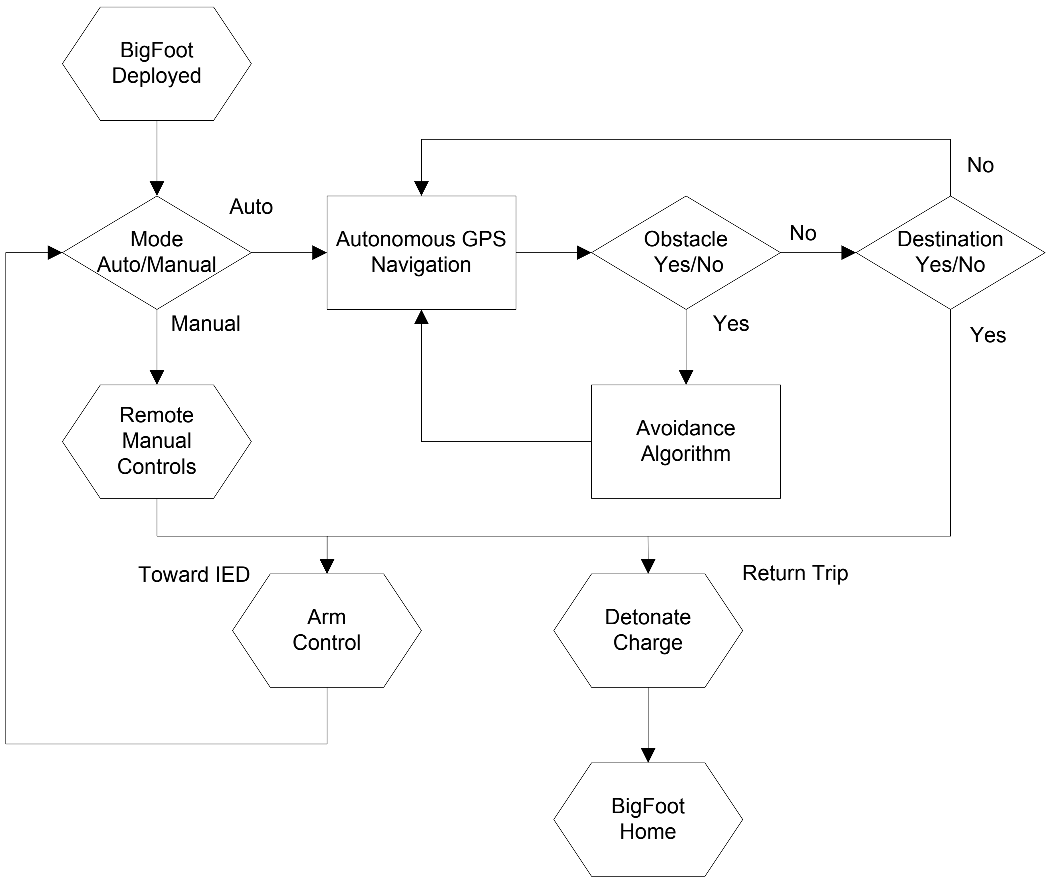


Figure 10. BigFoot Operational Flowchart.

Figure 10 shows a basic block diagram for BigFoot from the time it is dropped until it has returned to its origin.

2. Concept of Operations

Briefly, when placed, BigFoot will wait until a route is sent or it is driven manually. When driven manually, all controls will be as directed by a user at a remote laptop computer. When a route is sent, BigFoot will determine the appropriate course to steer to reach the next waypoint. BigFoot will then begin traveling to its next waypoint. While traveling autonomously, it will use onboard ultrasonic and infrared (IR) rangers to detect obstacles. When an obstacle is detected, BigFoot will stop and maneuver to avoid the obstacle and then return to the necessary course to reach the next waypoint. If BigFoot deviates from the necessary course, it will use Proportional-Integral-Derivative (PID) controls to return to course. Once a waypoint is achieved, BigFoot will take action to navigate to the next waypoint or stop and await command at the final waypoint. The operator will then take remote manual control to order BigFoot to mix the shaped charge’s explosive chemicals. The operator will remotely control the arm to place and release the charge. The operator will then order BigFoot to return. BigFoot will again autonomously navigate to ordered points. The operator will then direct BigFoot to detonate the charge to neutralize the suspected IED.

THIS PAGE INTENTIONALLY LEFT BLANK

III. EXPERIMENTAL DESIGN

Table 1 is a summary of the major components used in BigFoot's construction

Hardware Component	Vendor	Price	Operating Parameters
BigFoot Base	Superdroid Robots	$458.80	20" x 18" x 7", 2.5" ground clearance
Motor Battery	Superdroid Robots	$88.60	NiMH, 2 x 10 array of C batteries, 4Ah
Electronics Battery	Electrovaya	$369.00	Lithium Ion, 15V, 11Ah
Motors	Superdroid Robots	$21.95 ea.	24-volt, 190-rpm gear motors
Motor Controllers	Superdroid Robots	$125.00 ea	5-50VDC, 20A, Analog / I2C control
Ultrasonic Range Finder	Superdroid Robots	$62.00	Objects from 0" to 254", utilizes I2C bus
IR Rangers	Superdroid Robots	$14.85 ea.	Objects from 5 to 80cm, analog output
Thermopile	Superdroid Robots	$103.00	2-22um, 41º X 6º Field of View
Servos	Superdroid Robots	$114.95 ea	180º Sweep, 333 oz-in Torque
Router	Newegg	$129.99	802.11g wireless router, max range apx. 300m
Antenna	D-Link	$44.99	7dBi omni-directional antenna
BL2000	Rabbit	$449.00	Single-board computer, 22.1 MHz, 11 analog inputs, 2 analog outputs
Compass	Superdroid Robots	$52.00	Digital magnetic compass, heading to +/- 1.4 degrees
GPS	Garmin	$199.00	Low voltage system, utilizes WAAS network
Camera	D-Link	$94.99	Web server 10/100Mbps, 640x480, 320x240 resolution
ooPIC	Superdroid Robots	$59.00	20MHz, four 8 bit AtoD channels
ooPIC Adapter	Superdroid Robots	$64.00	Onboard Voltage Regulator, Connections for servos

Table 1. Summary of BigFoot's major components.

A. MECHANICAL CONSTRUCTION

1. Platform

BigFoot’s frame is constructed from an All Terrain Robot kit that consists of a base, motor mounts, and wheels. The base is a 12 x 16 x 1/8 inch sheet of aircraft grade aluminum. The motor mounts are 2 x 2.25 x 5 inch welded aluminum cases. The mounts and base are pre-drilled to permit multiple mounting options. The wheels are constructed of rubber mounted to a plastic hub and are approximately 6 inches in diameter. These wheels have a 0.96 coefficient of static friction on most typical ground type surfaces (concrete, asphalt, grass, dirt, etc). A bumper constructed from two pieces of ¼ x 1 x 22 inch polystyrene is installed on the front of BigFoot to prevent damage to front mounted electronic components in the event of a collision. The bumper extends past the wheels to prevent BigFoot from inadvertently flipping over if the wheels make contact with a vertical surface. When the wheels are installed, BigFoot is approximately 22 x 19 inches with a ground clearance of 2¼ inches. Figure 11 shows a bottom view of BigFoot.

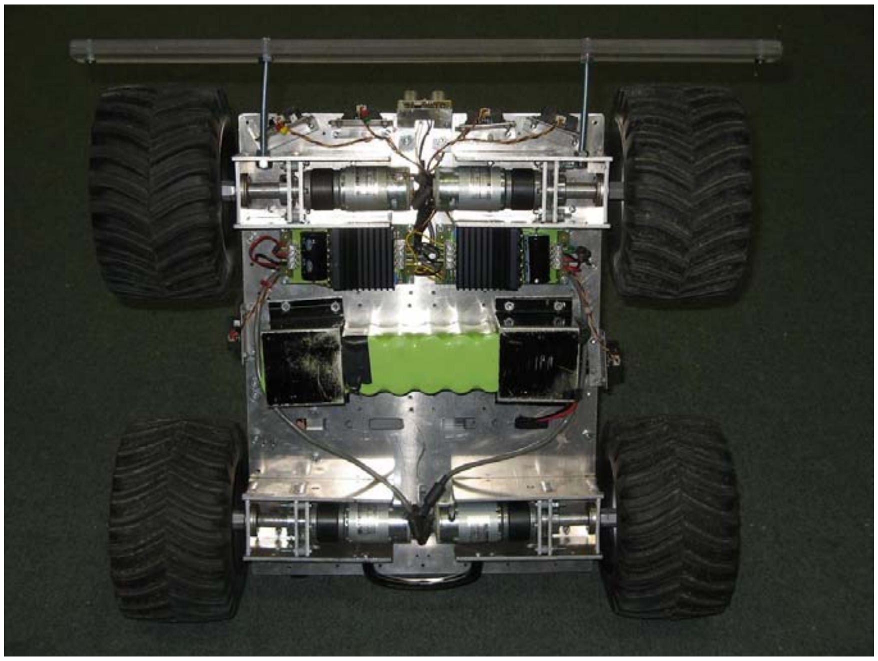


Figure 11. BigFoot viewed from bottom.

2. Motors and Motor Controllers

BigFoot employs four 24VDC 190 RPM geared motors with a maximum torque of 0.49 N-m. The rated motor current is <900mA at full load and <450mA under no load conditions. [9] Each motor drives one wheel. The motors are connected in parallel to two motor controllers. This permits both motors on a side to be driven at the same speed for stable control.

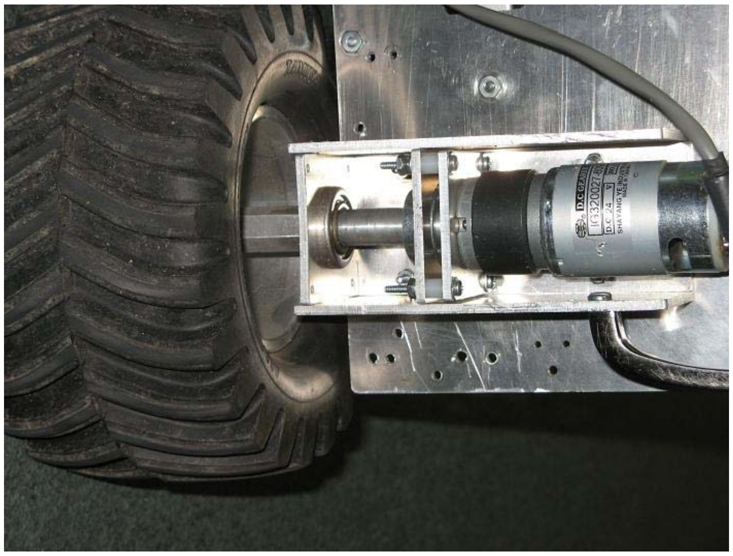


Figure 12. One of four motors.

The motor controllers are 50VDC, 20A H-Bridge motor drivers. The controllers produce 7.8 kHz pulse width modulated power to drive the motors. They require a 5VDC signal for logic circuitry and draw less than 50mA. They also require input voltage between 5 and 50VDC for the motors. The motors and controllers are mounted under BigFoot's frame base permitting maximum ground clearance as shown in Figure 13.

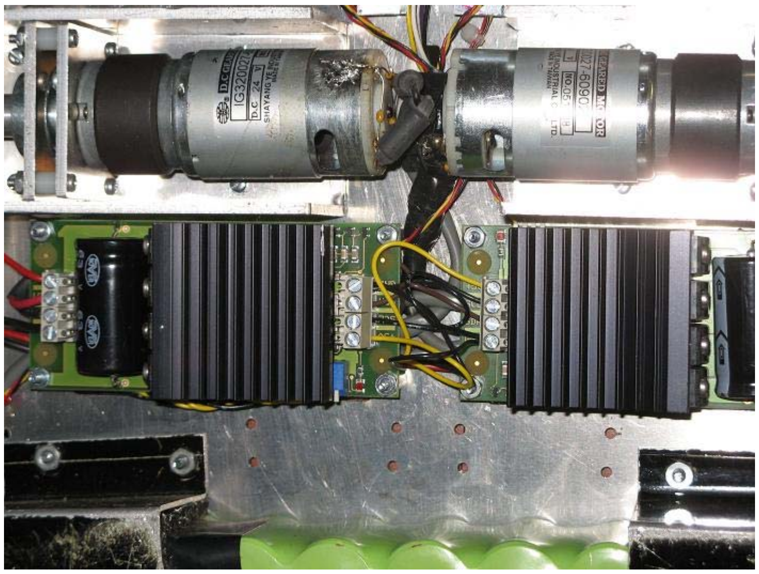


Figure 13. Motors and motor controllers mounted under base.

The controllers have five modes of operation. The motor controllers can operate in I2C mode, 0-2.5-5v mode, 0-5v mode, RC mode, or PWM (Pulse Width Modulation) mode. BigFoot utilizes the 0-2.5-5v mode. In this mode, BigFoot sends an analog signal between 0 and 5 VDC to the motor controller to control the motor speed and direction. When the input voltage is below 2.5VDC, BigFoot travels forward. Between 2.5 and 5VDC, BigFoot travels in reverse. When the motor controller receives an input voltage of 2.5VDC±2.7%, the motors are stopped. Previous projects (Agbot and AGV) utilized an external PWM circuit to drive a motor controller and relied on an external brake line to stop the motors. Stop and slow speed voltages were unsteady and susceptible to noise. This motor controller's stop voltage is reliable and does not require an external brake signal. Similarly the motor controllers have a linear speed response to analog voltage input as shown in Figure 14. BigFoot uses an input voltage range of 0 to 4VDC based on the limits of the onboard computer controlling BigFoot's operation.

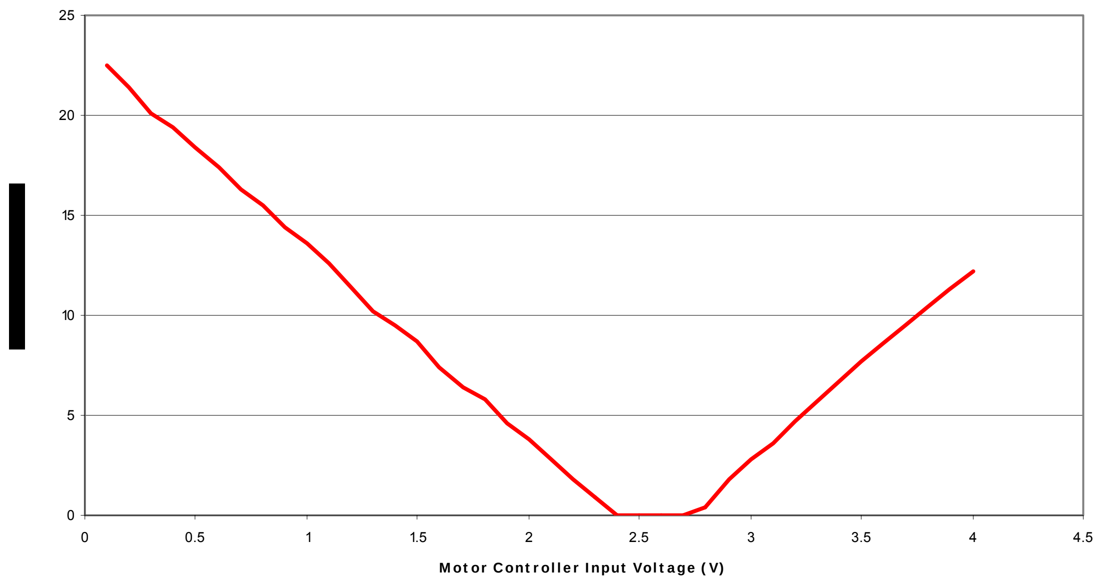


Figure 14. Motor controller and motor speed response.

3. Arm

To accomplish BigFoot’s mission, it will require an arm. A prototype arm constructed from aluminum is mounted on the front, right corner of BigFoot. The arm uses five hobby servo motors to allow 4 degrees of freedom. The base servo is mounted on BigFoot’s platform and provides rotation in the horizontal plane. This servo is connected to a plate that holds two high torque servos operating together to provide rotation in the vertical plane. These shoulder servos provide the main lifting force. The shoulder servos are connected to an 11 inch aluminum bar. At the end of the bar is connected a wrist servo motor that is mounted to rotate in a plane parallel to the shoulder servos. The wrist servo is connected to a hand servo that controls the gripping motion of the hand. The arm is shown in Figure 15. The wrist servo and arm bar weigh approximately 3 ounces each, and the hand and servo combination weighs approximately 2 ounces. The payload mass is estimated to be 8 ounces. This is shown in the simplified free body diagram of Figure 16.

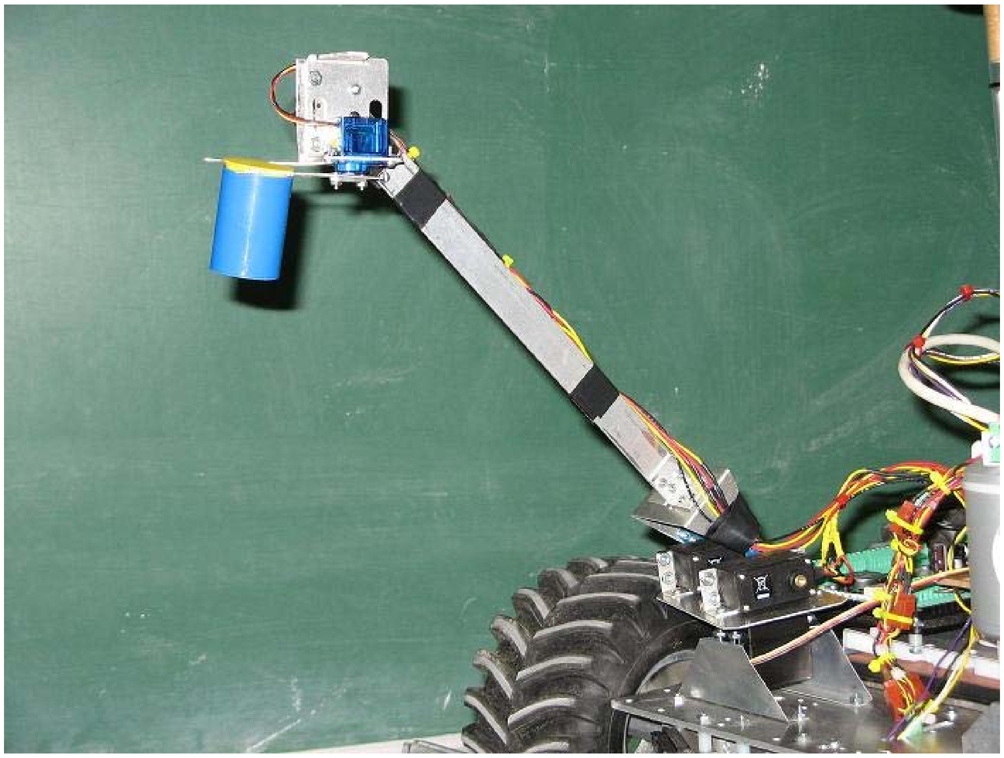


Figure 15. BigFoot's Arm.

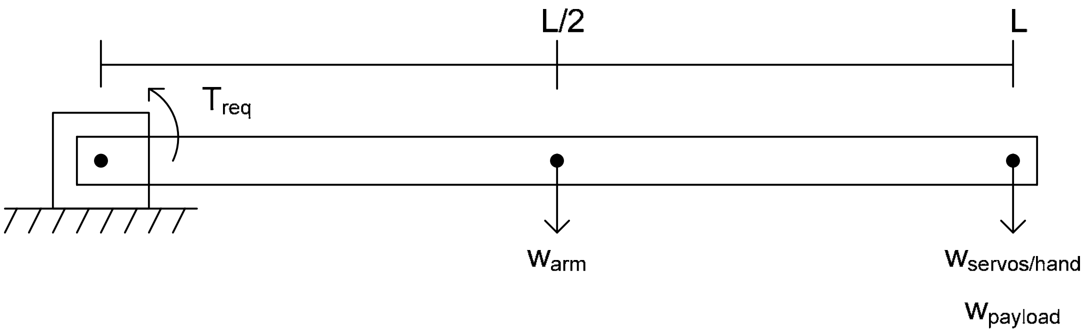


Figure 16. Simplified Arm Free Body Diagram.

In Figure 16, T_{req} is the required torque to lift the arm. The arm is length L, and w_{arm} and $w_{servos/hand}$ arm the masses of the arm and of the wrist and hand servos and hand itself. Mass of the payload is denoted as w_{arm}. The required torque is then calculated using Equation 1.

$$T_{req} = (w_{arm} \cdot L/2) + (w_{servos/hand} \cdot L) + (w_{payload} \cdot L)$$

Equation 1. Arm Torque Equation.

Substituting the design values into Equation 1 yields a required torque of 1.21 N-m. For this reason, two HSR-5995TG Ultra Torque servos are used with a maximum torque of 2.35 N-m each at 6 volts. BigFoot's servos operate at 5 volts and thus require two motors in parallel. The remaining available torque allows for underestimation of payload mass as well as other possible applications.

B. ELECTRICAL

1. Batteries

BigFoot requires 24, 15, 12, and 5VDC power. This power is drawn from separate batteries for the motors and electronics. The motors require high current both in steady state and in surges. The electronics are typically operated at lower current, but have higher overall loading. A rechargeable 24 VDC, 4000 mAhr Nickel Metal Hydride (NiMH) battery pack is used for the motors. This battery pack is mounted on underside of BigFoot in the center for even weight distribution. The battery pack is a 2 x 10 array of C cell batteries and weighs 3½ pounds. The motor battery is shown in Figure 17.

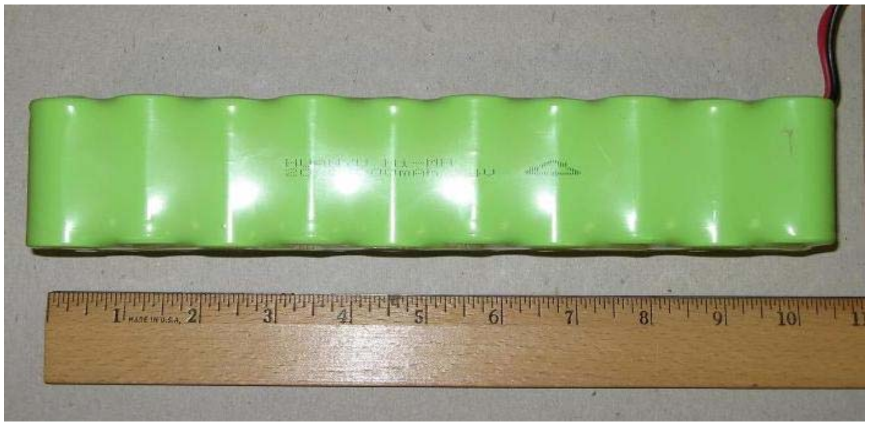


Figure 17. 20 cell, 24VDC, 4000 mAhr, rechargeable NiMH battery pack.

The electronics battery is a rechargeable, 15VDC, 11,000 mAhr Lithium Ion battery. This battery is mounted flat on BigFoot's chassis and weighs 2½ pounds. This battery permits extended on station time while motors are not running. The battery capacity leads to an on station time of approximately 11½ hours. Figure 18 shows the electronics battery.

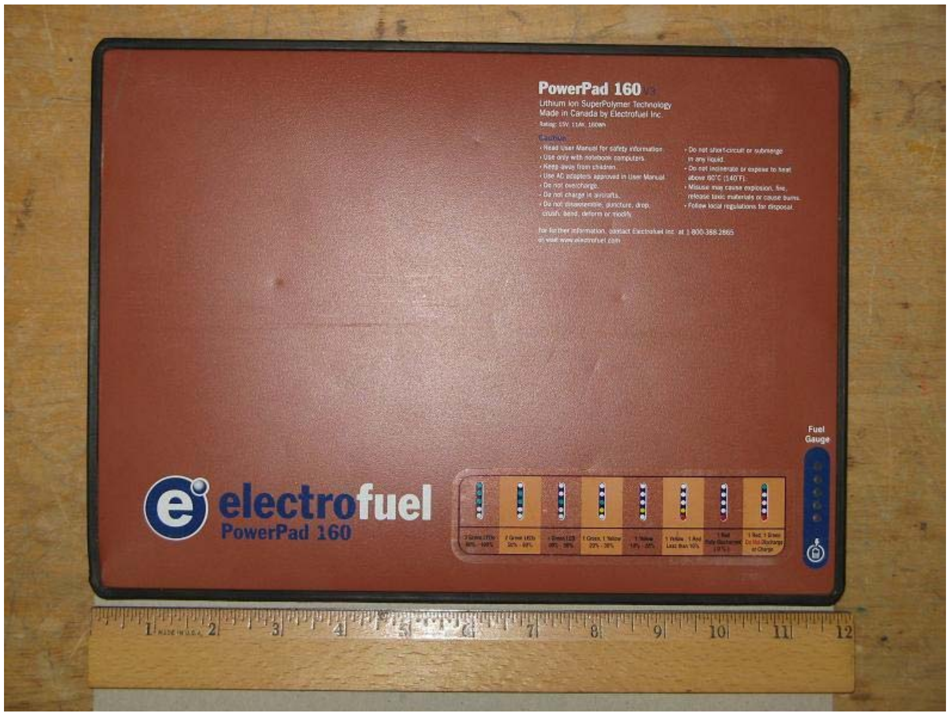


Figure 18.   15VDC, 11000mAHr, rechargeable Lithium Ion battery.

2.   Power Regulation

Figure 19 shows a functional diagram of BigFoot's power regulation system. The electronics power is regulated from 15V to 12V by a standard 7812 voltage regulator. The 12V regulator is a TO-3 package allowing a high input voltage of 35V and an output current of greater than 1A [10]. This regulator is connected to a large heat sink to allow high current operation. 5V power is produced by regulating output from the 12V regulator to 5V using a standard 7805 regulator. The 5V regulator is a TO-220 package connected to a heat sink for high current outputs. A 330μF capacitor is connected across the output of the 5V regulator to prevent any noise or fluctuations on the 5V bus. A second 5V regulator in a TO-3PS package is employed for the camera. The camera requires over 1.5A at 5V. This, with the other loads, would easily overload the primary 5V regulator. Both batteries are connected to a common ground in the power regulation system. Figure 20 shows the primary voltage regulators. The camera voltage regulator is mounted separately.

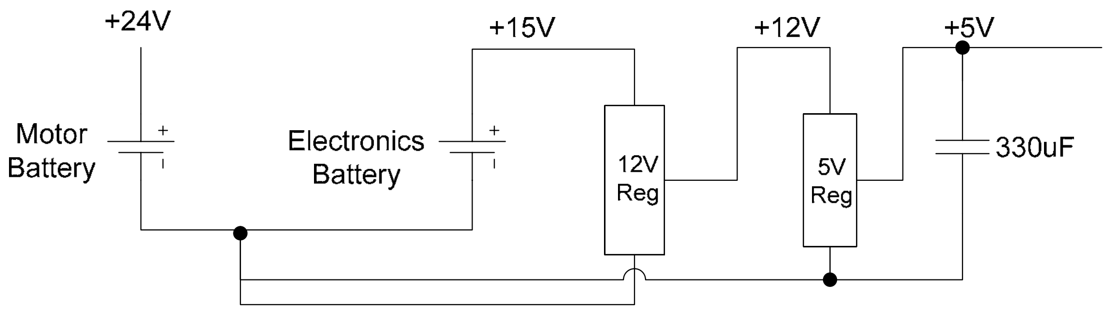


Figure 19. Power Regulation System Functional Diagram.

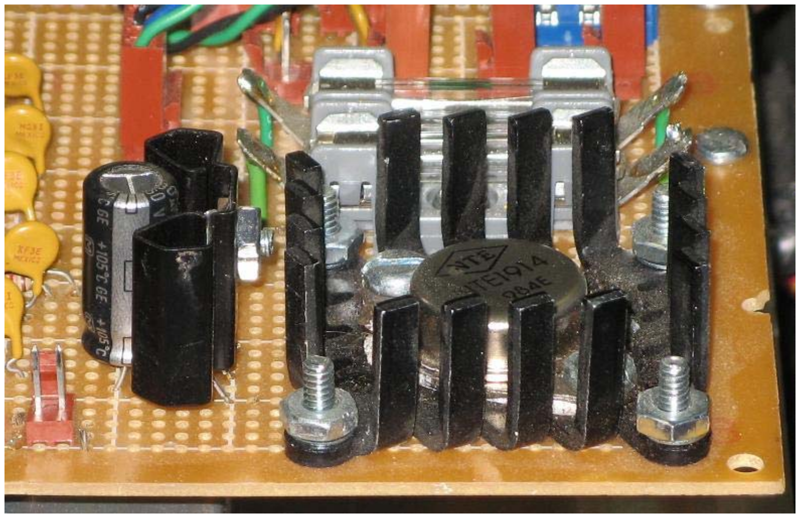


Figure 20. BigFoot's Power Regulation System.

3. Power Distribution

Figure 21 shows a basic schematic diagram of BigFoot's power distribution system. The system is split into two major sections. Power from the motor battery is sent through a set of disconnects to a three position, double pole switch. The switch has a center off position. When the switch is in "ON", the power is directed through a pair of 25A fuses to the motor controllers' input terminals. When the motor switch is in the "CHARGE" position, the battery is connected to a pair of red and black terminals. These terminals are then connected to a battery charger to charge the motor battery. This enables convenient charging without disconnecting or removing the motor battery. The motor switch is shown on the left of the switch panel in Figure 22.

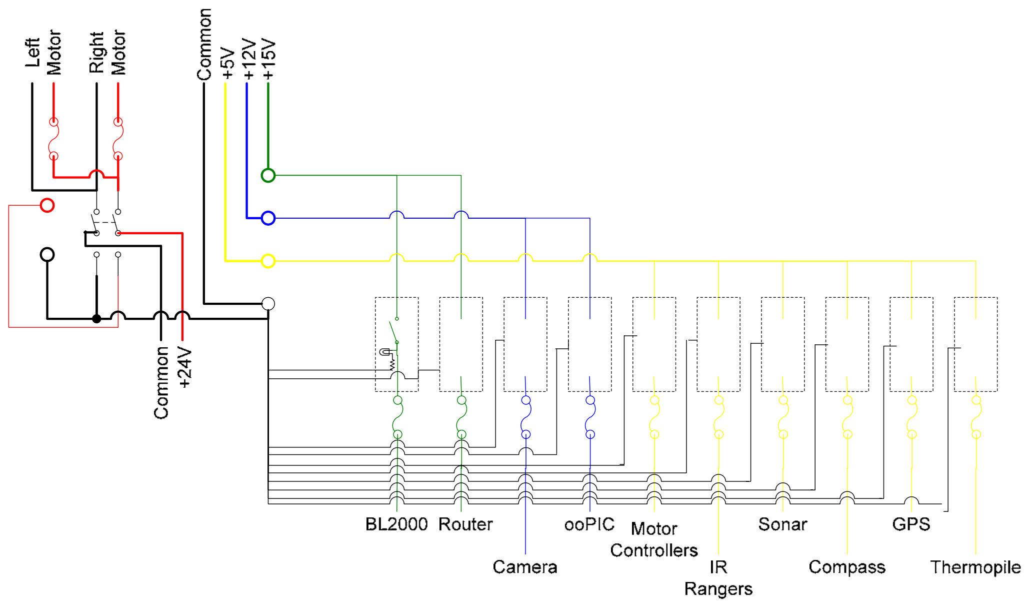


Figure 21. Basic Power Distribution Schematic.

The second major portion of the power distribution system distributes the electronics power. This section distributes power from the electronics battery and 12 and 5V regulators and distributes it via individual switches to BigFoot's various loads. Each of the different voltages is connected to a terminal for convenient testing. These terminals are color coded as is all wiring on BigFoot. 15V power is green, 12V is blue, 5V is yellow, 24V is black, and common is black. Power is sent to switches on the switch panel (Figure 22). The switches are equipped with Light Emitting Diodes (LEDs) to indicate when power is being sent to a load. The output of the switches is sent to the power panel where the power is directed through fuses for the 15V and 12V loads or poly-switches for 5V loads. The fuses and poly-switches are connected by quick connectors to the individual loads for easy removal or testing. All loads connect to the common at the regulator that provides their individual power to ensure a 0V common. Total system loading is listed in Table 2. These currents are measured at 15V. The 5V loads will actually draw three times the listed currents at their operating voltage.

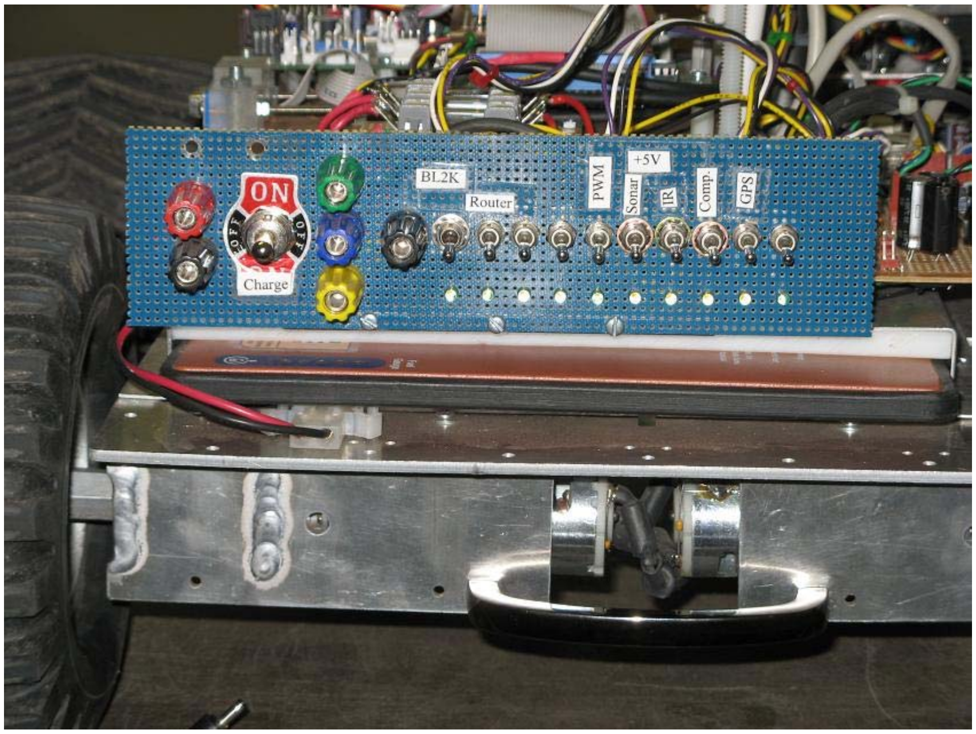


Figure 22. Switch Panel.

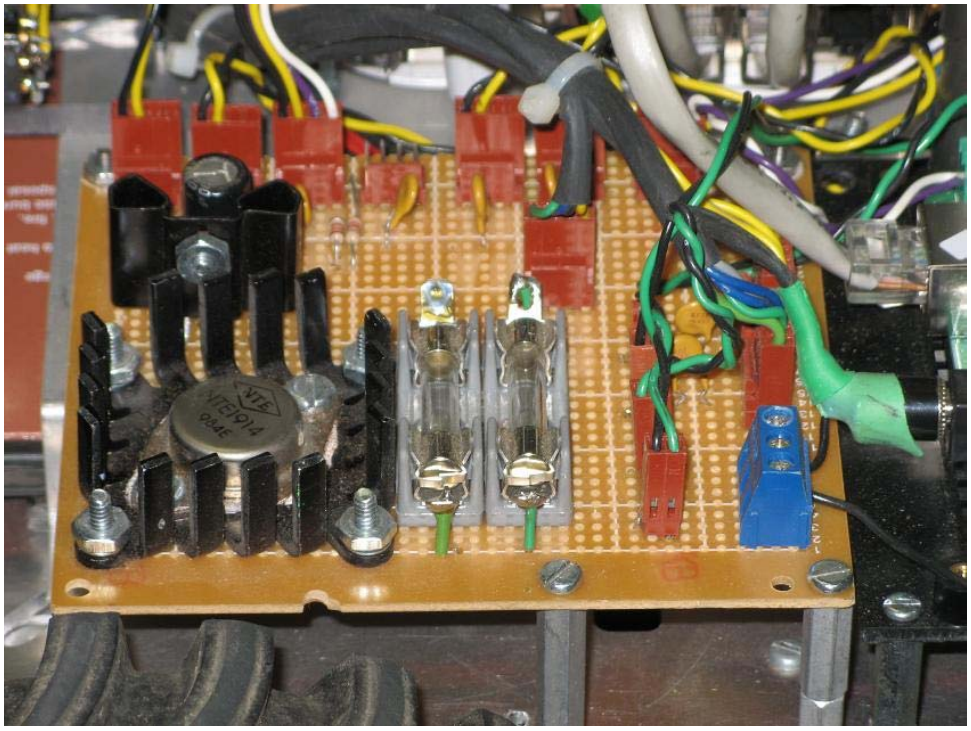


Figure 23. Power Panel.

Load	Current
Voltage Regulators	9mA
BL2000	60 mA
Router	160 mA
ooPIC	100 mA
Motor Controllers	50 mA
IR Rangers	138 mA
Sonar	10 mA
Compass	10 mA
GPS	70 mA
Camera	340 mA
Thermopile	10mA
Total	957mA

Table 2. Electrical Loading.

All connections to the 5V bus also include connections to the I2C bus. The I2C bus uses a pair of 1.2kΩ pull-up resistors connected to the 5V bus. The BL2000 operates as the master device. Currently only the compass, sonar, and

BL2000 are connected to the I2C bus. The I2C bus color scheme uses a white data line using white and a purple clock line. Brown wires are used for signals and data.

C. SENSORS

1. Compass

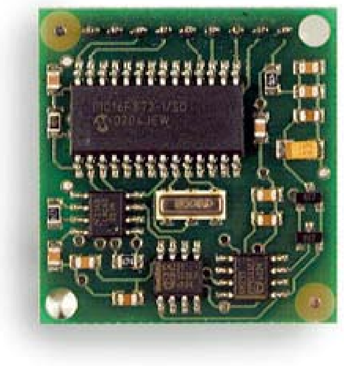


Figure 24. Devantech CMPS03 Electronic Magnetic Compass (From [12]).

BigFoot uses a Devantech CMPS03 electronic magnetic compass (Figure 24). This compass is accurate to within 3 degrees with a resolution of 0.1 degrees [12]. This compass uses two orthogonally mounted Phillips KMX51 magnetic field sensors. The sensors are comprised of four magnetoresistors connected in a Wheatstone bridge configuration. These magnetoresistors are formed from ferromagnetic permalloy, an alloy of 19% Fe and 81% Ni [13], strips that are deposited on a silicon substrate with a pair of coils for compensation and field flipping. When the permalloy strips are deposited onto the silicon substrate, a strong magnetic field is applied parallel to the strip axis to establish a preferred magnetic direction within the strip which is aligned in the direction of current flow. When an external magnetic field is applied, a net field is established at an angle α with respect the direction of current flow as shown in Figure 25.

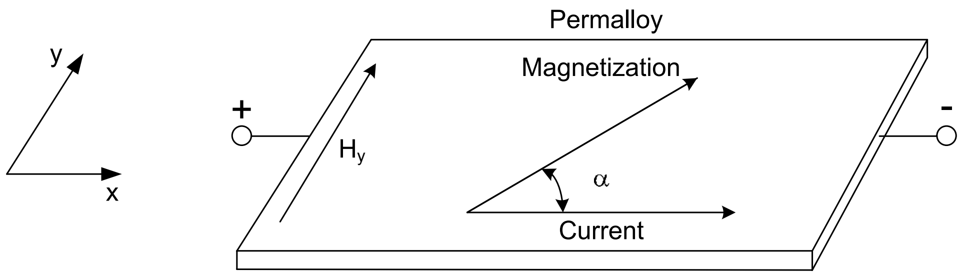


Figure 25. Magnetic Effect on Permalloy (From [13]).

This results in electrons traveling in curved, rather than straight, paths across the permalloy strip due to the Lorentz force (Equation 2) effect on the electrons in the permalloy. This results in a higher resistance which is proportional to the angle α according to Equation 3.

$$\overline{F} = q\left(\overline{E} + \overline{v} \times \overline{B}\right)$$

Equation 2. Lorentz Force Equation.

$$R = R_0 + \Delta R \cdot \cos^2 \alpha$$

Equation 3. Magnetoresistance Equation (From [13]).

This effect can lead to up to 3% variance resistance. This effect is linear under small magnetic fields less than 10 Gauss. Note that the earth's magnetic field is on the order of 0.5 Gauss. This resistance change causes the Wheatstone bridge to be imbalanced and thus creates a potential proportional to the magnetic field orthogonal to the sensor strips. Two sensors are employed perpendicular to one another to determine vector components of the earth's magnetic field. Together the components form a two dimensional vector for the earth's magnetic field [14]. This is then processed by the compass electronics to produce an 8-bit word which is passed when the I2C master device calls address 0XC0. The compass is mounted to a mast approximately four inches above the platform base to prevent interference from the motors' magnetic fields.

2. Ultrasonic Range Finder

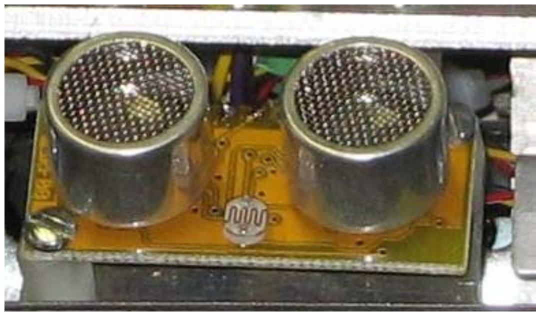


Figure 26. Devantech SRF08 Ultrasonic Ranger.

BigFoot uses two different types of sensors for collision avoidance. The primary is a Devantech SRF08 ultrasonic range finder (Figure 26). The range finder is constructed from a pair of piezoelectric transducers. The ceramic piezoelectric crystal flexes to create a pressure pulse when an electrical signal is applied. These pressure pulses create a 40 kHz sound pulse in a conical beam with the beam angle shown in Figure 27.

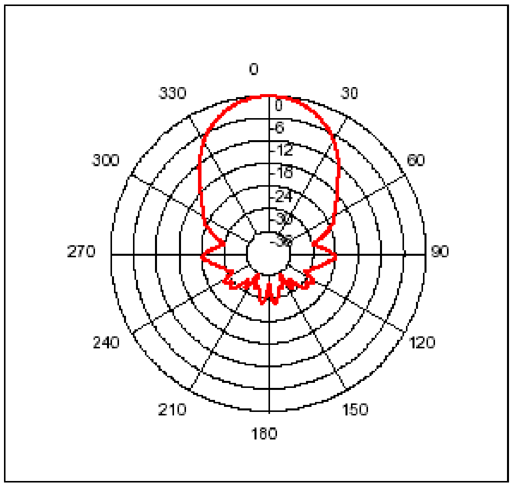


Figure 27. SRF08 Beam Pattern (From [15]).

The pulse is reflected from any objects in the beam path. The ranger then receives portions of the reflected wave. The time difference between the pulse's emission and reception of the reflected wave is proportional to the distance to the obstacle as shown by Figure 28.

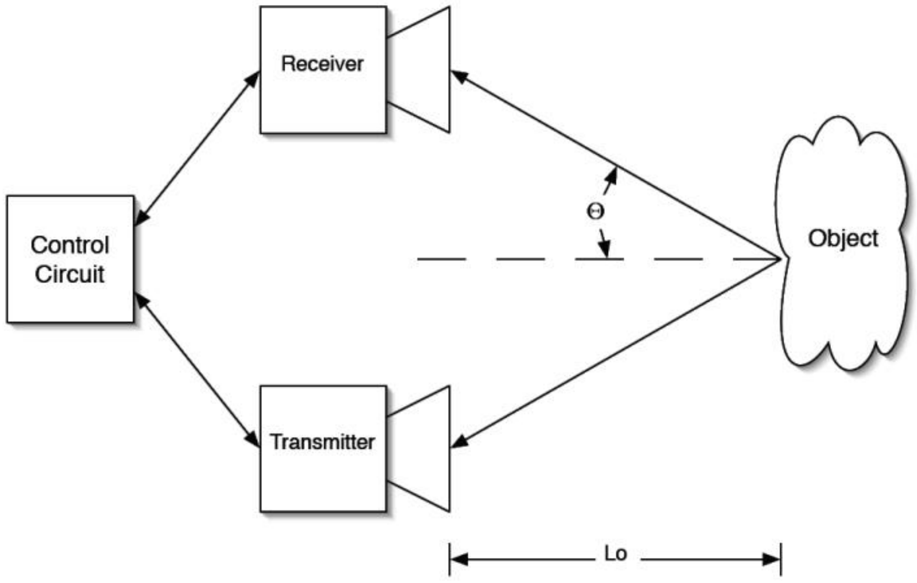


Figure 28. Ultrasonic Range Finder Operational Diagram (From [16]).

$$L_o = \frac{vt \cos \Theta}{2}$$

Equation 4. Ultrasonic Range Equation (From [16]).

Equation 4 is used to calculate the range. In this equation, L_0 is the range to the obstacle, v is the speed of sound in air, and t is the time between pulse emission and detection of the reflected wave. Θ is the angle of incidence of the wave to the obstacle. The SRF08 can detect obstacles at a maximum range of 6 meters as shown in Figure 29. Due to the approximate beam width of 55 degrees at -6 dB, the ranger is mounted at an upward angle of approximately 30 degrees to prevent reflections from the ground causing indications of false obstacles. The ultrasonic ranger translates the calculated distance and reports either time, or distance in centimeters or inches. This is converted it to an 8 bit ASCII word that is read by the calling the I2C address 0xE0.

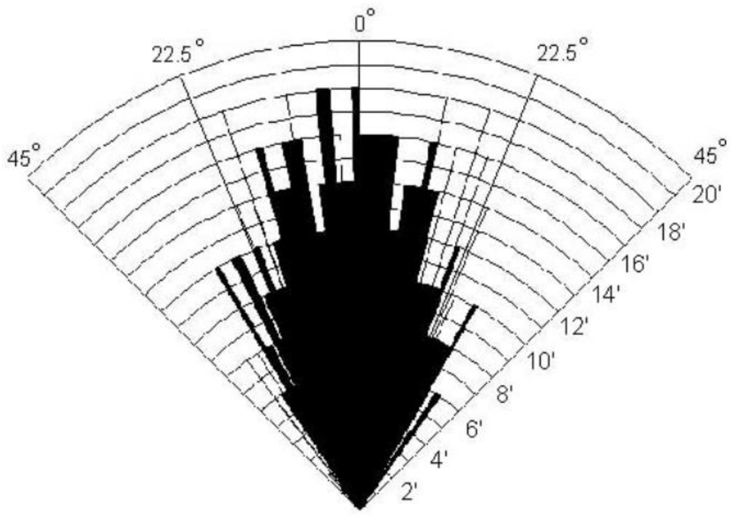


Figure 29. SRF08 Detection Pattern (From [15]).

3. Infrared Rangers

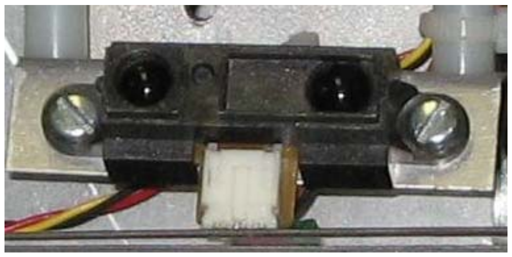


Figure 30. Sharp GP2D12 Infrared Ranger.

BigFoot employs six Sharp GP2D12 Infrared (IR) rangers (Figure 30) as a secondary means of obstacle detection as well as for decision making purposes. Four IR rangers are mounted on the front and two are mounted on the sides as shown in Figure 31.

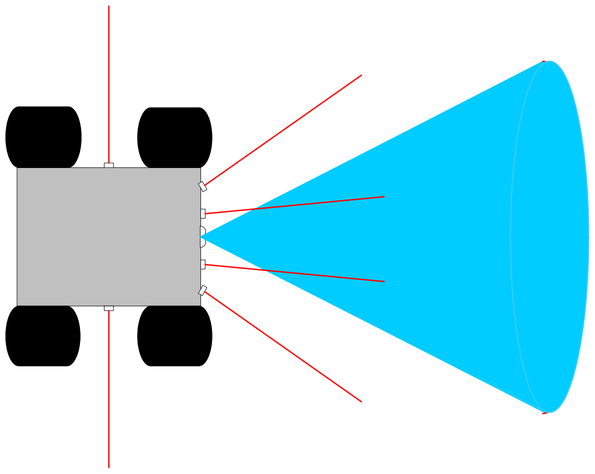


Figure 31. Collision Avoidance Sensor Locations.

In Figure 31, the IR rangers are shown with the red lines representing their range of coverage. The ultrasonic ranger is represented by the blue cone. The four forward facing rangers are used to detect collision threats in front of BigFoot while the two side-mounted rangers are used to determine which direction is clear when turning to avoid obstacle. IR rangers continuously transmit an 850nm wavelength light beam [17]. When an obstacle is present, the beam is reflected back to the ranger where it is detected by a linear CCD array detector [18]. This determines the angle at which the beam is returning. This angle is proportional to the distance from the target. Figure 32 and Equation 5 show the relationship between the detection angle and target distance.

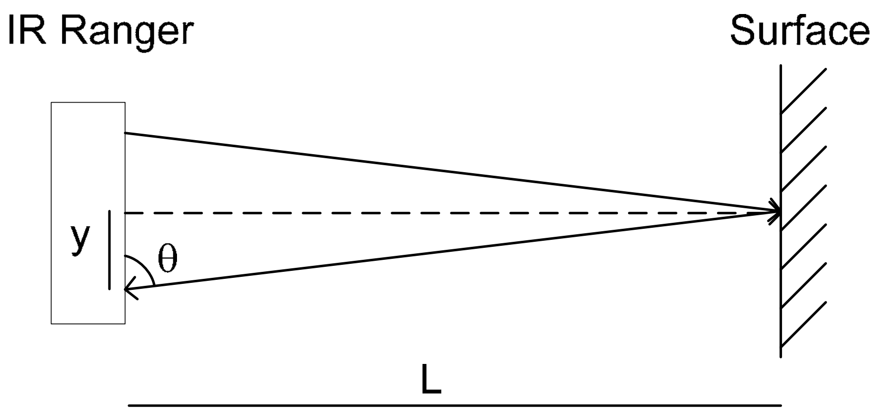


Figure 32. IR Ranger Detection Configuration.

$$L = y \cdot \tan\theta$$

Equation 5. IR Ranger Equation

In Equation 4, the distance to the object is L, y is the known distance from the center of the detector to the sensor, and θ is the angle measured by the CCD sensor array. The GP2D12 produces an analog voltage that is non-linearly related to the object range. This relation is shown in Figure 33. As shown, the maximum effective range is approximately 80cm with a minimum range of 10cm.

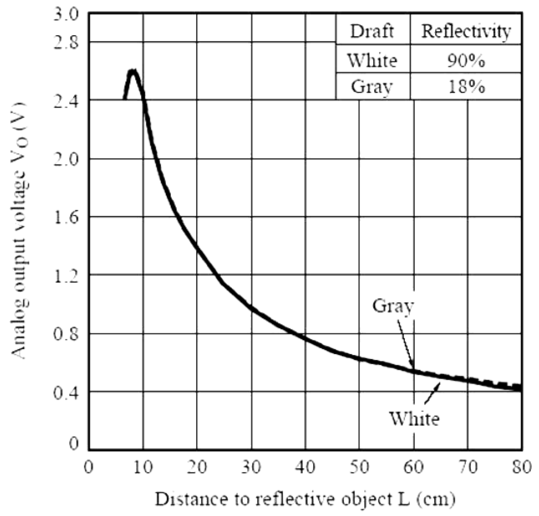


Figure 33. IR Ranger Output Voltage versus Object Distance (From [17]).

4. Thermopile

Additional sensing capability is available by use of a Trekker Thermal Array Sensor TPA81 (Figure 34). This sensor is a linear array of eight thermopiles. This array produces an eight pixel temperature reading with each pixel representing a solid angle of 5.12 x 6 degrees. The thermopiles are a series of interconnected thermocouples. These thermopiles respond to IR radiation in the wavelength range of 2 to 22μm [19].

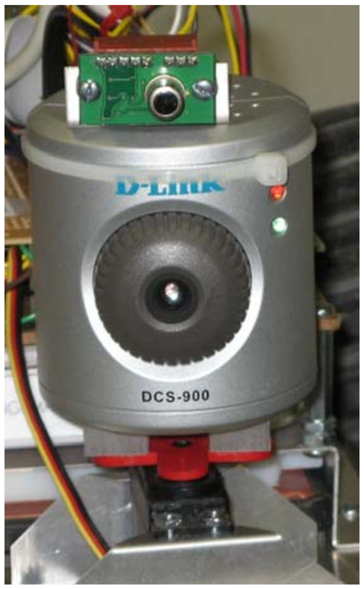


Figure 34. Thermal Sensor Mounted on Camera.

The Thermal Sensor is mounted directly above the camera to permit thermal information being correlated with visual images by the operator. The thermal sensor operates on the I2C protocol and is programmed to address 0xD0. Each of the eight registers is read individually and returns the temperature 0 to 255 Celsius.

5. Camera

A commercial off the shelf D-Link DCS-900 internet camera is used to send real time streaming video and still pictures to the operator. The camera is a 2 ½ x 2 ½ x 2 ¾ cylinder that weighs 0.61 pounds. It plugs directly into BigFoot's onboard router and has its own static IP address. The camera has a manual focus, but the standard setting is acceptable. The lens has a 6.0mm focal length, automatic brightness and contrast controls and an automatic frame rate. The camera can transmit 640x480 and 320x240 pixel images and video [20]. The camera utilizes a separate voltage regulator due to drawing approximately 1.5 Amps at 5V. The camera is mounted on a servo to permit panning to view BigFoot's surroundings without having to move. The camera is shown in Figure 34.

6. Global Positioning System (GPS)

BigFoot exclusively uses the Garmin GPS 18-5Hz for navigation. The GPS receives 5 location reports per second and can receive data from ten satellites at one time. BigFoot's GPS unit is shown in Figure 35. The GPS is mounted at the top of the communications antenna to prevent interference from BigFoot's router. This is required as the GPS satellite signals and BigFoot's wireless communications both operate at 2.4 GHz. The GPS reports BigFoot's location in a Garmin proprietary data format using standard RS-232 serial communications. The data is received by the BL2000, processed, and is passed on to the user interface laptop. The location report is in standard latitude and longitude coordinates [22] and includes the number of satellites currently being tracked as well as the time of the last position fix.

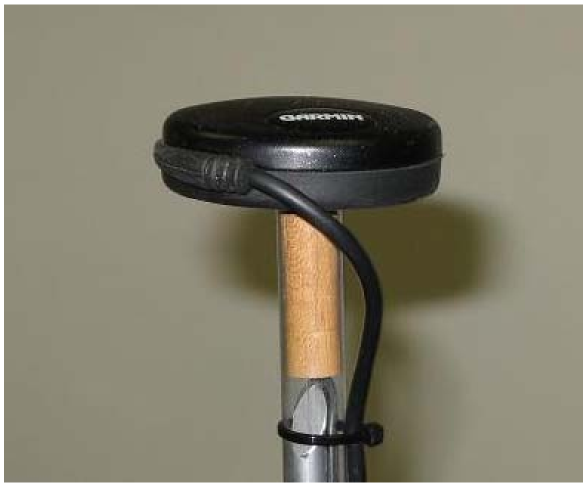


Figure 35. Garmin GPS 18-5Hz.

The GPS capability includes the use of WAAS (Wide Area Augmentation System) while in North America. Originally intended as a system to improve GPS accuracy for aviation, WAAS has transitioned to also improve land based applications. There are only two WAAS satellites in operation (one over the Atlantic and one over the Pacific Oceans) [21]. The two satellites work 25 ground stations to calculate possible GPS errors such as clock drift, orbital errors, atmospheric delays, etc. These satellites broadcast the errors which are then compensated for by the GPS receiver [7].

D. MICROCONTROLLERS

1. BL2000

BigFoot's primary computer processor is the Z-World BL2000 Wildcat. The BL2000 is a single-board 22.1 MHz computer with a Rabbit 2000 microprocessor. The BL2000 has 4 analog and 11 digital inputs as well as 2 analog and 10 digital outputs. The BL2000 also has four serial ports, 128K of static RAM, and 256K of flash memory [23]. Figure 36 shows BigFoot's BL2000. The computer is connected via RJ 45 connections to the router for two way communications with the operator laptop. It is also connected to the OOPic via four of the digital output lines. Table 3 lists all connections to the BL2000. The BL2000 controls all sensors with the exception of the camera and analyzes sensor data to control BigFoot. The processor is programmed with dynamic C code (see Section IV) via a serial cable connection which is permanently mounted on BigFoot. The BL2000 utilizes 2 digital input and two digital output ports to serve as the master device for the I2C bus.

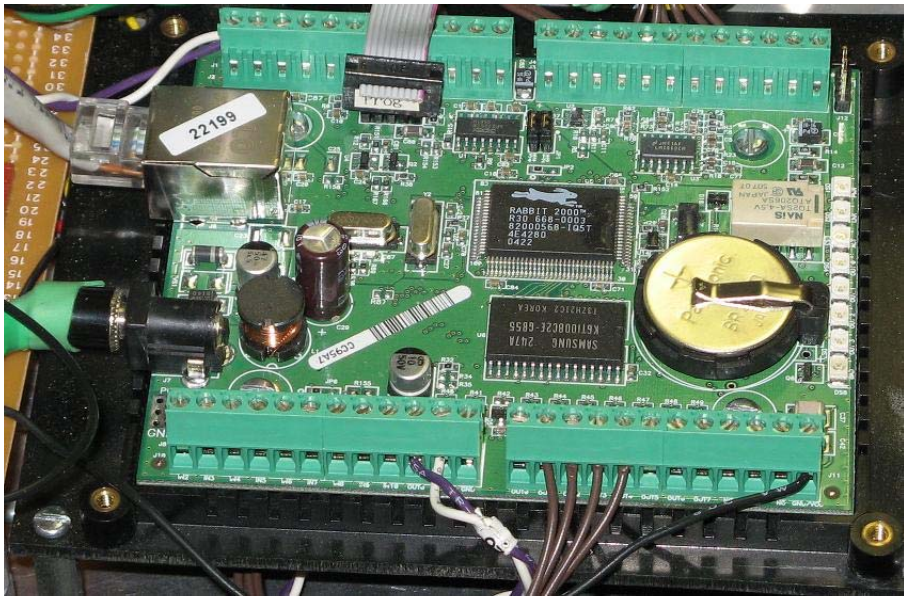


Figure 36. BL2000 Rabbit Microprocessor.

BL2000	Load	BL2000	Load
Analog Output 1	Left Motor Controller	Digital Input 0	I2C Data Line
Analog Output 2	Right Motor Controller	Digital Input 1	I2C Clock Line
Analog Input 3	Left Side IR Ranger	Digital Output 1	OOPic I/O Group 3 Pin 0
Analog Input 4	Left Corner IR Ranger	Digital Output 2	OOPic I/O Group 3 Pin 1
Analog Input 5	Left Center IR Ranger	Digital Output 3	OOPic I/O Group 3 Pin 2
Analog Input 6	Right Side IR Ranger	Digital Output 4	OOPic I/O Group 3 Pin 3
Analog Input 7	Right Corner IR Ranger	Digital Output 8	I2C Data Line
Analog Input 8	Right Center IR Ranger	Digital Output 9	I2C Clock Line
TX2	GPS Receive Line	RX2	GPS Transmit Line

Table 3. BL2000 Connections.

2. OOPic

BigFoot's secondary processor is the Object Oriented Programmable Interface Controller (OOPic). Specifically, BigFoot utilizes an OOPic II+. The OOPic II+ utilizes B.2+ firmware which includes seven 10-bit Analog to Digital (A2D) channels, 86 bytes of object memory, 72 bytes of variable memory, 8K of program code space, and 256 bytes of internal EEPROM space. The processor is a PIC16F877 microcontroller operating at 20MHz [24]. The OOPic is used to drive the servos for BigFoot's arm control. The OOPic is perfect for this due to the ability to drive multiple servos concurrently with its onboard pulse width modulators. The OOPic is connected via a 40 pin parallel cable to an adapter board which provides convenient connections to many standard small robotic project components (such as IR Rangers, Sonar, Compass, servos, etc). The BL2000 is also connected to the OOPic via the 40 pin parallel cable. The BL2000 requires pull-up resistors on its digital outputs and are mounted on an interface board where the BL2000 is connected to the 40 pin ribbon cable. The OOPic, adapter board, and interface board are shown in Figure 37. The 40 pin ribbon cable is removed for clarity.

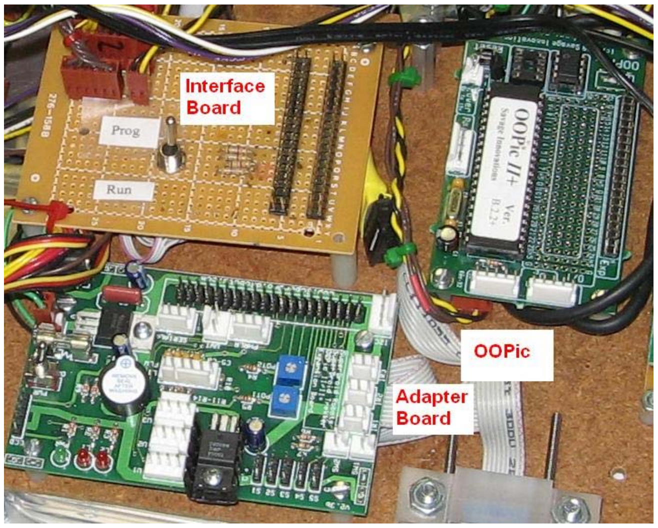


Figure 37. OOPic, Adapter Board, and Interface Board.

E. COMMUNICATIONS

1. Router

BigFoot communicates via a Netgear Rangemax 240 wireless router. The router operates with both IEEE 802.11b and 802.11g standards at 2.4 GHz. This router has four RJ 45 connections to be used for wired connections. Two of these are used for the BL2000 and the camera leaving two connections for additional components. The network is password protected to prevent uncontrolled access to BigFoot or the camera. All communications utilize standard UDP protocol. Figure 38 shows the router used by BigFoot.

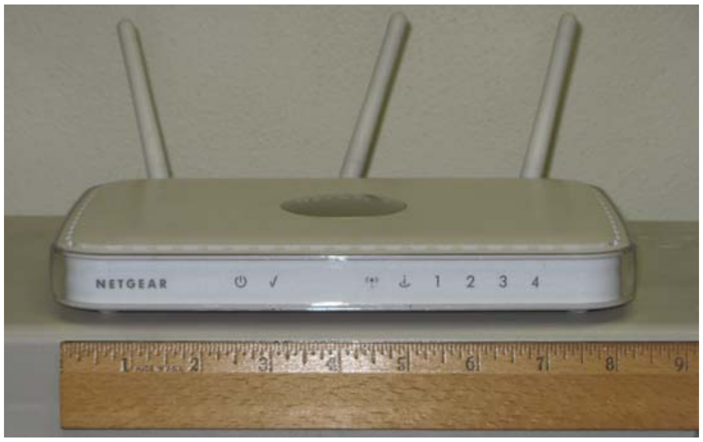


Figure 38. Netgear Rangemax 240.

2. Antenna

Previous projects have been limited in range due to communications range of the router. To alleviate this, BigFoot has been equipped with a D-Link 2.4GHz Omni-Directional 7dBi antenna. This 11 inch antenna is connected directly to the router’s center antenna connection via a SMA connector. The antenna is vertically polarized and has an impedance of 50 ohms [25]. BigFoot’s antenna is shown in Figure 39.

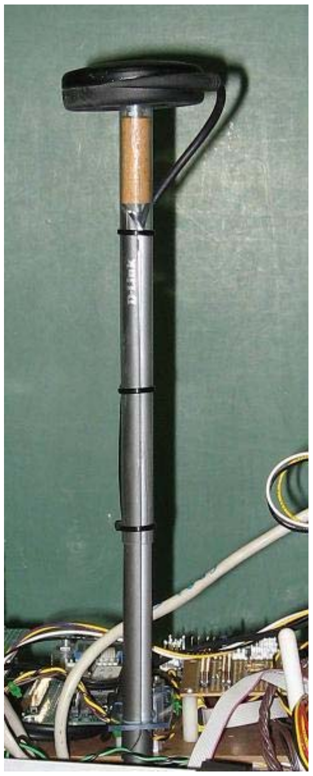


Figure 39. D-Link 7dBi Antenna.

IV. BIGFOOT'S PROGRAM

BigFoot's operation is directed by the BL2000. The BL2000 is programmed using Dynamic C. Dynamic C is a Z-World version of the C programming language. BigFoot's complete Dynamic C code is included in Appendix A.

A. GENERAL PROGRAM OVERVIEW

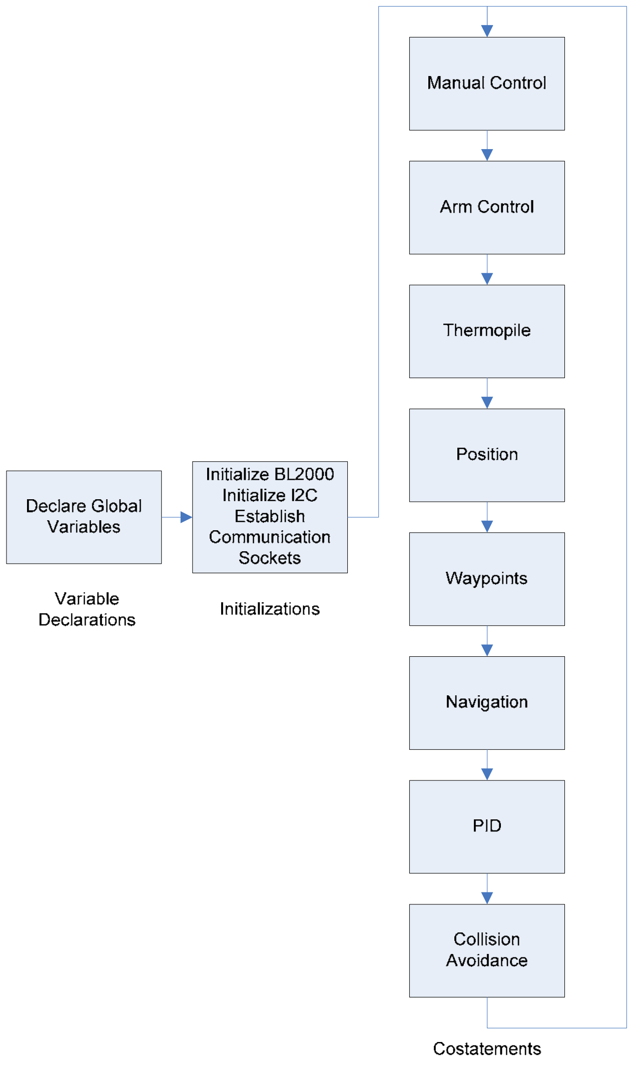


Figure 40. Basic Program Flow.

Figure 40 shows the overall program flow. The program starts with declaring global variables and function declarations. A small amount of assembly language code is included to establish the I2C clock at 100 KHz. The Rabbit 2000 microprocessor includes built in I2C functionality, but the BL2000 does not offer access to these ports. The declaration portion of the program defines port operations for I2C functions on non-I2C ports.

The second major portion of the program is the initialization phase. In this section, the BL2000 is initialized. The I2C communications are initialized and communications sockets are opened. Five communications ports are used in BigFoot's communication scheme, three for receiving data, and two for sending data. The receive ports are designated for waypoint instructions, manual controls, and arm control instructions and the transmit ports are designated for GPS and error data. The compass and thermopile data are passed in the error stream and are separated from error messages in the GUI. All communications are with a UDP protocol.

The remainder and bulk of the program is dedicated to the costatements. Costatements are a Dynamic C specific functionality that allows the program to wait to perform the instructions until a specific event occurs. This allows the program to perform other functions while waiting on that event. Each costatement is checked for its qualifying condition and either executed or skipped until the program loops to check again.

BigFoot requires eight costatements. The first checks the manual control port to determine if any manual control orders are being sent. If so, BigFoot is placed in manual mode and manual controls are implemented. If there is no data in the manual control stream and the robot is not already in manual, this costatement is skipped. The second costatement is for arm control. If the robot is in manual mode, this costatement will execute orders to control the arm. The third costatement is the compass statement. This checks the compass and reports the heading to the GUI. This is performed every loop of the program. The program then checks the waypoint stream to determine whether autonomous orders are being sent. If the waypoint stream has orders, the robot is placed in auto and the rest of the costatement is performed. The next costatement is for the GPS. This is performed every loop through the program. The remaining costatements are only performed while in autonomous mode. These are the navigation orders, the controls for turning, and the obstacle avoidance controls.

B. MANUAL OPERATION

Figure 41 shows the program loop while in manual mode of operation.

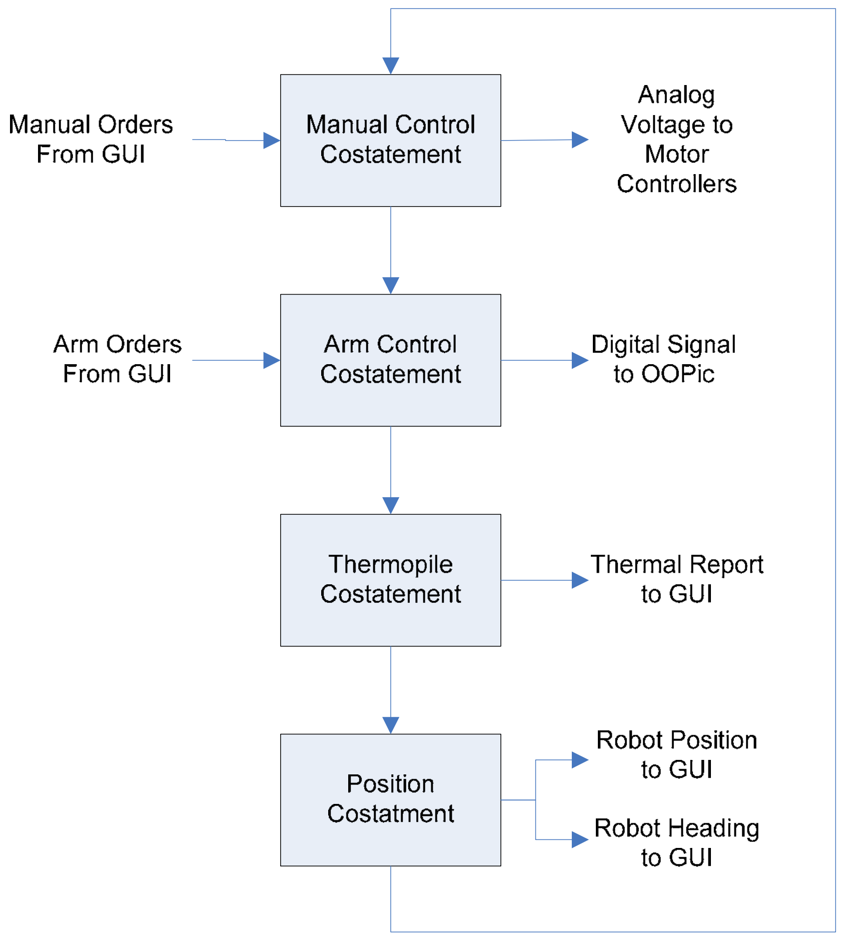


Figure 41. Manual Control Program Flow.

1. Manual Control Costatement

When the robot is in manual operation, it is waiting on one of two inputs from the GUI. It waits for manual orders from the manual control port. The manual control stream is comprised of ASCII words for left and right side motor controller voltages delimited by a space. These voltages are directed to the analog output ports to control the motor controllers. If the robot was not previously in manual mode when this order was received, the robot is placed in manual at this time. The BL2000 will maintain a constant output to the motor controllers until a new manual control order is received.

2. Arm Control Costatement

Secondly, the program is awaiting orders on the arm control port. It checks the arm control port to determine whether any arm orders have been received. If there are no new arm orders, the program continues to the next costatement. If an arm order is received, the program will convert it from ASCII format to an integer. The order will be set to 0 if it is any value greater than 12, as those are not defined. The program then converts the order to a 4 bit digital word that is placed on digital output ports 1-4. These ports are connected to the OOPic. After a delay 300 milliseconds, the digital output ports are set to 0 to prevent further operation of the arm. This limits the speed of the arm movement and permits changing the arm position in 3-degree increments.

3. Thermopile Costatement

The third costatement in the manual control loop is the thermopile costatement. The program calls the thermopile function to read each of the 8 thermopile registers. This data is then converted to ASCII format and placed in a string with the leading keyword “THERM”. This string is placed in the error buffer and is transmitted to the GUI on the error port.

4. Position Costatement

The final costatement is the position costatement. This costatement is enacted in both manual and autonomous modes. It calls the compass function, which orders the compass to report current heading. The compass function calls a series of I2C functions to read the compass and returns a value between 0 and 255 proportional to heading. This is converted to 0 to 360 degrees. The costatement then adds the compass heading to an ASCI string with the leading string “Compass”, which is placed in the error buffer to be transmitted on the error port to the GUI. The costatement then reads the data stream from the GPS receiver and calls functions to convert the data into standard integer and floating point data types for convenient calculations. The costatement sends the GPS data to the GPS port to be sent to the GUI. The current and previous GPS positions are compared to determine whether BigFoot is moving. If the position is exactly the same for 6 program loops times, the robot is placed in manual mode and stopped.

C. AUTONOMOUS OPERATION

Figure 42 shows the program loop while in autonomous mode of operation.

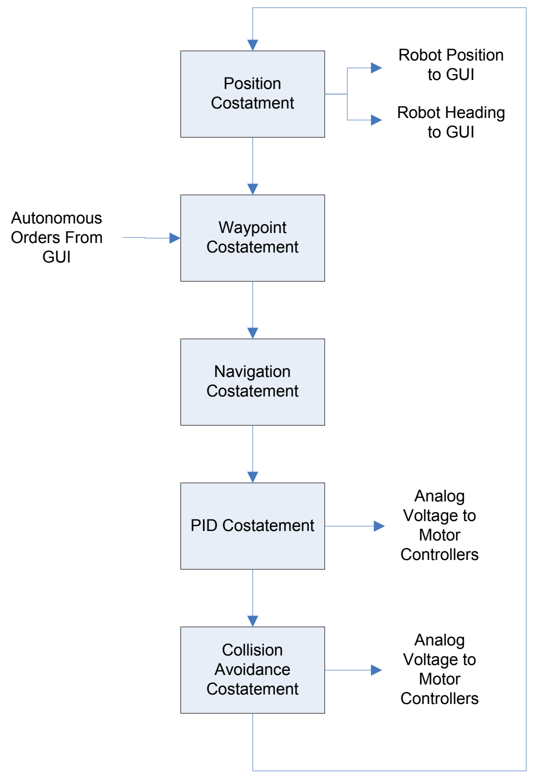

Position Costatement
Robot Position to GUI
Robot Heading to GUI
Autonomous Orders From GUI
Waypoint Costatement
Navigation Costatement
PID Costatement
Analog Voltage to Motor Controllers
Collision Avoidance Costatement
Analog Voltage to Motor Controllers

Figure 42. Autonomous Operation Program Loop.

1. Position Costatement

In autonomous operation, the program loop begins with the position costatement. This is the same costatement described in the manual control section. However, in autonomous mode, the information from the GPS and compass is no longer just passed to the GUI for display. This data is stored and used later for navigation.

2. Waypoint Costatement

The waypoint costatement is the second in the autonomous control loop. The program checks the waypoint port to determine whether any waypoint orders are being received. The waypoint stream is tokenized and stored in an array. The array is sized to accept a maximum of 9 waypoints. This is also the limit in the GUI. The latitude and longitude components are converted into double format while the action component is stored as a character. At the end of the costatement, the program directs the motor controllers to drive forward for 500 milliseconds. This starts BigFoot traveling in a forward direction to prevent interference with the compass due to an electromagnetic surge from the motors.

3. Navigation Costatement

The navigation costatement calculates the necessary course for BigFoot to travel to reach the next waypoint. This is accomplished by first converting latitude and longitude into minutes and decimal minutes. These are then used to calculate the range to the next waypoint using a conversion factor of 2000 to convert from minutes to yards and applying the Pythagorean Theorem to create a straight line range to the next waypoint. If this distance is less than 5 yards, BigFoot determines that the waypoint has been achieved. This range is chosen due to accuracy of the GPS unit without WAAS available. When a waypoint is achieved, the BigFoot will take action as assigned by the operator. This could include stopping, turning, or returning to its origin. BigFoot will also pass a message on the error port to the GUI to alert the operator that a waypoint has been reached or to alert if an error has occurred, such as an invalid waypoint. In the event an error does occur, BigFoot is placed in manual control and is stopped. If BigFoot is not within 5 yards of the waypoint, the necessary course to reach the waypoint is calculated by using trigonometry and logic.

4. PID Costatement

The PID (Proportional-Derivative-Integral) costatement is next in the autonomous loop. In order to prevent overshoot or unstable operation while turning, a PID control is used for turns while driving and while turning from a stationary position. PID control also allows BigFoot to smoothly return to a course while moving to minimize jerky operation.

The program first calculates a heading error. This is the difference between the current heading and the heading necessary to achieve the next waypoint. This heading error first must be used to determine the direction to turn. This is accomplished by using logic to produce Equation 6.

$$Heading \quad Error \geq 180^\circ \quad or \quad -180^\circ < Heading \quad Error < 0^\circ$$

Equation 6. Heading Error for Turn Logic.

When this condition is satisfied, BigFoot will turn left. For all other possible heading errors, BigFoot will turn right. This results in the shortest turn for any possible heading error. Turn direction is then stored as a 0 or 1 value for use later. If the heading error is less than 5 degrees in magnitude, the heading error is defined as 0 and BigFoot continues on its current course. This is necessary to prevent unstable operations due to compass accuracy and stop voltage dead band for the motor controllers.

The heading error is normalized to a range of -180˚ to 180˚. This is then used in a calculation to produce a voltage variable that is proportional to the heading error. This is done by Equation 7.

$$pScale = Error \cdot \frac{2.4volts}{180^\circ}$$

Equation 7. Proportional Scale Equation.

The differential component is calculated similarly. The differential component is proportional to the time rate of change of heading error. This is accomplished by subtracting the heading error from the last loop from the current heading error. The time component is accounted for by the program loop calculating the error once per cycle. Equation 8 is the differential scale equation.

$$dScale = (Error - previousError) \cdot \frac{2.4volts}{180^\circ}$$

Equation 8.   Differential Scale Equation.

The final component is the integral component. This is a time compounded variable related to the cumulative proportional error. The proportional component is added to the integral component each loop to produce a voltage variable proportional to cumulative heading error. Equation 9 is the integral scale equation.

$$iScale = pScale + iScale$$

Equation 9.   Integral Scale Equation.

These components are multiplied by weighting factors to tune the robot to ensure a critically damped turn. If the proportional component is incorrect, the robot will possibly overshoot the target heading if it is too large or could have a heading offset if it is too small. If the differential component is too large, the robot may be overdamped and be unable to reach the target heading. If the differential component is too small, the robot's turn may become very unstable causing the robot to oscillate many times before reaching its destination heading. If the integral component is too small, the robot will not be able to reach its destination course. As a result, the weighting factors must be carefully chosen. In the past, they have been chosen by a method of trial and error, but recent NPS research has created a computer simulation to choose values that result in the most efficient turn possible [26].

$$insidevolts = P \cdot pScale + I \cdot iScale + D \cdot dScale$$

Equation 10. PID Equation.

The weighting factors are multiplied to the appropriate voltages (proportional, differential, or integral) and the three components are added as shown in Equation 10. This is the total turn voltage. This voltage is applied to the inside wheels to slow them. The outer wheels continue to drive at full forward voltage. This allows the robot to continue forward motion while turning to maintain course. If the course change is large enough, BigFoot will pivot about its inside wheels to turn sharply at the beginning of the turn, but will end up in a slower turn as the desired heading is approached. The inside voltage is limited to 2.35 volts to prevent driving the inside wheels in reverse. 2.35 allows a slight amount of forward motion of the wheels to ease the turn by preventing the dragging of the inside wheels.

5. Collision Avoidance Costatement

The final costatement of the autonomous loop is the collision avoidance section. The program calls the sonar function twice. The sonar function calls a series of I2C functions to direct the sonar to check for obstacles. All results of 0 are set to 255 due to the sonar reporting no contact as 0 instead of greater than 255. The sonar is checked twice to prevent false detections by averaging the two results. After the sonar is checked, the program checks each of the IR rangers. The program then compares all forward looking IR rangers and sonar ranges to determine if a collision threat is present. A threat is defined as a sonar contact at less than 75 inches or an IR contact at greater than 0.2 volts, which corresponds to a contact at any range. If a threat is present, BigFoot backs up for 1 second. The program then checks the side IR rangers to determine which direction is clear and orders BigFoot to turn to that direction. BigFoot turns for 1 second and then returns to autonomous operation.

V. OOPIC PROGRAM

The OOPic is programmed using visual basic. The OOPic program receives an input from the BL2000 and positions servos as ordered. The basic flow chart for the OOPic program is shown in Figure 43 and the full OOPic code is included Appendix B.

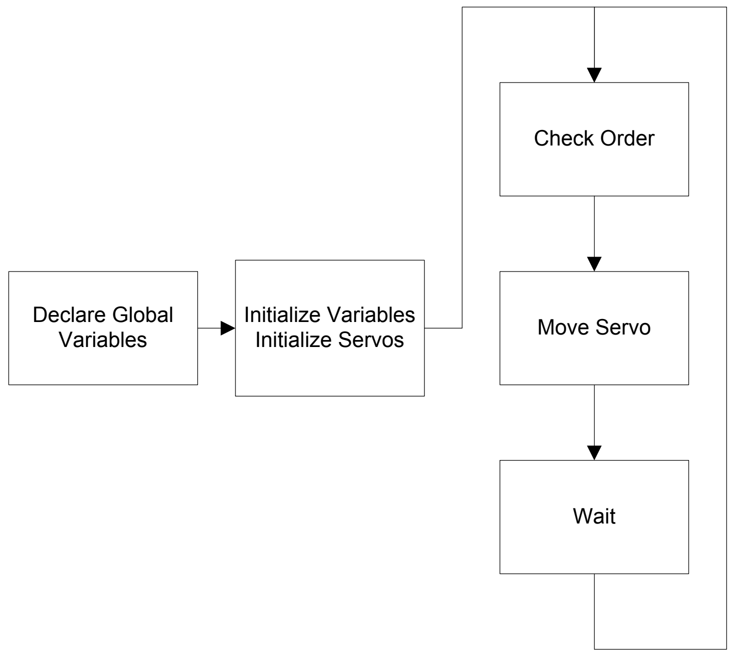


Figure 43. OOPic Program Loop.

The OOPic program starts by declaring global variables. These variables are initialized in the beginning of the main program. The initialization process declares which pins are to be used for the input and output as well as initial values. OOPic uses a variable type known as nibble. A nibble is a 4 bit word, literally a half of a byte. The servos have a center position defined and an initial value set. This will place the servo in the stowed position at the beginning of the program. The loop portion of the program is a series of logic questions. The program reads in the nibble input and determines which servo to position and whether to position it clockwise or counter clockwise. After repositioning the servo, the program waits for 250 milliseconds to slow the speed of operation.

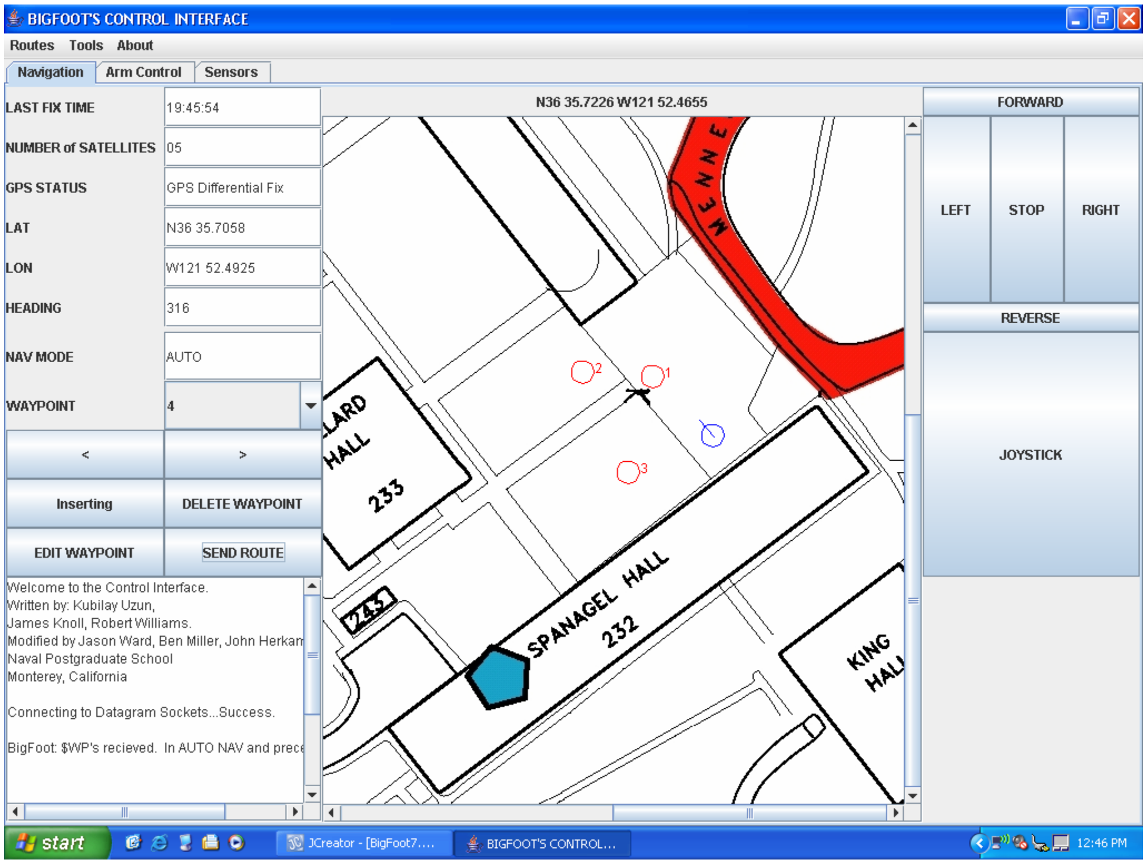
VI. JAVA GUI

BigFoot interfaces with the operator via a JAVA based Graphical User Interface (GUI) on a laptop computer. The GUI was originally developed at NPS by LT Kubilay Uzun (Turkish Air Force) [27]. This application has been rewritten and modified for ease of use and for application of additional sensors and control of BigFoot's arm. The full JAVA code is included in Appendix C.

The constructor class establishes the GUI window and all components. All components reside on one of three tabs or a menu bar at the top of the screen. The menu bar holds commands for infrequently used commands. The menu bar is divided into three menus: “Routes”, “Tools”, and “About”. The menus offer options to load or save routes, scale the waypoint map as well as to view the error log and program information. The three tabs house the remainder of GUI functionality. When a tab is selected a page is displayed that contains graphics, controls, and reports for various BigFoot functions.

A. MAP TAB

The map tab is the location for all navigation information. This page is broken into three major sections. The largest is the center section. The center section contains a predefined scalable map or figure. The map can be a digital image in any format, BMP, JPG, etc. The GUI has been tested using scale computer drawings as well as satellite imagery from “Google Earth”. When the robot is first used with a new map, the map must be scaled. This is accomplished by placing the robot in recognizable positions from the map and using the map scaling menu. Figure 44 shows the map tab with a scale drawing used as the navigation map.



LAST FIX TIME	19:45:54
NUMBER of SATELLITES	05
GPS STATUS	GPS Differential Fix
LAT	N36 35.7058
LON	W121 52.4925
HEADING	316
NAV MODE	AUTO
WAYPOINT	4

```
Welcome to the Control Interface.
Written by: Kubilay Uzun,
James Knoll, Robert Williams.
Modified by Jason Ward, Ben Miller, John Herkam
Naval Postgraduate School
Monterey, California

Connecting to Datagram Sockets...Success.

BigFoot: $WP's recieved.  In AUTO NAV and prec
```

Figure 44. GUI Map Tab.

The left side of the map page is a section for telemetry. Here, BigFoot's GPS position, heading, number of satellites tracked, and current time are displayed. Also in this location are the waypoint selection tools. These buttons allow the user to add or remove waypoints. When a route is constructed, the user presses the send route button and the route is sent to BigFoot. When BigFoot receives the route, it shifts to autonomous mode and navigates the waypoints as previously described. The lower left had side of the map page is an error pane. Any errors that BigFoot encounters are displayed in this area to notify the user. This is also where status messages such as manual mode, autonomous mode, waypoint reached, etc. are displayed.

The right side of the map page contains controls for manual control of BigFoot. The top buttons are for motion in the forward and reverse directions, pivoting, and stopping. These buttons send predetermined voltage values to

BigFoot to send to its motors. The forward and reverse buttons are set to send less than full speed voltages to allow more precise control. Below the manual control buttons is the Joystick panel. This joystick sends voltage values to BigFoot based on the mouse pointer position within the joystick panel. Due to BL2000 limitations, BigFoot only has a voltage range of 0 to 4 volts with 2.5 volts as the stop voltage. As a result, the scaling of the joystick is not uniform in the forward and reverse directions. This results in finer control in the forward direction, but non-linear response when turning with the joystick. Figure 45 shows BigFoot's joystick scaling.

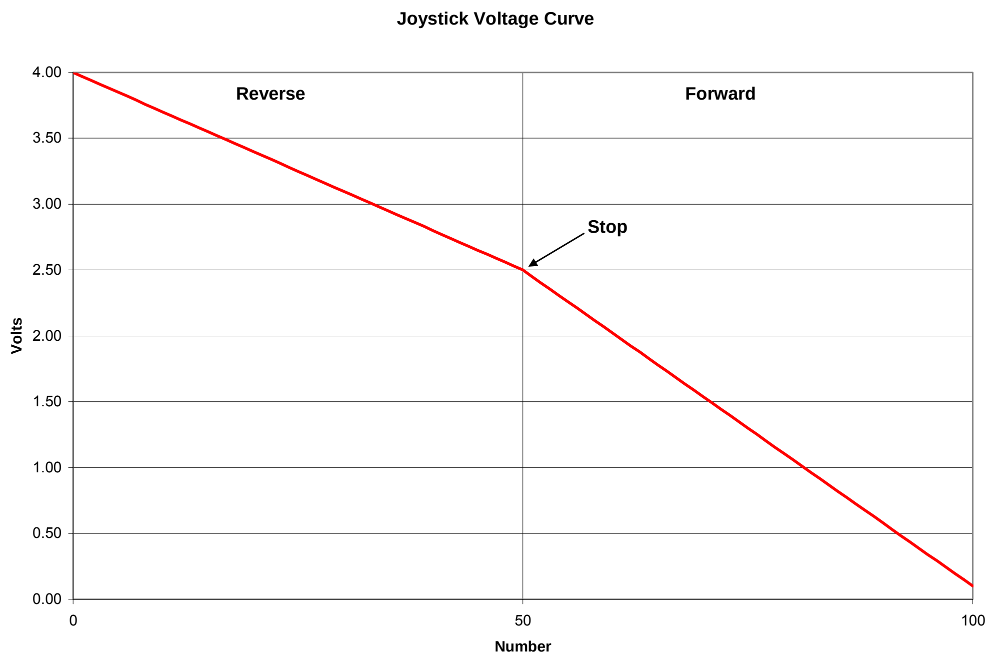


Figure 45. BigFoot's Joystick Scaling Graph.

B. ARM CONTROL TAB

The arm tab contains all controls necessary to position the arm. The page also provides dynamic position information for the operator to monitor arm position. The arm page is also divided into two sections. The largest section is on the left and contains overhead and side view drawings of BigFoot. This drawing is a dynamic graphic in that as the position of the arm is changed, the drawing reflects the ordered arm position with a scale drawing of the arm. This shows the location in all three dimensions of the arm's payload with respect to BigFoot. The picture indicates only the ordered arm position and not actual position as there are no physical arm position indicators. Figure 46 shows the GUI arm tab.

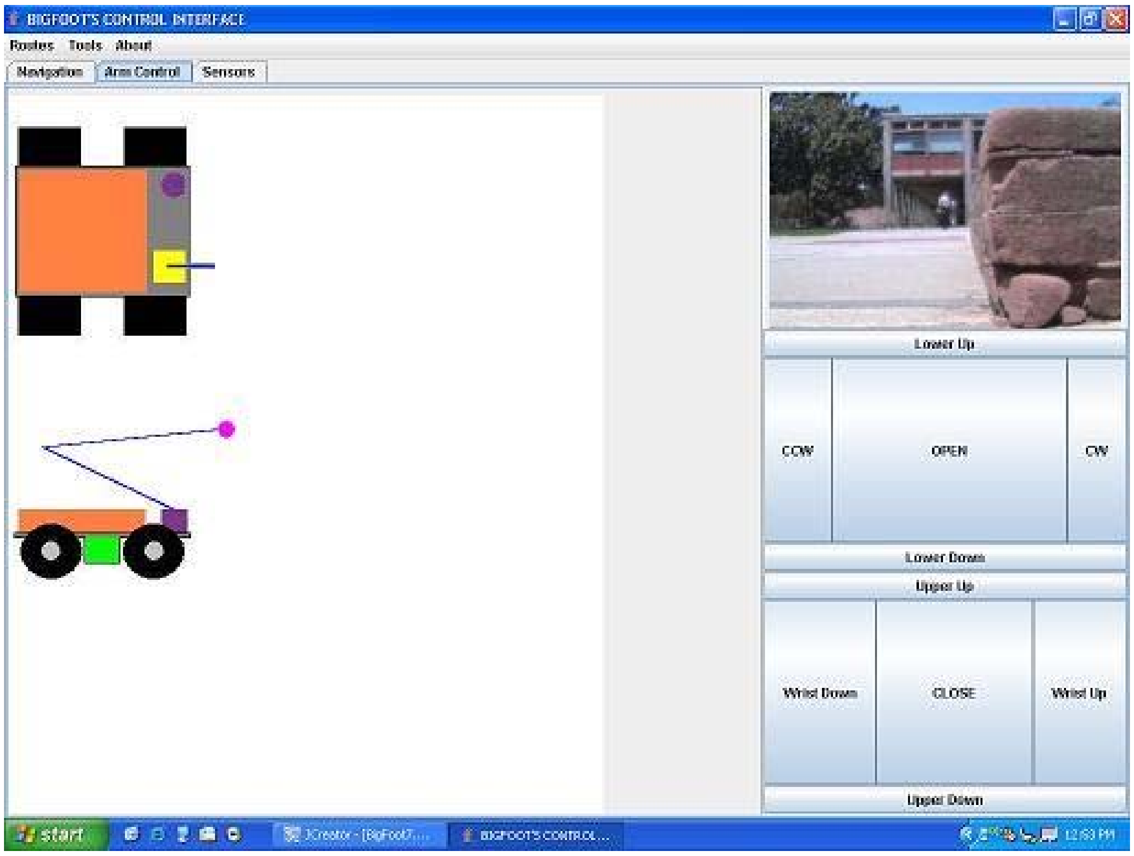


Figure 46. GUI Arm Tab.

The second section, on the right, contains a images from BigFoot's web camera as well as controls for the arm. The camera provides still images that are updated every time anything with in the program changes. This includes every time the GUI receives updated position and heading information from BigFoot. As a result, the camera does not provide streaming video, but it does provide images that are updated more frequently than once per second. Below the camera view are two button panels. These buttons control the servos for BigFoot's arm. When a button is clicked, the GUI sends a numerical command to operate a particular servo either clockwise or counter clockwise. The button click also causes the arm image to be redrawn at the new ordered position.

C. SENSOR TAB

The third tab contains BigFoot's sensor information. This page is also comprised of two major portions. The left and largest pane again contains a drawing of BigFoot. The picture is an overhead view. This picture is also a dynamic graphic. This drawing includes an arc that indicates the direction the camera and thermopile are pointing. When the camera is repositioned, the arc is redrawn to show the field of view with respect to BigFoot's position. This picture can also be used to indicate the direction of detected motion when used with a robot that employs motion detectors. Figure 47 shows the GUI sensor tab.

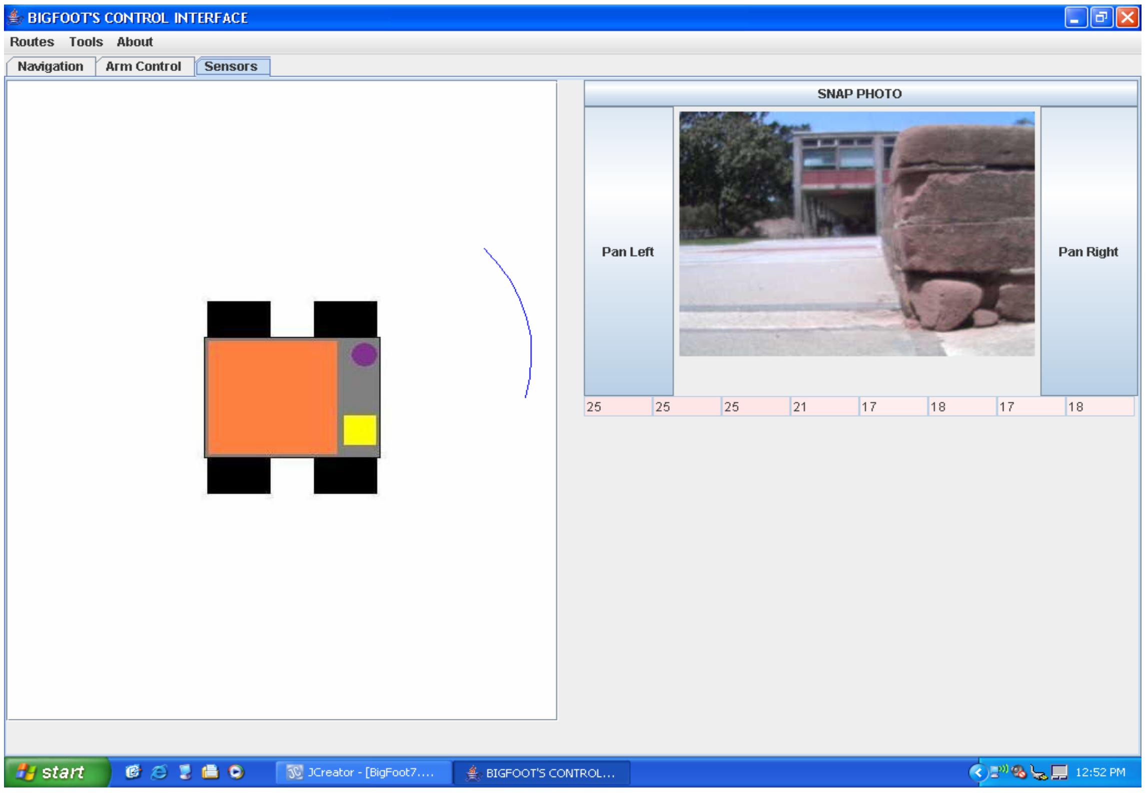


Figure 47. GUI Sensor Tab.

The second pane again contains an image feed from BigFoot's web camera. This pane also contains buttons to reposition the camera by pivoting clockwise or counter clockwise. These buttons are located on the side of the camera image. Directly above the image is a "Snap Photo" button. When this button is clicked, the GUI stores records an image in JPG format from the web camera. This image is saved with a unique name including a date-time stamp on the hard drive of the GUI laptop. This can be done as long as the interface laptop has space on its hard drive. Finally, directly below the camera image is an eight block display that approximately corresponds to the camera field of view. The eight blocks contain temperature information from the thermopile. This can be used in low light conditions to aid the operator in locating objects or even people. The display shows the numerical Centigrade temperature and is filled with a red scale color scheme in which white represents 0 degrees and red represents 255 degrees.

D. ADDITIONAL FUNCTIONALITY

The GUI now includes an error log and a GPS log. These logs record all data passed in the error and GPS streams, respectively. They are stored on the hard drive in text format. The data can be used to recreate the robot's track and troubleshoot in the unlikely event that errors are occurring.

VII. RESULTS

BigFoot's mechanical construction is sound. BigFoot is capable of taking a moderate shock. When run directly and indirectly into solid object, BigFoot is not damaged. The polystyrene bumper prevents any damage to front mounted sensors. The mounting of the battery makes BigFoot very stable and resistant to flipping over when climbing or descending obstacles. BigFoot has also been dropped from heights of greater than four feet with no noticeable effects. When driven off of a four foot high platform, the bottom heavy design and bumper allowed BigFoot to land on its wheels with only minor cosmetic damage. Secondly, BigFoot needs an enclosure. Currently, BigFoot is unable to operate in dusty, sandy, or wet environments due to risk of fouling the electronics.

The wheels work well on most surfaces. The wheels have a measured coefficient of friction of 0.96 on concrete, pavement, dirt, and grass. The coefficient of friction is measured by locking BigFoot's wheels and pulling with a spring-scale to measure the force necessary to start sliding. This coefficient of friction allows the robot to come to a dead stop from full speed with no sliding and only minor slipping when taking off from a dead stop at full speed. On loose surfaces, such as dirt, sand, or debris, BigFoot slides an average of 4 inches when coming to a full stop from full speed. When operating at less than full speeds, the skidding is minimal or non-existent.

BigFoot's arm is currently insufficient for the task of delivering an explosive charge. The servos do not produce enough torque to lift the hand efficiently. This is due to misalignment of the shoulder servos. By being slightly out of alignment, much of the torque produced by the servos is lost while the servos are fighting against each other. This can be alleviated by more precise construction of the shoulder joint components and precisely aligning the servos. Secondly, a single larger servo that could produce sufficient torque on its own would alleviate the torque loss. Finally, the hand is not complete. This hand can open and close with minimal gripping strength. The hand must either be rebuilt to include a more powerful grip or must be purchased off the shelf.

The electrical system has proven to be sufficient when modified for all purposes. The camera draws 1.5 amps at 5V. This alone is the capacity of the original 5V regulator in BigFoot’s power regulation system. The addition of a separate high capacity 5V regulator for the camera prevents excessive loading of the main 5V regulator. This permits all loads to be operated concurrently.

BigFoot’s batteries performed satisfactorily. The motor battery capacity, however, limits BigFoot’s range. The motor battery capacity gives an operational time limit of 2 hours while continuously running. The electronics battery capacity supports an operational time limit of approximately 11 hours. Together, they allow BigFoot the ability to perform multiple consecutive missions limited by the distance BigFoot must travel for each. If BigFoot is equipped with sensors for surveillance, on station time will be limited to approximately 11 hours. This is a dramatic increase over previous NPS robotic projects [8].

BigFoot’s compass performs satisfactorily in most cases. The compass is sensitive to stray magnetic fields which made mounting a difficult task. The motors produce large variable magnetic fields. The compass was originally placed by measuring the deviation from actual heading at 768 different points on BigFoot’s body. This provided a location that proved to be immune to stray steady state magnetic fields. However, further testing showed an effect on compass heading when the motors were stopped or started. This caused an electromagnetic pulse that would cause the compass to deviate from true heading by as much as 30 degrees. This was alleviated by moving the compass farther away from the motors. The compass is now mounted on a mast that elevates the compass beyond the effects of motor electromagnetic surges. Secondly, the BL2000 code has been modified to prevent the compass from taking a reading for 500 milliseconds after the motors start. This has eliminated all electromagnetic surge effects. BigFoot’s Devantech CMPS03 compass is also a better choice from that used by previous robotic projects [Refs. 7, 8]. The HMR3000 compass has pitch and roll measurement capability, but it requires 585 milliseconds to take a single reading. The CMPS03 requires 340 milliseconds, but does not have pitch and roll measurements available. For the NPS ground robotic projects, pitch and roll have not been employed and are not necessary at this point. This makes the CMPS03 more desirable due to reducing the program loop time by 245 milliseconds. This represents a 13% reduction from the total loop time of 1900 milliseconds.

The collision avoidance sensors effectively avoid most collisions. The sensors have been shown to be accurate to the manufacturer's design specifications [8]. The time duration of the program loop limits the frequency at which the sensors are sampled, however. Furthermore, sampling the sonar is a major contributor to the program loop time. The sonar requires 665 milliseconds from the initiation of the order to the time the measurement has been reported using the I2C protocol. The program calls this twice representing 70% of the total program loop time. As a result, the program loop time allows BigFoot to close on obstacles between samples. This effect has been minimized by calling all contacts detected by the IR rangers and any sonar contact less than 35 inches an obstacle. This provides sufficient range for BigFoot to detect an obstacle and stop if it is just outside this range on the previous sample. A second problem with the ultrasonic ranger arises when it reports an obstacle when none is present. This tends to be a problem as it occurs relatively frequently at an approximate frequency of 1 in 20 reports. This is not however a predictable frequency. The cause of these spurious detections is unknown. They are compensated for by ordering the sonar to check for obstacles twice in succession every loop through the program. The results are then averaged to mitigate effect of the spurious detection. The average is then used as the test sample to determine whether an obstacle is present or not.

The camera accomplishes all tasks that are demanded of it. The camera is mounted on a servo to enable panning left and right. However, the camera cannot view up and down. For BigFoot's purpose, this is not necessary as the camera can view what the arm is doing. However, the camera is only useful in well lit situations. As shown in Figure 48, BigFoot's camera performs poorly in darkness. The left pane shows an image in dim natural morning light. The middle pane shows the same scene with one light, and the right pane shows the same scene with normal interior lighting on. This is due to having no attached light to allow the camera to view the surroundings. To improve this, a camera with night vision, such as Swann's Max IP Cam network camera, is required. This would enable BigFoot to be useful in the dark and for surveillance. The camera's video is not accessible in the JAVA GUI. This is due to the camera's video interface requiring an internet browser. The streaming video is still available at the user laptop by merely opening an internet browser and typing in the camera's IP address. This requires another window to be open, but if streaming video is desired for surveillance, it is available.

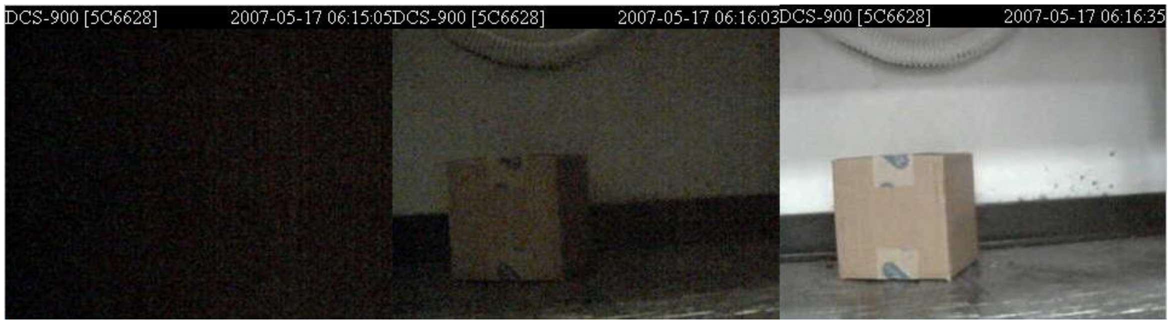


Figure 48. BigFoot's Camera in Multiple Lighting Scenarios.

BigFoot's GPS shows marked improvement over the GPS employed by previous NPS robotics projects. Previous units utilized the GPS 16 LVS. This unit updated once per second. The GPS 18-5Hz updates five times per second. Secondly, the previous projects included a time delay in the program loop for the GPS to ensure it was updated prior to plotting the robot's position. The GPS 18-

5Hz system update is slower than the time necessary to read the compass. This time delay is no longer required. This reduces the program by 500 milliseconds which is a reduction of 26% of the program loop time. The net reduction in program loop time by using the new compass and new GPS unit is 39%.

BigFoot's communication range is significantly improved over that of previous NPS robotics projects. The antenna increased range dramatically. The range increased from about 100 yards to almost 300 yards. Figure 49 shows the result of anechoic testing of the antenna to demonstrate its effectiveness [29].

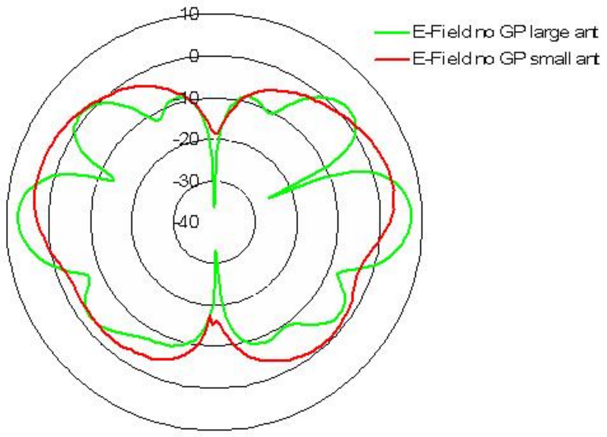


Figure 49. D-Link Antenna Anechoic Test Result (From [29]).

The PID controls for BigFoot required much testing. Previous PID controls were incomplete in their employment. After updating the code, the PID coefficients required tuning to set for optimal tuning. BigFoot was extensively tested to measure turning characteristics. These characteristics were used to predict optimal coefficients. The predicted coefficients have been implemented and shown to turn efficiently [26]. These coefficients are not optimal for all surfaces. In order to optimize turns on all surfaces, the robot would have to

detect its current surface characteristics and adjust the PID coefficients as necessary to adapt to the changing surfaces. This is not currently implemented as no method is available to passively detect the surface characteristics.

VIII. FUTURE WORK AND CONCLUSIONS

A. FUTURE WORK

The majority of BigFoot's remaining work will be in development of the arm. BigFoot's current arm is meant only as a proof of concept and a test platform for integrating OOPic and BL2000 functions. The arm will require more robust servos and a more complicated gripper hand. The hand requires a method to mix the chemicals of the explosive by puncturing a diaphragm. When the chemicals are mixed and the explosive deployed, a system must be added to remotely detonate the explosive. This can be done by either streaming a wire to the charge or, ideally, by a radio frequency signal. This must be completed in order to successfully meet BigFoot's design objectives.

Secondly, the communications need improvement. The D-Link antenna greatly boosted range, but ideal communications would be much longer range. The end state goal is to have the capability to communicate with a UAV to relay information to remote control stations. This would extend BigFoot's utility to a wide range of possible missions.

BigFoot also needs an improved camera. The current camera suffices for well lit operations, but a camera with night vision would be more effective and would allow operators to employ BigFoot in dark environments with no local modifications. This would also permit BigFoot to be used for surveillance operations with included motion detectors. This improvement is critical for BigFoot's usefulness in the field.

Finally, BigFoot requires a more robust body. Currently, all electronics are exposed to the environment. This will not suffice in a sandy or rainy environment. The case does not need to be particularly strong, but needs to be dust and splash proof. BigFoot does not have the capability to be thrown to deploy. BigFoot's ability to absorb shock must be improved to allow use in a battlefield environment.

B. CONCLUSIONS

BigFoot is capable of autonomously navigating to a user defined destination. Once it has arrived, it is capable of maneuvering its arm to deploy a shaped charge. BigFoot can deliver an explosive to the site of an IED and though it is not complete, BigFoot is near completion. Once the delivery mechanism is completed, BigFoot will be ready for mass production.

BigFoot cost approximately $3,000 to build from off the shelf components. This is significantly less than the $60,000 to purchase a TALON. BigFoot weighs 26lbs. This is also significantly less than the 140lbs of TALON with its arm installed. BigFoot is also significantly smaller in physical size than Talon. Together these make BigFoot an ideal addition to the military's arsenal for IED neutralization.

APPENDIX A – DYNAMIC C CODE

```
/* *******************************************************************
BigFoot.c

BigFoot.c controls a semiautonomous ground vehicle.

*******************************************************************
BL2000 CONNECTIONS

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

  GPS Serial Communications
  TX2           BROWN
  RX2           BROWN WITH RED
  GROUND        BLACK

  I2C connections
  SDA          OUT8/IN0  white
  SCL          OUT9/IN1  purple

  OOPic Connections
  OUT1, 2, 3, 4  arm control signals

  IR Rangers
  ADC3-ADC8

  I2C Components
  Compass        0XC0/1
  Sonar          0XE0/1
  Thermopile     0XD0/1
```

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

#define READDELAY 15

#define MAX_SENTENCE 100

//  Network Settings
#define MY_IP_ADDRESS              "192.168.1.91"
#define INTERFACE_ADDRESS      "192.168.1.90"
#define MY_NETMASK                      "255.255.255.0"
#define MY_GATEWAY                     "192.168.1.1"

#define MAN_PORT 4001       // receives manual control data
#define WP_PORT 4002        // receives waypoint data
#define GPS_PORT 4003       // sends gps data
#define ERROR_PORT 4005     // sends error reports, compass, thermopile data
#define ARM_PORT 4007       // receives arm orders

#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 = 3.14;

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;


//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


// Compass variables
char cmpd;                   // byte input from compass
int val;                   // numerical compass heading
char s[60];                 // compass sentence to error stream
char p[12];                 // compass data in ASCII


// Sonar variables
int sonarRange;              //
int sonarRange1;              //
int sonarRange2;              //
char SonarRaw;                // byte input from sonar
int SonRange;                //


// Communications
word status, port;
longword host;

udp_Socket gps_data, error_data;
```

```
sock_type wp_data, man_data, arm_data;

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

char wptBuf[4096];
char wptstr[500], *wptptr, *wpttmp;

char error_buf[200];

char armBuf[2048];
char armstr[20], *armptr;


//Nav Variables
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 BigFoot's current
                             //    position and the next waypoint)

float theta;          // Angle (deg) from True North to next waypoint
int hdg_error;        // Angle (deg) from current heading to next
                  //   waypoint
int new_hdg;          // The Desired heading in degrees
double rng;           // Range and temporary range (in yards)


//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;



// Arm Variables
int armCount;
int prevarmCount;
int armOrder;
int armFlag;
int serv1, serv2, serv3, serv4, serv5, serv6;



// Control Variables
const float PW_STOP = 2.50;   //Pulse width that results in stop command
const float PW_REV = 4.00;    //Pulse width that results in max reverse
const float PW_FWD = 0.10;

float LeftSide, RightSide;     // wheel control for manual control
const int rt_ch = 0;      //right side
const int lt_ch = 1;       //left side



// Collision Avoidance Variables
float fright, fleft, front, lside, rside, rear;
float ir6;        // right corner   anaIn(8)
float ir5;        // right center   anaIn(7)
float ir4;        // right side     anaIn(6)
float ir3;        // left corner    anaIn(5)
float ir2;        // left side      anaIn(4)
float ir1;        // left center    anaIn(3)



//Thermopile
char temp1;
char temp2;
char temp3;
char temp4;
char temp5;
char temp6;
char temp7;
char temp8;
char amb;
char t[70];

int t1i, t2i, t3i, t4i, t5i, t6i, t7i, t8i;
char t1a[8], t2a[8], t3a[8], t4a[8], t5a[8], t6a[8], t7a[8], t8a[8];

// Function Prototypes
int gps_get_position(GPSPosition *newpos, char *sentence);
int gps_parse_coordinate(char *coord, int *degrees, float *minutes);
int ERROR_function(int new_hdg);
```

```
void msDelay (long sd);
void compass(void);
int sonar(void);
void CommStart(void);
void goOut(void);
void write_byte(char);
int read_byte(char *ch);
void i2c_start_tx();
void i2c_stop_tx();
void i2c_init();
void getack();
void giveack();
void therm();

// I2C functions
#ifndef i2c_SCL_H()
#define i2c_SCL_H()   BitWrPortI(PEDR,&PEDRShadow,0,0)
#define i2c_SCL_L()    BitWrPortI(PEDR,&PEDRShadow,1,0)
#define i2c_SDA_H()   BitWrPortI(PEDR,&PEDRShadow,0,1)
#define i2c_SDA_L()    BitWrPortI(PEDR,&PEDRShadow,1,1)
#endif

int i2c_clocks_per_us;

#define cWAIT_5_us  asm ld a, (i2c_clocks_per_us) $\
             sub 3 $\
             ld b,a $\
            db 0x10, -2

unsigned long t0;
#define time 5


// ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
//
//                              Main Function
//
// ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

main()
{

         int i;

// --------------------------------------------------------------------------
//                                  Initializations
// --------------------------------------------------------------------------
    brdInit();
    i2c_init();
    CommStart();

     serv1=0;
     serv2=-15;
     serv3=14;
     serv4=0;
     serv5=0;



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

    anaOutVolts(rt_ch, PW_STOP);
    anaOutVolts(lt_ch, PW_STOP);

     new_hdg=0;
     compass();
     iScale=0;
    pScale=0;
    dScale=0;

        // set flags
                man_ctrl = 1;
                GPS_updated = 0;

        // 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)

  //new location for the following 6 lines, it works well here

//    P = 1;
//    I = 5;
//    D = 3;

  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(rt_ch, RightSide);
       anaOutVolts(lt_ch, 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




// ---------------------------------------------------------------------------
//                          Receive Arm Control Data
// ---------------------------------------------------------------------------



         costate
         {
     waitfor(sock_recv( &arm_data, armstr, (word)sizeof(armstr)));

       if(man_ctrl!=1)
       {
         abort;
```

```
   }

     anaOutVolts(rt_ch, PW_STOP);
     anaOutVolts(lt_ch, PW_STOP);

//Tokenize the string and convert to integers

          armCount = atoi(strtok(armstr, " "));
          armOrder = atoi(strtok(NULL, "/n"));


    if (armCount==prevarmCount)
    {
     abort;
    }
    else
    {

      armFlag=1;

      while(armFlag==1)

    {
      if (armOrder>12)
       {
         armOrder=0; // handle undefined orders
       }

      if (armFlag==1)
       {
         goOut();      // control the arm with the current order
         msDelay(300); // timedelay to prevent excessive operation speed
         armFlag=0;    // clear order
         armOrder=0;   // stop after one position change per button click
```

```
            goOut();      // send hold order
          } // end if
         }  // end while
       }  // end else

       prevarmCount = armCount;

      // end if man ctrl
   }// end arm control costate



// ------------------------------------------------------------------------
//                                  Thermopile Costatement
//
//  this is where we transmit the thermopile report to the GUI
// ------------------------------------------------------------------------

   costate
         {
    waitfor(man_ctrl);

       therm();
       // convert individual temperatures to ASCII
       itoa(t1i, t1a);
       itoa(t2i, t2a);
       itoa(t3i, t3a);
       itoa(t4i, t4a);
       itoa(t5i, t5a);
       itoa(t6i, t6a);
       itoa(t7i, t7a);
       itoa(t8i, t8a);

       // create temperature string
       strcpy(t, "$*,");   // $* denotes thermal information
```

```
        strcat(t, t1a);
        strcat(t, ",");
        strcat(t, t2a);
        strcat(t, ",");
        strcat(t, t3a);
        strcat(t, ",");
        strcat(t, t4a);
        strcat(t, ",");
        strcat(t, t5a);
        strcat(t, ",");
        strcat(t, t6a);
        strcat(t, ",");
        strcat(t, t7a);
        strcat(t, ",");
        strcat(t, t8a);

              sock_puts(&error_data, t);  // transmit temperature report
        tcp_tick(NULL);

         }// send thermopile data


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

   costate
         {
        compass();

        // create compass string
        itoa(val, p);
```

```
          strcpy(s, "$^,");     // $^ denotes compass data
          strcat(s, p);
          strcat(s, ",N,0,N,0");

                  sock_puts(&error_data, s);  // transmit compass report
          tcp_tick(NULL);

            }// send compass data



// ---------------------------------------------------------------------------
//                              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 ++;

                        // Stop Bigfoot and place in manual control if BAD position data
       //   for 6 times (1 minute) changed to 30 for new GPS (18-5Hz)
                                if ((gps_error_count > 30) && man_ctrl == 0)
                                {
                                        sock_puts(&error_data,
                "$GPS error count exceeded. Bigfoot in MANUAL CONTROL.\n");
                                        tcp_tick(NULL);

         anaOutVolts(rt_ch, PW_STOP);
                                        anaOutVolts(lt_ch, PW_STOP);

                                        //update flags for manual control
                                        man_ctrl = 1;
                                        abort;     //still parse if -1
                                }
                 }

// following three lines commented out due to not required
//                      if (1)//   (gps_error == 0)|| (gps_error == -1))
//                      {
                                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
        {

         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

         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
  {
     //add something here to turn in a circle
     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);      // Stops Bigfoot and places
          man_ctrl = 1;                     // in manual control
          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);          // Stops Bigfoot and places
        man_ctrl = 1;                        //in manual control

        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


// ------------------------------------------------------------------------------
//                                Collision Avoidance
// ------------------------------------------------------------------------------


  costate
   {
     waitfor(!man_ctrl);
```

```
//Sonar Itself

sonarRange1=sonar();
sonarRange2=sonar();

   if (sonarRange1 == 0)
         {
         sonarRange1 = 255;
         }  // take care of the sonar equal to zero when there is no return
   if (sonarRange2 == 0)
         {
         sonarRange2 = 255;
         }  // take care of the sonar equal to zero when there is no return

   sonarRange=((sonarRange1+sonarRange2)/2);

//IR Rangers
    ir1 = anaInVolts(3);   // left side
    ir2 = anaInVolts(4);   // left corner
    ir3 = anaInVolts(5);   // left center
    ir4 = anaInVolts(6);   // right side
    ir5 = anaInVolts(7);   // right corner
    ir6 = anaInVolts(8);   // right center



    if ((ir2 > 0.2)||(ir1 > 0.2)||(ir5 > 0.2)||(ir4 > 0.2)||
       (sonarRange < 35.0))
      {
        anaOutVolts(lt_ch, PW_REV);
        anaOutVolts(rt_ch, PW_REV);
        msDelay(1000);
       if (ir1 < ir4)
        {
         anaOutVolts(lt_ch, PW_REV);
```

```
            anaOutVolts(rt_ch, PW_FWD);
           }
        else
              {
            anaOutVolts(rt_ch, PW_REV);
            anaOutVolts(lt_ch, PW_FWD);
           }
        msDelay(1000);
        anaOutVolts(lt_ch, PW_FWD);
        anaOutVolts(rt_ch, PW_FWD);
     }
  }   //Collision Avoidance Costatement

}//while(1)

}//main



// *********************************************************************
//                         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(int new_hdg)
{
   hdg_error = new_hdg - val;

   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); );
}


//*******************************************************************
//                          Compass Function
// *******************************************************************

void compass()
  {
  i2c_start_tx();
  write_byte(0xC0);
  getack();
  write_byte(0x01);
  getack();
  i2c_start_tx();
  write_byte(0xC1);
  getack();
  read_byte(&cmpd);
  val=(int)(1.4*cmpd);
  i2c_stop_tx();
  }


// *******************************************************************
//                          Sonar Function
// *******************************************************************

int sonar()
{
  unsigned long tSonar;
  i2c_start_tx();
  write_byte(0xE0);
  getack();
  write_byte(0x00);
  getack();
  write_byte(0x50);
  getack();
  cWAIT_5_us;
  i2c_stop_tx();
  for(tSonar=MS_TIMER;MS_TIMER<tSonar+65;);
  i2c_start_tx();
  write_byte(0xE0);
        getack();
  write_byte(0x03);
  getack();
  i2c_start_tx();
  write_byte(0xE1);
  getack();
  read_byte(&SonarRaw);
  SonRange=(int)(SonarRaw);
  i2c_stop_tx();
  return SonRange;
  }
```

```
// *******************************************************************
//                          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 incoming arm order port
  if (!udp_open(&arm_data, ARM_PORT, 0xffffffff, ARM_PORT, NULL))
        {
                sock_puts(&error_data, "$Unable to open ARM UDP session\n");
                exit(3);
        }
        sock_mode( &arm_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);
        }

  if (sock_recv_init( &arm_data, armBuf, (word)sizeof(armBuf)))
```

```
            {
                        sock_puts(&error_data, "$Could not enable ARM buffer.\n");
                        exit(3);
            }
}                  // end Comm Start


// *********************************************************************
//                            Arm Output
//
//  This puts a 4 bit digital output based on input from GUI.  These inputs
//   drive the input pins to the OOPIC
// *********************************************************************

void goOut()
{
   switch (armOrder)
   {
     case 0:
     {
       digOut(1,0);
       digOut(2,0);
       digOut(3,0);
       digOut(4,0);
       break;
     }
     case 2:
     {
       if(serv2==32)
       {
         serv1=serv1;
       }
       else
       {
```

```
    digOut(1,1);
    digOut(2,0);
    digOut(3,0);
    digOut(4,0);
    serv1++;
    break;
  }
}
case 1:
{
  if(serv1==-32)
  {
    serv1=serv1;
  }
  else
  {
  digOut(1,0);
  digOut(2,1);
  digOut(3,0);
  digOut(4,0);
  serv1--;
  break;
  }
}
case 3:
{
  if(serv2==32)
  {
    serv2=serv2;
  }
  else
  {
  digOut(1,1);
  digOut(2,1);
  digOut(3,0);
  digOut(4,0);
  serv2++;
  break;
  }
}
case 4:
{
  if(serv2==-32)
  {
    serv2=serv2;
  }
  else
  {
  digOut(1,0);
  digOut(2,0);
  digOut(3,1);
  digOut(4,0);
  serv2--;
  break;
  }
}
case 5:
{
  if(serv3==32)
  {
    serv3=serv3;
  }
  else
  {
  digOut(1,1);
  digOut(2,0);
  digOut(3,1);
  digOut(4,0);
  serv3++;
  break;
  }
}
case 6:
{
  if(serv3==-32)
  {
    serv3=serv3;
  }
  else
  {
  digOut(1,0);
  digOut(2,1);
  digOut(3,1);
  digOut(4,0);
  serv3--;
  break;
  }
}
case 7:
{
  if(serv4==32)
  {
    serv4=serv4;
  }
  else
  {
  digOut(1,1);
  digOut(2,1);
  digOut(3,1);
  digOut(4,0);
  serv4++;
  break;
  }
}
case 8:
{
  if(serv4==-32)
  {
    serv4=serv4;
  }
  else
  {
  digOut(1,0);
  digOut(2,0);
  digOut(3,0);
  digOut(4,1);
  serv4--;
  break;
  }
}
case 9:
{
  if(serv5==32)
  {
    serv5=serv5;
  }
  else
  {
  digOut(1,1);
  digOut(2,0);
  digOut(3,0);
  digOut(4,1);
  serv5++;
  break;
  }
}
```

```
case 10:
{
 if(serv5==-32)
 {
  serv5=serv5;
 }
 else
 {
 digOut(1,0);
 digOut(2,1);
 digOut(3,0);
 digOut(4,1);
 serv5--;
 break;
 }
}
case 11:
{
 if(serv6==32)
 {
  serv6=serv6;
 }
 else
 {
 digOut(1,1);
 digOut(2,1);
 digOut(3,0);
 digOut(4,1);
 serv6++;
 break;
 }
}
case 12:
{
      if(serv6==-32)
      {
        serv6=serv6;
      }
      else
      {
      digOut(1,0);
      digOut(2,0);
      digOut(3,1);
      digOut(4,1);
      serv6--;
      break;
      }
    }
  } // end switch
} // end goOut


void write_byte(char d)    //done
{
  int i;
  for (i=0; i<8; i++)
  {
    for (t0=MS_TIMER;MS_TIMER<t0+time;);
                    if (d & 0x80)
                            {i2c_SDA_H();}
                    else
                            {i2c_SDA_L();}
  i2c_SCL_H();
  for (t0=MS_TIMER;MS_TIMER<t0+time;);
  i2c_SCL_L();
  d=d<<1;
  }
  i2c_SCL_L();
  i2c_SDA_H();
}


int read_byte(char *ch) // done
{
  auto char res,cnt;
  i2c_SDA_H();
  for (cnt=0,res=0; cnt<8; cnt++)
        {
                i2c_SCL_H();
    while (BitRdPortI(PEDR,2)==0);//SCL Clock Stretching
    for (t0=MS_TIMER;MS_TIMER<t0+time;);
    res<<=1;
                if(BitRdPortI(PEDR,3)) res|=0x01;
                i2c_SCL_L();
    for (t0=MS_TIMER;MS_TIMER<t0+time;);
        }
        *ch=res;
  return 0;
}


void i2c_start_tx() // done
{

        i2c_SCL_H();
        i2c_SDA_H();
  cWAIT_5_us;
  i2c_SDA_L();
  cWAIT_5_us;
        i2c_SCL_L();

}
```

```
void i2c_stop_tx()
{
        i2c_SDA_L();
  for(t0=MS_TIMER;MS_TIMER<t0+time;);
  //cWAIT_5_us;
        i2c_SCL_H();
  cWAIT_5_us;
        i2c_SDA_H();
}


void i2c_init()
{
        int i;
        void i2c_stop_tx();
        i2c_SDA_H();
        cWAIT_5_us;
        i2c_SCL_L();
        for (i=0; i < 3; i++)
        {
                i2c_stop_tx();
        }
}


void giveack()
{
        i2c_SDA_L();
  cWAIT_5_us;
        i2c_SCL_H();
        for(t0=MS_TIMER;MS_TIMER<t0+200;);
  //cWAIT_5_us;
```

```
            i2c_SCL_L();
   cWAIT_5_us;
            i2c_SDA_H();
}


void getack()
{
            i2c_SDA_H();
            while (BitRdPortI(PEDR,3) == 1);
 if (BitRdPortI(PEDR,3) == 1)i2c_stop_tx();      //originally uncommented
            i2c_SCL_H();
   for (t0=MS_TIMER;MS_TIMER<t0+time;);
   //cWAIT_5_us;
            i2c_SCL_L();
}


void therm()
{
   i2c_start_tx();
   write_byte(0xD0);
   getack();
   write_byte(0x00);
   getack();
   i2c_start_tx();
   write_byte(0xD1);
   getack();
   read_byte(&amb);

   i2c_start_tx();
   write_byte(0xD0);
   getack();
   write_byte(0x01);
```

```
getack();
i2c_start_tx();
write_byte(0xD1);
getack();
read_byte(&temp1);
t1i=(int)(temp1);

i2c_start_tx();
write_byte(0xD0);
getack();
write_byte(0x02);
getack();
i2c_start_tx();
write_byte(0xD1);
getack();
read_byte(&temp2);
t2i=(int)(temp2);

i2c_start_tx();
write_byte(0xD0);
getack();
write_byte(0x03);
getack();
i2c_start_tx();
write_byte(0xD1);
getack();
read_byte(&temp3);
t3i=(int)(temp3);

i2c_start_tx();
write_byte(0xD0);
getack();
write_byte(0x04);
getack();
```

```
i2c_start_tx();
write_byte(0xD1);
getack();
read_byte(&temp4);
t4i=(int)(temp4);

i2c_start_tx();
write_byte(0xD0);
getack();
write_byte(0x05);
getack();
i2c_start_tx();
write_byte(0xD1);
getack();
read_byte(&temp5);
t5i=(int)(temp5);

i2c_start_tx();
write_byte(0xD0);
getack();
write_byte(0x06);
getack();
i2c_start_tx();
write_byte(0xD1);
getack();
read_byte(&temp6);
t6i=(int)(temp6);

i2c_start_tx();
write_byte(0xD0);
getack();
write_byte(0x07);
getack();
i2c_start_tx();
```

```
  write_byte(0xD1);
  getack();
  read_byte(&temp7);
  t7i=(int)(temp7);

  i2c_start_tx();
  write_byte(0xD0);
  getack();
  write_byte(0x08);
  getack();
  i2c_start_tx();
  write_byte(0xD1);
  getack();
  read_byte(&temp8);
  t8i=(int)(temp8);

  i2c_stop_tx;
}// end therm
```

APPENDIX B – OOPIC CODE

```
' 5sevos.osc
' This program controls BigFoot's servos for the arm and camera

Dim count1 As New oByte
Dim count2 As New oByte
Dim count3 As New oByte
Dim count4 As New oByte
Dim count5 As New oByte

Dim serv1 As New oServo
Dim serv2 As New oServo
Dim serv3 As New oServo
Dim serv4 As New oServo
Dim serv5 As New oServo

Dim MyNib As New oDIO4


Sub main()

'declare input line numbers using pins 28-31
' board pins 26,28,30,32
   MyNib.IOGroup=3
   MyNib.Nibble=1
   MyNib.Direction=1

' inititialize counters
   count1.Value=31
   count2.Value=31
   count3.Value=31
   count4.Value=31
   count5.Value=31

' declare servos lines
   serv1.IOLine=9  ' pin 18
   serv2.IOLine=10 ' pin 16
   serv3.IOLine=11 ' pin 14
   serv4.IOLine=14 ' pin 8
   serv5.IOLine=15 ' pin 6

' order operate
   serv1.Operate=cvTrue
   serv2.Operate=cvTrue
   serv3.Operate=cvTrue
   serv4.Operate=cvTrue
   serv5.Operate=cvTrue

' set servo center values
   serv1.Center = 25          ' shoulder horizontal
   serv2.Center = 10          ' shoulder vertical
   serv3.Center = 18   ' wrist
   serv4.Center = 18          ' camera
   serv5.Center = 31          ' grip

serv1.Value=31    ' centered
serv2.Value=10    ' arm back
serv3.Value=1    ' wrist fully forward
serv4.Value=31    ' camera centered
serv5.Value=25    '
```

```
While (1)
Do

' determine if servo 1 is selected
If (MyNib.Value=1) Then
    count1.Value=count1.Value-1
    serv1.Value=count1.Value
    ooPIC.Delay = 25
End If

If (MyNib.Value=2) Then
    count1.Value=count1.Value+1
    serv1.Value=count1.Value
    ooPIC.Delay = 25
End If

' determine if servo 2 is selected
If (MyNib.Value=3) Then
    count2.Value=count2.Value-1
    serv2.Value=count2.Value
    ooPIC.Delay = 25
End If

If (MyNib.Value=4) Then
    count2.Value=count2.Value+1
    serv2.Value=count2.Value
    ooPIC.Delay = 25
End If

' determine if servo 3 is selected
If (MyNib.Value=5) Then
```

```
     count3.Value=count3.Value-1
     serv3.Value=count3.Value
     ooPIC.Delay = 25
End If

If (MyNib.Value=6) Then
     count3.Value=count3.Value+1
     serv3.Value=count3.Value
     ooPIC.Delay = 25
End If

' determine if servo 4 is selected
If (MyNib.Value=7) Then
     count4.Value=count4.Value-1
     serv4.Value=count4.Value
     ooPIC.Delay = 25
End If

If (MyNib.Value=8) Then
     count4.Value=count4.Value+1
     serv4.Value=count4.Value
     ooPIC.Delay = 25
End If

' determine if servo 5 is selected
If (MyNib.Value=9) Then
     count5.Value=count5.Value-1
     serv5.Value=count5.Value
     ooPIC.Delay = 25
End If

If (MyNib.Value=10) Then
```

```
    count5.Value=count5.Value+1
    serv5.Value=count5.Value
    ooPIC.Delay = 25
End If

Loop
Wend

End Sub
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C – JAVA CODE

```
//------------------------------------------------------------------
// Filename:   BigFoot.java
// Author:     Kubilay Uzun
// Rewritten:  John Herkamp
// Date:       5/16/2007
// Project:    Bigfoot
// Compiler:   JDK 1.4.1_06
//
// Original BigFoot GUI modified to include arm control functions, arm
// display functions, IP camera display and control, thermopile information
// error and gps logs.  The GUI is redesigned to establish page tabs and
// a menu bar.  Operates with BigFoot.c
//
//------------------------------------------------------------------

import javax.swing.*;
import javax.imageio.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.io.*;
import java.net.*;
import java.util.*;
import java.lang.*;
import java.lang.Integer;
import java.awt.image.BufferedImage;
import java.applet.*;
import java.applet.Applet;
import java.text.*;
import com.sun.image.codec.jpeg.*;
import javax.swing.event.*;


/*
 * The Bigfoot class provides all storage and services in order to monitor and
 * control the robot. The mission structure including waypoints and
 * actions are constructed and uploaded to the robot by this class. With a user
 * friendly GUI interface, Bigfoot helps the mission to be costructed on a
 * detailed map of the area. In addition, all robot sensor data can be monitored
 * continously via Bigfoot. Manual control of robot is accomplished by a
 * mouse-drag joystick which is very natural to adapt.
 * All network communications performed by Bigfoot use Datagram connections
 * which are fast and not sensitive to fading and connection losses which are
 * highly probable in a wireless network environment.
 *
 * @author Kubilay Uzun, modified John Herkamp
 */

public class BigFoot extends JFrame
            implements MouseMotionListener,
                    MouseListener,
                    WindowListener,
                    ActionListener {

/*
 * MOTOR_MIN,MOTOR_MAX constants set the precision of
 * manual control inputs. Selecting a small or big number is a trade-off between
 * precision and response speed. Since the control outputs are not sent unless
 * they're changed small numbers cause less control packets to be trasmitted.
 * However, Datagram connection used for this task is so fast that usually
 * there is no reason to worry about overloading buffer and degrading the
 * control response.
 */

   static final double MOTOR_MIN = 0;
   static final double MOTOR_MAX = 100;

   static final double MOTOR_MIN_VOLTS = 0.10;
   static final double MOTOR_MAX_VOLTS = 4.0;
   private double leftMotor = (MOTOR_MAX + MOTOR_MIN) / 2;
   private double rightMotor = (MOTOR_MAX + MOTOR_MIN) / 2;
   private double lOld, rOld;



/*
 * Map constants and globals. NAVMAP_FILE_NAME sets the file name
 * to be used as an area map. This should be a JPEG file. Huge files need
 * extended times to be loaded. Since the map is periodically updated, using
 * huge files may seriously degrade the performance of the program. Files
 * up to 1MByte are tolerated well depending on the performance of the
 * platform.
*/
   static final String NAVMAP_FILE_NAME = "NavMap_small.jpg";
   static final String ARM_PIC = "arm_pic.jpg";
   static final String SENSOR_PIC = "sensor_pic.jpg";
   static final String INTRO_PIC = "intro_pic.jpg";
   private JLabel map;
   private JLabel armmap;
   private JLabel sensormap;
   private JLabel intromap;
   private JLabel ArmCamMap;
   private JLabel SensorCamMap;
   private Graphics mapGraph;
```

```
    private Graphics armmapGraph;
    private Graphics sensormapGraph;
    private Image img;
    private Image armimg;
    private Image sensorimg;
    private Image introimg;
    private Image img1;
    private Image img2;
    private ImageIcon armmapIcon;
    private ImageIcon mapIcon;
    private ImageIcon sensormapIcon;
    private ImageIcon intromapIcon;
    private ImageIcon ArmCamMapIcon;
    private ImageIcon SensorCamMapIcon;
    private double WptCircleDia = 20; //WayPoint circle diameter

    static final String CAMERA_FILE_NAME = "http://192.168.1.2/image.jpg";
    static final String PROGRAM_LOCATION = "C:/j2sdk1.4.2_06/bin/";


// Global GUI items. These items are used by various methods of the program
// for I/O.

    private JButton joyStickPanel, insertWpt, stopButton, photoButton;
    private JPanel buttonControlPanel;
    private JTextArea messageArea;
    private JLabel cursorCoord;
    private JComboBox wayPointField;
    private JTextField fixTimeField,
                    numberOfSatellitesField,
                    gpsField,
                    latField,
                 lonField,
                 headField,
                 navModeField,
                 thermoText1,thermoText2,thermoText3,
                 thermoText4,thermoText5,thermoText6,
                 thermoText7,thermoText8;


// oldErrorMessage is used to keep track of the BigFoot origined error
// messages are being changed or not. Since the communication is
// asynchronous, this prevents the same message from being displayed
// repeatedly if there's nothing new.

  private String oldErrorMessage;



// Thermopile text data fields are used to display the thermopile
// temperature in centigrade.

  private String t1 = "0";
  private String t2 = "35";
  private String t3 = "70";
  private String t4 = "105";
  private String t5 = "140";
  private String t6 = "175";
  private String t7 = "210";
  private String t8 = "255";



// Networking constants and globals. Robot IP adress is defined here. This
// implies that a compiled intance of BigFoot belongs to a certain robot.
// In order to control other platforms ROBOT_IP_ADRESS should be modified.
```

```
    static final String ROBOT_IP_ADRESS = "192.168.1.91";


// Ports. Starting from 4001, ports are used for Manual control,
// route upload, arm control, gps monitor, and BigFoot error message
// monitoring respectively.

    static final int CMD_PORT = 4001;         //outgoing
    static final int ROUTE_PORT = 4002;       //outgoing
    static final int GPS_PORT = 4003;         //incoming
    static final int ERROR_PORT = 4005;       //incoming
    static final int ARM_ORDER_PORT = 4007;  //outgoing

    static final int CMD_BUFFER_SIZE = 2048;
    static final int ROUTE_BUFFER_SIZE = 2048;
    static final int ARM_BUFFER_SIZE = 2048;


// Datagram sockets. Note here that sockets are defined only for data output.
// The reason for this is data output is non-blocking. So, there no reason
// for using separate threads for Datagram output.

    private DatagramSocket cmdSock;
    private DatagramSocket routeSock;
    private DatagramSocket armSock;


// Datagram input threads. The blocking nature of asynchronous Datagram
// input pushes us to create separate threads for each input socket. See
// PacketReceivingThread for detailed explanation.
```

```
  private PacketReceivingThread gpsThread;
  private PacketReceivingThread errorThread;


// Timers. In this program javax.swing.Timer class is used. What is the
// reason for an initial delay? Since the timer is started immediately after
// timer.start() method is called, the tasks which are performed by the
// timer event may be calling not-yet-constructed objects. This probably
// causes the timer event to return a nullPointerException. Initial delay
// should ensure fully construction of the frame before operating.
// TMR_INTRO_DELAY is used for welcome screen

  static final int TMR_INITIAL_DELAY = 10000;
  static final int TMR1_DELAY = 10000;
  private javax.swing.Timer tmr1;

  static final int TMR_INTRO_DELAY = 2000;
  private javax.swing.Timer tmr2;


 // Waypoints. Maximum number of waypoints per mission is set by
 // MAX_WAYPOINTS constant. To change this, MAX_WAYPOINTS constant
 // should be modified.

  static final int MAX_WAYPOINTS = 10;
  static final int INSERT = 0;
  static final int OVERRIDE = 1;

  private int currentWpt;
  private int wptInsertMode;
  private int nowScaling;
  private double scalePos1Lat, scalePos1Lon, scalePos2Lat, scalePos2Lon;
```

```
   private double scalePos1x, scalePos1y, scalePos2x, scalePos2y;
   private double scaleX, scaleY;

   private WayPoint route[] = new WayPoint[MAX_WAYPOINTS];
   private WayPoint scalingWpt1;
   private WayPoint scalingWpt2;
   private WayPoint mapCursorWpt;

   private NewWayPoint nwpt;

  // Robot arm variables and constants.  x1, x2, x3, y1, y2, y3 are the coordinates
  // for the different joints on the arm map.  The arm lengths are scaled.
  //  Angles are stored in radian and degree units for specific calculations
  // Xref, Yref, Zref, X3ref are reference points for the bases of the joints
  // armCount is used to report which sequential order is being sent.

   private double x1, x2, x3, y1, y2, z3;
   private double L1=130, L2=30;  // the lengths of the arms
   private double Ang1, Ang2, Ang3, Ang5;
   private int Xref=150, Yref=375, X3ref, Zref=81;
   private int x1i, x2i, x3i, y1i, y2i, z3i; //integer form for line drawing
   private double Ang1Deg=0, Ang2Deg=0, Ang3Deg=0, Ang5Deg=0;
   private int serv; //servo control variable
   private int Ang1Temp=0, Ang2Temp=-30, Ang3Temp=0, Ang5Temp=0;
   private int armCount=0; // initial counter for arm


 // Camera variables.  This is for servo position information for the camera
   private double Ang4;
   private double Ang4Deg=0;
   private int Ang4Temp=0;
```

```
// Robot position. Current robot position and heading is retrieved from
// the Datagram socket, tokenized, parsed and assigned to these data members.
// Constant ROBOT_HEADING_TICK_LENGTH should be modified to change
// the heading tick.

  static final int ROBOT_HEADING_TICK_LENGTH = 20;
  private WayPoint robotPos;
  private double robotHeading;

  /**
   * File streams. Scaling data of the current mission map is read and written
   * each time program is entered or exited. Route streams provide load/save
   * functions of the current route.
   */
  private FileInputStream fStorIn;
  private FileOutputStream fStorOut;
  private DataInputStream storIn;
  private DataOutputStream storOut;
  private FileInputStream fRouteIn;
  private FileOutputStream fRouteOut;
  private ObjectInputStream routeIn;
  private ObjectOutputStream routeOut;
  private FileInputStream fLogOpen;
  private ObjectInputStream LogOpen;


  public JPanel ctrPanel, navPanel, pnl2;


// The BigFoot() constructor initializes data members of the main
// application and creates all GUI elements.  There are three tabs:
// one for navigation, one for arm control, and one for sensor display.
// A menu bar has been added to remove infrequently used tools from the
// main display page.

// @param none
// @exception none

   public BigFoot() {



      //Application's name
      super("BIGFOOT'S CONTROL INTERFACE");

      // Create launch window
      JWindow IntroWindow = new JWindow();
      IntroWindow.setLocation(320, 240);
            IntroWindow.setSize(400,303);
            intromap = new JLabel();
      JScrollPane IntroPicturePane = new JScrollPane(intromap);
      IntroWindow.add(IntroPicturePane, BorderLayout.WEST);
      intromap.setName("I");

      // Read launch image from file
      File file4 = new File(INTRO_PIC);
      try {
         introimg = ImageIO.read(file4);
      } catch (Exception ex) {
      // messageArea.append("\nCannot Read Picture Map.\n");
      }

      // Set image to launch window
      intromapIcon = new ImageIcon(introimg);
      intromap.setIcon(intromapIcon);
```

```
//launch window time delay
long time1;
for (time1=System.currentTimeMillis();
        System.currentTimeMillis()<time1+3000;)
{
IntroWindow.setVisible(true);
}

// Hide launch window
IntroWindow.setVisible(false);



// Main GUI window Size.
setSize(new Dimension(800,600));
setExtendedState(Frame.MAXIMIZED_BOTH);
setDefaultCloseOperation(EXIT_ON_CLOSE);

//Start javax.swing.Timer
tmr1 = new javax.swing.Timer(TMR1_DELAY, this);
tmr1.setInitialDelay(TMR_INITIAL_DELAY);
tmr1.start();

// Initialize current robot pos/heading to default
// default coordinates near outside of Halligan. TWD 29APR04
try {
robotPos = new WayPoint("N36 35.760 W121 52.600", "Turn");
robotHeading = 0;
} catch (Exception ex) {}

//Create an istance of NewWayPoint
nwpt = new NewWayPoint(this);
```

```
//Create menu bar buttons
JMenuItem j1 = new JMenuItem("Load Route");
j1.addActionListener(this);
JMenuItem j2 = new JMenuItem("Save Route");
j2.addActionListener(this);

JMenuItem j3 = new JMenuItem("Map Scaling");
j3.addActionListener(this);
JMenuItem j4 = new JMenuItem("View Error Log");
j4.addActionListener(this);

JMenuItem j5 = new JMenuItem("Acknowledgements");
j5.addActionListener(this);

//Create menu bar
JMenuBar menubar = new JMenuBar();

JMenu routemenu = new JMenu("Routes");
routemenu.add(j1);
routemenu.add(j2);

JMenu toolmenu = new JMenu("Tools");
toolmenu.add(j3);
toolmenu.add(j4);

JMenu aboutmenu = new JMenu("About");
aboutmenu.add(j5);

menubar.add(routemenu);
menubar.add(toolmenu);
menubar.add(aboutmenu);

// Establish Page Tabs

// Navigation Tab
JPanel navPanel = new JPanel(new BorderLayout());

// Arm Control Tab
JPanel armPanel = new JPanel(new BorderLayout());

// Sensor Tab
JPanel sensorPanel = new JPanel(new BorderLayout());

JTabbedPane tabs = new JTabbedPane();
tabs.addTab("Navigation", navPanel);
tabs.addTab("Arm Control", armPanel);
tabs.addTab("Sensors", sensorPanel);

// Main Panel
setJMenuBar(menubar);
setContentPane(tabs);
addWindowListener(this);

// Center Video Panel
JPanel ctrPanel = new JPanel(new BorderLayout());
navPanel.add(ctrPanel, BorderLayout.CENTER);

// Top Button Manifold
JPanel ctrUpperButtonPanel = new JPanel();
ctrPanel.add(ctrUpperButtonPanel, BorderLayout.NORTH);

// Cursor Position
```

```
cursorCoord = new JLabel("N0 0.0 W0 0.0");
ctrUpperButtonPanel.add(cursorCoord);

//Navigation map
map = new JLabel();

//Put this into a JScrollPane
JScrollPane mapPane = new JScrollPane(map);
ctrPanel.add(mapPane, BorderLayout.CENTER);

//Name is used to discriminate between GUI element calls
map.setName("M");

//Change cursor for accuracy of perception
map.setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
map.addMouseListener(this);
map.addMouseMotionListener(this);

//Read map from file
File file = new File(NAVMAP_FILE_NAME);
try {
  img = ImageIO.read(file);
} catch (Exception ex) {
  messageArea.append("\nCannot Read Navigation Map.\n");
}

mapIcon = new ImageIcon(img);
map.setIcon(mapIcon);

//Manual Control & Joystick Panels
JPanel pnl2 = new JPanel(new GridLayout(3,1));
navPanel.add(pnl2,BorderLayout.EAST);
```

```
JPanel buttonControlPanel = new JPanel(new BorderLayout());
pnl2.add(buttonControlPanel);

// Joystick
joyStickPanel = new JButton("JOYSTICK");
joyStickPanel.setName("J");
pnl2.add(joyStickPanel);
joyStickPanel.addMouseMotionListener(this);

// Manual Control Buttons
JButton stopButton = new JButton();
stopButton.setLabel("STOP");
stopButton.setVisible(true);
buttonControlPanel.add(stopButton, BorderLayout.CENTER);
stopButton.addActionListener(this);
JButton fwdButton = new JButton("FORWARD");
buttonControlPanel.add(fwdButton, BorderLayout.NORTH);
fwdButton.addActionListener(this);
JButton rvsButton = new JButton("REVERSE");
buttonControlPanel.add(rvsButton, BorderLayout.SOUTH);
rvsButton.addActionListener(this);
JButton rightButton = new JButton("RIGHT");
buttonControlPanel.add(rightButton, BorderLayout.EAST);
rightButton.addActionListener(this);
JButton leftButton = new JButton("LEFT");
buttonControlPanel.add(leftButton, BorderLayout.WEST);
leftButton.addActionListener(this);


//Position & Waypoint Panels
JPanel pnl3 = new JPanel(new GridLayout(3,1));
```

```
navPanel.add(pnl3,BorderLayout.WEST);

JPanel positionPanel = new JPanel(new GridLayout(6,2));
pnl3.add(positionPanel);
JLabel fixTimeLabel = new JLabel("LAST FIX TIME");
positionPanel.add(fixTimeLabel);
fixTimeField = new JTextField();
positionPanel.add(fixTimeField);
JLabel numberOfSatellitesLabel = new JLabel("NUMBER of SATELLITES");
positionPanel.add(numberOfSatellitesLabel);
numberOfSatellitesField = new JTextField();
positionPanel.add(numberOfSatellitesField);
JLabel gpsLabel = new JLabel("GPS STATUS");
positionPanel.add(gpsLabel);
gpsField = new JTextField();
positionPanel.add(gpsField);
JLabel latLabel = new JLabel("LAT");
positionPanel.add(latLabel);
latField = new JTextField();
positionPanel.add(latField);
JLabel lonLabel = new JLabel("LON");
positionPanel.add(lonLabel);
lonField = new JTextField();
positionPanel.add(lonField);
JLabel headLabel = new JLabel("HEADING");
positionPanel.add(headLabel);
headField = new JTextField();
positionPanel.add(headField);

//Waypoint Panel
JPanel wayPointPanel = new JPanel(new GridLayout(5,2));
pnl3.add(wayPointPanel);
```

```
JLabel navModeLabel = new JLabel("NAV MODE");
wayPointPanel.add(navModeLabel);
navModeField = new JTextField("MANUAL");
wayPointPanel.add(navModeField);
JLabel wayPointLabel = new JLabel("WAYPOINT");
wayPointPanel.add(wayPointLabel);
wayPointField = new JComboBox();
Integer ii;
for (int i = 1; i <= MAX_WAYPOINTS; i++) {
  ii = new Integer(i);
  wayPointField.addItem(ii.toString());
}
wayPointField.addActionListener(this);
wayPointPanel.add(wayPointField);
JButton wptDecrease = new JButton("<");
wayPointPanel.add(wptDecrease);
wptDecrease.addActionListener(this);
JButton wptIncrease = new JButton(">");
wayPointPanel.add(wptIncrease);
wptIncrease.addActionListener(this);
insertWpt = new JButton("Inserting");
wayPointPanel.add(insertWpt);
insertWpt.addActionListener(this);
JButton deleteWpt = new JButton("DELETE WAYPOINT");
wayPointPanel.add(deleteWpt);
deleteWpt.addActionListener(this);
JButton editWpt = new JButton("EDIT WAYPOINT");
wayPointPanel.add(editWpt);
editWpt.addActionListener(this);
JButton sendRoute = new JButton("SEND ROUTE");
wayPointPanel.add(sendRoute);
sendRoute.addActionListener(this);
```

```
//Message Output Area
messageArea = new JTextArea(20,22);
JScrollPane masp = new JScrollPane(messageArea);
pnl3.add(masp);

//Construct route
for (int i = 0; i < MAX_WAYPOINTS; i++) {
  route [i] = new WayPoint();
}


//   Arm Tab

//Center Picture Panel
JPanel ArmPicturePanel = new JPanel(new BorderLayout());
armPanel.add(ArmPicturePanel, BorderLayout.CENTER);

//Arm map
armmap = new JLabel();
JScrollPane ArmPicturePane = new JScrollPane(armmap);
ArmPicturePanel.add(ArmPicturePane, BorderLayout.NORTH);
armmap.setName("A");
armmap.setCursor(new Cursor(Cursor.HAND_CURSOR));
armmap.addMouseListener(this);
armmap.addMouseMotionListener(this);

// Read ARM map from file
File file1 = new File(ARM_PIC);
try {
  armimg = ImageIO.read(file1);
   }
```

```
catch (Exception ex) {
  // messageArea.append("\nCannot Read Navigation Map.\n");
   }
armmapIcon = new ImageIcon(armimg);
armmap.setIcon(armmapIcon);

// Arm Button and Camera Panels
JPanel armPanel2 = new JPanel(new GridLayout(3,1));
armPanel.add(armPanel2, BorderLayout.EAST);

// Camera Panel
ArmCamMap = new JLabel();
JPanel ArmCamPanel = new JPanel();
armPanel2.add(ArmCamPanel);
ArmCamPanel.add(ArmCamMap);
ArmCamMap.setName("AC");

//Read camera
URL url;
ArmCamMap.setIcon(null);
try {
   url = new URL(CAMERA_FILE_NAME);
   img2 = ImageIO.read(url);
  }
catch (MalformedURLException e){
    messageArea.append("\nBad URL\n");
  }
catch (java.io.IOException e){
    messageArea.append("\nBad File Reading\n");
  }
ArmCamMapIcon = new ImageIcon(img2);
ArmCamMap.setIcon(ArmCamMapIcon);
```

```
//Arm Control Button Panel 1
JPanel armControlPanel1 = new JPanel(new BorderLayout());
armPanel2.add(armControlPanel1);

JButton openButton = new JButton("OPEN");
armControlPanel1.add(openButton, BorderLayout.CENTER);
openButton.addActionListener(this);

JButton rotCWButton = new JButton("CW");
armControlPanel1.add(rotCWButton, BorderLayout.EAST);
rotCWButton.addActionListener(this);

JButton rotCCWButton = new JButton("CCW");
armControlPanel1.add(rotCCWButton, BorderLayout.WEST);
rotCCWButton.addActionListener(this);

JButton lowerUpButton = new JButton("Lower Up");
armControlPanel1.add(lowerUpButton, BorderLayout.NORTH);
lowerUpButton.addActionListener(this);

JButton lowerDownButton = new JButton("Lower Down");
armControlPanel1.add(lowerDownButton, BorderLayout.SOUTH);
lowerDownButton.addActionListener(this);

//Arm Panel 2
JPanel armControlPanel2 = new JPanel(new BorderLayout());
armPanel2.add(armControlPanel2);

JButton closeButton = new JButton("CLOSE");
armControlPanel2.add(closeButton, BorderLayout.CENTER);
closeButton.addActionListener(this);
```

```
JButton upperUpButton = new JButton("Wrist Up");
armControlPanel2.add(upperUpButton, BorderLayout.NORTH);
upperUpButton.addActionListener(this);

JButton upperDownButton = new JButton("Wrist Down");
armControlPanel2.add(upperDownButton, BorderLayout.SOUTH);
upperDownButton.addActionListener(this);


// Sensor Tab

//Center Picture Panel
JPanel SensorPicturePanel = new JPanel(new BorderLayout());
sensorPanel.add(SensorPicturePanel, BorderLayout.WEST);

//Sensor map
sensormap = new JLabel();

JScrollPane SensorPicturePane = new JScrollPane(sensormap);
SensorPicturePanel.add(SensorPicturePane, BorderLayout.NORTH);

sensormap.setName("S");

//Read map from file
File file2 = new File(SENSOR_PIC);
try {
  sensorimg = ImageIO.read(file2);
} catch (Exception ex) {
  messageArea.append("\nCannot Read Navigation Map.\n");
}
sensormapIcon = new ImageIcon(sensorimg);
```

```
sensormap.setIcon(sensormapIcon);

JPanel sensorPanel2 = new JPanel(new GridLayout(2,1));
sensorPanel.add(sensorPanel2, BorderLayout.EAST);

//Sensor Panel 1
JPanel sensorControlPanel1 = new JPanel(new BorderLayout());
sensorPanel2.add(sensorControlPanel1);

JButton camleftButton = new JButton("Pan Left");
sensorControlPanel1.add(camleftButton, BorderLayout.WEST);
camleftButton.addActionListener(this);

JButton camrightButton = new JButton("Pan Right");
sensorControlPanel1.add(camrightButton, BorderLayout.EAST);
camrightButton.addActionListener(this);

// Camera Panel
SensorCamMap = new JLabel();//map = new JLabel();
JPanel SensorCamPanel = new JPanel();//(ArmCamMap);
sensorControlPanel1.add(SensorCamPanel, BorderLayout.CENTER);
SensorCamPanel.add(SensorCamMap);
SensorCamMap.setName("SC");
SensorCamMap.setIcon(null);
SensorCamMapIcon = new ImageIcon(img2);
SensorCamMap.setIcon(SensorCamMapIcon);

//picture button
JButton photoButton = new JButton("SNAP PHOTO");
sensorControlPanel1.add(photoButton, BorderLayout.NORTH);
photoButton.addActionListener(this);
```

```
//Thermopile panel
JPanel thermoPanel = new JPanel(new GridLayout(1,8));

sensorControlPanel1.add(thermoPanel, BorderLayout.SOUTH);

// Set thermopile readout
thermoText1 = new JTextField(4);
thermoText2 = new JTextField(4);
thermoText3 = new JTextField(4);
thermoText4 = new JTextField(4);
thermoText5 = new JTextField(4);
thermoText6 = new JTextField(4);
thermoText7 = new JTextField(4);
thermoText8 = new JTextField(4);
thermoText1.setEditable(false);
thermoText2.setEditable(false);
thermoText3.setEditable(false);
thermoText4.setEditable(false);
thermoText5.setEditable(false);
thermoText6.setEditable(false);
thermoText7.setEditable(false);
thermoText8.setEditable(false);
thermoPanel.add(thermoText1);
thermoPanel.add(thermoText2);
thermoPanel.add(thermoText3);
thermoPanel.add(thermoText4);
thermoPanel.add(thermoText5);
thermoPanel.add(thermoText6);
thermoPanel.add(thermoText7);
thermoPanel.add(thermoText8);
thermoText1.setText(t8);
```

```
    thermoText2.setText(t7);
    thermoText3.setText(t6);
    thermoText4.setText(t5);
    thermoText5.setText(t4);
    thermoText6.setText(t3);
    thermoText7.setText(t2);
    thermoText8.setText(t1);

    //assign colors from white to red depending on the string number
    thermoText1.setBackground(new
         Color(255,255-Integer.decode(t8),255-Integer.decode(t8)));
    thermoText2.setBackground(new
         Color(255,255-Integer.decode(t7),255-Integer.decode(t7)));
    thermoText3.setBackground(new
         Color(255,255-Integer.decode(t6),255-Integer.decode(t6)));
    thermoText4.setBackground(new
         Color(255,255-Integer.decode(t5),255-Integer.decode(t5)));
    thermoText5.setBackground(new
         Color(255,255-Integer.decode(t4),255-Integer.decode(t4)));
    thermoText6.setBackground(new
         Color(255,255-Integer.decode(t3),255-Integer.decode(t3)));
    thermoText7.setBackground(new
         Color(255,255-Integer.decode(t2),255-Integer.decode(t2)));
    thermoText8.setBackground(new
         Color(255,255-Integer.decode(t1),255-Integer.decode(t1)));

    draw();

  } //Constructor


// Establish all Datagram connections/instantiate and start all packet
// receiving threads.

  public void establishDatagramConnections() {
    messageArea.append("\nConnecting to Datagram Sockets...");

    InetAddress hostAdress= null;
    try {
        //Instantiate sockets for data output
        cmdSock = new DatagramSocket();
        cmdSock.connect(hostAdress.getByName(ROBOT_IP_ADRESS),
              CMD_PORT);
        cmdSock.setSendBufferSize(CMD_BUFFER_SIZE);

        routeSock = new DatagramSocket();
        routeSock.connect(hostAdress.getByName(ROBOT_IP_ADRESS),
               ROUTE_PORT);
        routeSock.setSendBufferSize(ROUTE_BUFFER_SIZE);

        armSock = new DatagramSocket();
        armSock.connect(hostAdress.getByName(ROBOT_IP_ADRESS),
               ARM_ORDER_PORT);
        armSock.setSendBufferSize(ARM_BUFFER_SIZE);

        //Instantiate threads for data input
        gpsThread = new PacketReceivingThread(ROBOT_IP_ADRESS,
                                  GPS_PORT);
        errorThread = new PacketReceivingThread(ROBOT_IP_ADRESS,
                                   ERROR_PORT);
        //Start threads
        gpsThread.start();
        errorThread.start();
       } catch (Exception ex) {
        messageArea.append("Failed.\n");
        return;
    }
    messageArea.append("Success.\n");

  }//establishDatagramConnections



// Send control voltages to the control socket

  public void sendControlData() {
    double leftMotorVolts = 2.50;
    double RightMotorVolts = 2.50;

     //Calculate motor voltages
    if (leftMotor > 50)
    {
        leftMotorVolts = -0.048*leftMotor + 4.9;
    }
    else
    {
        leftMotorVolts = -0.03*leftMotor + 4.0;
    }

    if (rightMotor > 50)
    {
        RightMotorVolts = -0.048*rightMotor + 4.9;
    }
    else
    {
        RightMotorVolts = -0.03*rightMotor + 4.0;
    }
    Double lmv = new Double (leftMotorVolts);
    Double rmv = new Double (RightMotorVolts);
    Integer ml= new Integer((int)leftMotor);
    Integer mr= new Integer((int)rightMotor);
    Integer rmv1 = new Integer((int)leftMotorVolts);

    //Output motor values to control surface
    joyStickPanel.setText ("L" + ml.toString() + " R" + mr.toString());

    //Send control voltages
    InetAddress hostAdress= null;
    String lstr = lmv.toString() + "00000";
    String rstr = rmv.toString() + "00000";

    String cmdStr = lstr.substring(0,5) + " " +
                rstr.substring(0,5) + "\n\n\n\n";
    byte[] cmdBytes = cmdStr.getBytes();
    try {
        DatagramPacket cmdPack = new DatagramPacket(cmdBytes,
                        cmdBytes.length,
                        hostAdress.getByName(ROBOT_IP_ADRESS),
                        CMD_PORT);
        cmdSock.send(cmdPack);
        navModeField.setText("MANUAL");
      } catch (Exception ex) {
        messageArea.append("\nCannot Send Control Voltages.\n");
        return;
      }
  } //sendControlData
```

```
// sendArmData() added 11/13/06 - creates a command to move the arm.
// a counter is included to ensure robot moves only once per signal.
// also ensures old orders are ignored.

  public void sendArmData() {

    //Send control voltages
    armCount=armCount+1;

    Double aCount = new Double (armCount);
    Double aOrder = new Double (serv);

    InetAddress hostAdress= null;

    String CountStr = aCount.toString() + "00000";
    String OrderStr = aOrder.toString() + "00000";

    String armStr = CountStr.substring(0,5) + " " +
               OrderStr.substring(0,5) + "\n\n\n\n";

    byte[] armBytes = armStr.getBytes();
    try {
        DatagramPacket armPack = new DatagramPacket(armBytes,
                             armBytes.length,
                             hostAdress.getByName(ROBOT_IP_ADRESS),
                             ARM_ORDER_PORT);
        armSock.send(armPack);
      } catch (Exception ex) {
         messageArea.append("\nCannot Send Arm Control.\n");
         return;
      }
    serv=0;
```

```
   }//sendArmData


// Increase current waypoint indicator by one

   public void increaseCurrentWaypoint() {
      currentWpt++;
      if (currentWpt > (MAX_WAYPOINTS - 1)) {
        currentWpt = (MAX_WAYPOINTS - 1);
        }
      wayPointField.setSelectedIndex(currentWpt);
   }//increaseCurrentWaypoint


// Decrease current waypoint indicator by one

   public void decreaseCurrentWaypoint() {
      currentWpt--;
      if (currentWpt < 0) {
         currentWpt = 0;
      }
      wayPointField.setSelectedIndex(currentWpt);
   }//decreaseCurrentWaypoint


// When displaying navigation map, draw WayPoint symbols and robot symbol.
// When displaying arm map, draw arm position and camera image
// when displaying sensor map, draw camera position, camera image,
//  themopile information.

   public void draw() {
```

```
//Update map
map.setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));

File file = new File(NAVMAP_FILE_NAME);
try {
  img = ImageIO.read(file);
} catch (Exception ex) {
  messageArea.append("\nCannot Read Navigation Map.\n");
}

mapIcon = new ImageIcon(img);
map.setIcon(mapIcon);
mapGraph = img.getGraphics();
//Draw Waypoints
double lat, lon, screenX, screenY;
for (int i = 0; i < MAX_WAYPOINTS; i++) {
  lat = route[i].getLatNum();
  lon = route[i].getLonNum();

  screenY = scalePos1y + ((lat - scalePos1Lat) / scaleY);
  screenX = scalePos1x + ((lon - scalePos1Lon) / scaleX);

  mapGraph.setColor(Color.RED);
  mapGraph.drawOval((int)(screenX - WptCircleDia / 2 - 1),
                (int)(screenY - WptCircleDia / 2 - 1),
                (int)WptCircleDia,
                (int)WptCircleDia);
  Integer ii = new Integer(i + 1);
  mapGraph.drawString(ii.toString(),(int)(screenX + WptCircleDia / 2),
                 (int)screenY);
}
//Draw Robot Position & Heading
```

```
lat = robotPos.getLatNum();
lon = robotPos.getLonNum();

double x1,y1,x2,y2;
y1 = scalePos1y + ((lat - scalePos1Lat) / scaleY);
x1 = scalePos1x + ((lon - scalePos1Lon) / scaleX);

mapGraph.setColor(Color.BLUE);
mapGraph.drawOval((int)(x1 - WptCircleDia / 2 - 1),
              (int)(y1 - WptCircleDia / 2 - 1),
              (int)WptCircleDia,
              (int)WptCircleDia);
//robotHeading = robotHeading % 360;
double robotHeadingRad = (Math.toRadians(robotHeading)) -
                  ((Math.PI)/2);
y2 = y1 + ROBOT_HEADING_TICK_LENGTH *
             Math.sin(robotHeadingRad);
x2 = x1 + ROBOT_HEADING_TICK_LENGTH *
             Math.cos(robotHeadingRad);
mapGraph.drawLine((int)x1,(int)y1,(int)x2,(int)y2);

map.repaint();
img.flush();
mapGraph.dispose();

// Draw Arm Page
URL url;
ArmCamMap.setIcon(null);
try {
   url = new URL(CAMERA_FILE_NAME);
   img2 = ImageIO.read(url);
}
```

```
catch (MalformedURLException e){
   messageArea.append("\nBad URL\n");
}
catch (java.io.IOException e){
   messageArea.append("\nBad File Reading\n");
}

ArmCamMapIcon = new ImageIcon(img2);
ArmCamMap.setIcon(ArmCamMapIcon);

File file2 = new File(ARM_PIC);
try {
     armimg = ImageIO.read(file2);
    } catch (Exception ex) {
      messageArea.append("\nCannot Read Navigation Map.\n");
    }

armmapIcon = new ImageIcon(armimg);
armmap.setIcon(armmapIcon);
armmapGraph = armimg.getGraphics();
img2.flush();

// convert arm servo positions to degrees
Ang1Deg=2.8125*Ang1Temp;
Ang2Deg=2.8125*Ang2Temp;
Ang3Deg=2.8125*Ang3Temp;

// reference arm angles to one another
Ang2Deg=180-Ang2Deg;
Ang3Deg=-(Ang3Deg+90);

// convert arm angles to radians
```

```
Ang1=Ang1Deg*Math.PI/180;
Ang2=Ang2Deg*Math.PI/180;
Ang3=Ang3Deg*Math.PI/180;

//calculate positions
x1=L1*Math.sin(Ang2);
y1=L1*Math.cos(Ang2);

x2=L2*Math.sin(Ang3+Ang2);
y2=L2*Math.cos(Ang3+Ang2);

x3=(x2+x1)*Math.cos(Ang1);
z3=(x2+x1)*Math.sin(Ang1);

//find position and convert to int
x1i=Xref+(int)Math.round(x1);
x2i=x1i+(int)Math.round(x2);
x3i=Xref+(int)Math.round(x3);
y1i=Yref+(int)Math.round(y1);
y2i=y1i+(int)Math.round(y2);
z3i=Zref+(int)Math.round(z3);

//draw lines on side view
armmapGraph.setColor(Color.BLUE);
armmapGraph.drawLine(Xref-1,Yref, x1i-1, y1i);
armmapGraph.drawLine(Xref,Yref,x1i,y1i);
armmapGraph.drawLine(Xref+1,Yref,x1i+1,y1i);

armmapGraph.setColor(Color.BLUE);
armmapGraph.drawLine(x1i-1, y1i, x2i-1, y2i);
armmapGraph.drawLine(x1i,y1i,x2i, y2i);
armmapGraph.drawLine(x1i+1,y1i,x2i+1, y2i);
armmapGraph.setColor(Color.MAGENTA);
armmapGraph.fillOval(x2i, y2i-8, 16, 16);

//draw lines on top view
armmapGraph.setColor(Color.BLUE);
armmapGraph.drawLine(Xref, Zref-1, x3i, z3i-1);
armmapGraph.drawLine(Xref, Zref, x3i, z3i);
armmapGraph.drawLine(Xref, Zref+1, x3i, z3i+1);

armmap.repaint();
armimg.flush();
armmapGraph.dispose();

// Draw sensor page

//redraw camera on sensor page
SensorCamMap.setIcon(null);
SensorCamMapIcon = new ImageIcon(img2);
SensorCamMap.setIcon(SensorCamMapIcon);

//Read map from file
File file3 = new File(SENSOR_PIC);
try {
     sensorimg = ImageIO.read(file3);
   } catch (Exception ex) {
     messageArea.append("\nCannot Read Navigation Map.\n");
   }

sensormapIcon = new ImageIcon(sensorimg);
sensormap.setIcon(sensormapIcon);
```

```
        sensormapGraph = sensorimg.getGraphics();

        Ang4Deg=2.8125*Ang4Temp;

        int CamAng = (int)Ang4Deg-30;

        //draw camera view arc
        sensormapGraph.setColor(Color.BLUE);
        sensormapGraph.drawArc(172,119,300,300,CamAng,60);

        sensormap.repaint();
        sensorimg.flush();
        sensormapGraph.dispose();
    }//draw

// Get rid of leading, trailing and middle spaces

    public String cleanString(String s) {
      int spacePos;
      s = s.trim();
      while ((spacePos = s.indexOf(" ")) != -1 ) {
        s = s.substring(0, spacePos) +
            s.substring(spacePos + 1, s.length());
      }
      return s;
    }//cleanString

// Listen for and handle mouse clicking events on GUI elements. This happens
// when map is getting scaled or working with WayPoints. The method
// discriminates the call by caller's Name and do not react to other callers.

// @param e MouseEvent to be handled
```

```
public void mouseClicked(MouseEvent e) {
  //Left Double Click on Map
  if ((e.getComponent().getName()).equals("M")) {
    if ((e.getButton() == 1) && (e.getClickCount() == 2)) {
      //If WayPoint is being added
      if (nowScaling == 0) {
        WayPoint temp = new WayPoint();
        temp = nwpt.getWayPointData(this, mapCursorWpt);
        if (temp == null) {
          return;
        }
        if (wptInsertMode == INSERT) {
          for (int i = (MAX_WAYPOINTS - 1); i > currentWpt; i-- ) {
            route[i] = route[i - 1];
          }
        }
        route[currentWpt] =temp;
        draw();
        increaseCurrentWaypoint();
        return;
      }
      //If map scaling step 1 is happening
      if (nowScaling == 1) {
        scalingWpt1 = nwpt.getWayPointData(this, mapCursorWpt);
        if (scalingWpt1 == null) {
          return;
        }
        nowScaling =2;
        scalePos1Lat = scalingWpt1.getLatNum();
        scalePos1Lon = scalingWpt1.getLonNum();
            scalePos1x = e.getX();
            scalePos1y = e.getY();
            return;
          }
          //If map scaling step 2 is happening
          if (nowScaling == 2) {
            scalingWpt2 = nwpt.getWayPointData(this, mapCursorWpt);
            if (scalingWpt2 == null) {
              return;
            }
            nowScaling =0;
            messageArea.append("\nFollowing Points are Processed:\n");
            messageArea.append(scalingWpt1.getLatLon() + "\n");
            messageArea.append(scalingWpt2.getLatLon() + "\n");
            scalePos2Lat = scalingWpt2.getLatNum();
            scalePos2Lon = scalingWpt2.getLonNum();
            scalePos2x = e.getX();
            scalePos2y = e.getY();

            scaleX= (scalePos2Lon - scalePos1Lon) /
                   (scalePos2x - scalePos1x);
            scaleY= (scalePos2Lat - scalePos1Lat) /
                   (scalePos2y - scalePos1y);
         System.out.println("scalepos1x" + scalePos1x);
                 System.out.println("scalepos1y" + scalePos1y);
            return;
          }
        }
      }
    }//mouseClicked

  //Events not being implemented
    public void mousePressed(MouseEvent e) {
    }
    public void mouseReleased(MouseEvent e) {
    }
    public void mouseEntered(MouseEvent e) {
    }
    public void mouseExited(MouseEvent e) {
    }


  // Listen for and handle mouse dragging events on GUI elements. This happens
  // when manual control joystick is used. The method discriminates the call
  // by caller's Name and do not react to other callers.

  // @param e MouseEvent to be handled

    public void mouseDragged(MouseEvent e) {
      if ((e.getComponent().getName()).equals("J")) {
        Dimension rv = new Dimension();
        joyStickPanel.getSize(rv);
        double maxx = rv.getWidth() / 2;
        double maxy = rv.getHeight() / 2;



        // x and y values from joystick normalized to -1 to 1.
        double x = e.getX() / maxx - 1;
        double y = (maxy - e.getY()) / maxy;

        //Calculate motor values (not voltages)
        leftMotor =  ((y + x + 1) * ((MOTOR_MAX - MOTOR_MIN) / 2) +
                     MOTOR_MIN);
        rightMotor = ((y - x + 1) * ((MOTOR_MAX - MOTOR_MIN) / 2) +
                      MOTOR_MIN);
```

```
        //Check for overflows
        if (leftMotor > MOTOR_MAX) leftMotor = MOTOR_MAX;
        if (leftMotor < MOTOR_MIN) leftMotor = MOTOR_MIN;
        if (rightMotor > MOTOR_MAX) rightMotor = MOTOR_MAX;
        if (rightMotor < MOTOR_MIN) rightMotor = MOTOR_MIN;

        //Set current values not to send every time even if nothing's changed
        if ((leftMotor != lOld) || (rightMotor != rOld)) {
          sendControlData();
          lOld = leftMotor;
          rOld = rightMotor;
        }
      }
    }//mouseDragged


  // Listen for and handle mouse moving events on GUI elements. This happens
  // when cursor is moved over the map.This method computes and otputs the
  // current position. The method discriminates the call by caller's Name
  // and do not react to other callers.

  // @param e MouseEvent to be handled

    public void mouseMoved(MouseEvent e) {
      if ((e.getComponent().getName()).equals("M")) {
        double curX, curY;

        curX = scalePos1Lat + (((e.getY() - scalePos1y )) * scaleY);
        curY = scalePos1Lon + (((e.getX() - scalePos1x)) * scaleX);

        try {
```

```
        mapCursorWpt = new WayPoint(curX, curY, "Turn");
      } catch (Exception ex) {
        return;
      }
      cursorCoord.setText(mapCursorWpt.getLatLon());
    }
  }//mouseMoved


// Perform tasks which should be done immediately the program starts such as
// welcoming user, establishing connections and loading map scaling data.

// @param e WindowEvent to be handled

  public void windowOpened(WindowEvent e) {

    //Welcome user
    messageArea.append("Welcome to the Control Interface.\n");
    messageArea.append("Written by: Kubilay Uzun,\n");
    messageArea.append("James Knoll, Robert Williams.\n");
    messageArea.append
        ("Modified by Jason Ward, Ben Miller, John Herkamp\n");
    messageArea.append("Naval Postgraduate School\n");
    messageArea.append("Monterey, California\n");

    //Establish Connections
    establishDatagramConnections();

    //Read Scaling Data
    try {
      fStorIn = new FileInputStream("Storage.dat");
      storIn = new DataInputStream(fStorIn);
      scalePos1Lat = storIn.readDouble();
      scalePos1Lon = storIn.readDouble();
      scalePos1y = storIn.readDouble();
      scalePos1x = storIn.readDouble();
      scaleY = storIn.readDouble();
      scaleX = storIn.readDouble();
    } catch (Exception ex) {
      messageArea.append("\nCannot Read Map Scaling Data.\n");
    }
   }//windowOpened

// Perform tasks which should be done just before the program exits such as
// closing sockets, saving map scaling data.

// @param e WindowEvent to be handled

  public void windowClosing(WindowEvent e) {

    //Close sockets
    try {
      cmdSock.close();
      routeSock.close();
      armSock.close();
    } catch (Exception ex) {
      messageArea.append("\nCannot Close Sockets.\n");
    }
    //Write Scaling Data
    try {
      fStorOut = new FileOutputStream("Storage.dat");
      storOut = new DataOutputStream(fStorOut);
```

```
        storOut.writeDouble(scalePos1Lat);
        storOut.writeDouble(scalePos1Lon);
        storOut.writeDouble(scalePos1y);
        storOut.writeDouble(scalePos1x);
        storOut.writeDouble(scaleY);
        storOut.writeDouble(scaleX);
      } catch (Exception ex) {
        messageArea.append("\nCannot Write Map Scaling Data.\n");
      }
    }//windowClosing


//Events not being implemented
    public void windowClosed(WindowEvent e) {
    }
    public void windowIconified(WindowEvent e) {
    }
    public void windowDeiconified(WindowEvent e) {
    }
    public void windowActivated(WindowEvent e) {
    }
    public void windowDeactivated(WindowEvent e) {
    }



// Perform numerous tasks including all functional buttons and the timer

// @param ev ActionEvent to be handled

    public void actionPerformed(ActionEvent ev) {

       //Timer event happened
```

```
if (ev.getActionCommand() == null) {
   boolean anythingChanged = false;

   //Query packet receiving threads
   String gpsStream = gpsThread.getReceivedData();
   String errorStream = errorThread.getReceivedData();


   //If GPS socket have something
   if (gpsStream != null) {

   //Tokenize GPS data
   StringTokenizer tok = new StringTokenizer(gpsStream, ",");
   if (tok.countTokens() < 8) {
      messageArea.append("\nCorrupted GPS Data.\n");
      return;
      }
   tok.nextToken(); //ignore header
   String GPSTime = tok.nextToken();
   GPSTime = cleanString(GPSTime);
   String GPSLatNums = tok.nextToken();
   GPSLatNums = cleanString(GPSLatNums);
   String GPSLatHemi = tok.nextToken();
   GPSLatHemi = cleanString(GPSLatHemi);
   String GPSLonNums = tok.nextToken();
   GPSLonNums = cleanString(GPSLonNums);
   String GPSLonHemi = tok.nextToken();
   GPSLonHemi = cleanString(GPSLonHemi);
   String GPSStatus = tok.nextToken();
   GPSStatus = cleanString(GPSStatus);
   String GPSSatellites = tok.nextToken();
   GPSSatellites = cleanString(GPSSatellites);
```

```
GPSTime = GPSTime.substring(0,2) + ":" +
        GPSTime.substring(2,4) + ":" +
        GPSTime.substring(4,6);

String GPSPos = GPSLatHemi +
             GPSLatNums.substring(0,2) + " " +
             GPSLatNums.substring(2,9) + " " +
             GPSLonHemi +
             GPSLonNums.substring(0,3) + " " +
             GPSLonNums.substring(3,10);
if (GPSStatus.equals("0")) {
   GPSStatus = "GPS not Available";
   } else if (GPSStatus.equals("1")) {
   GPSStatus = "GPS Available";
   } else {
   GPSStatus = "GPS Differential Fix";
   }

 fixTimeField.setText(GPSTime);
 numberOfSatellitesField.setText(GPSSatellites);
 gpsField.setText(GPSStatus);
 try {
   robotPos = new WayPoint(GPSPos, "Turn");
    } catch (Exception ex) {
     messageArea.append("\n Cannot Parse Robot Position.\n");
    }
 latField.setText(robotPos.getLat());
 lonField.setText(robotPos.getLon());
 anythingChanged = true;

 try {
```

```
            FileWriter gpsl = new FileWriter("gps_log.txt", true);
            gpsl.write(gpsStream);
            gpsl.write("\n");
            gpsl.close();
           }
        catch (IOException e) {
        System.out.println("Error -- could not open gps log");
         }
    }


      //If error socket has something
      if (errorStream != null) {
         String temp = new String();
         StringTokenizer ErrorTok = new StringTokenizer(errorStream, ",");
         temp = cleanString(ErrorTok.nextToken());
         if (temp.equals("$^"))
                {
           temp = ErrorTok.nextToken();
                 headField.setText(cleanString(temp));
                try {
                        robotHeading = Double.parseDouble(headField.getText());
                }
           catch(Exception ex) {
                        messageArea.append
                             ("\nCannot Parse Compass Heading.\n");
                   }
                anythingChanged = true;
           }

         else if (temp.equals("$*"))
                {
```

```
t1 = cleanString(ErrorTok.nextToken());
t2 = cleanString(ErrorTok.nextToken());
t3 = cleanString(ErrorTok.nextToken());
t4 = cleanString(ErrorTok.nextToken());
t5 = cleanString(ErrorTok.nextToken());
t6 = cleanString(ErrorTok.nextToken());
t7 = cleanString(ErrorTok.nextToken());
t8 = cleanString(ErrorTok.nextToken());
thermoText1.setText(t1);
thermoText2.setText(t2);
thermoText3.setText(t3);
thermoText4.setText(t4);
thermoText5.setText(t5);
thermoText6.setText(t6);
thermoText7.setText(t7);
thermoText8.setText(t8);
thermoText1.setBackground(new Color
    (255,255-Integer.decode(t1),255-Integer.decode(t1)));
thermoText2.setBackground(new Color
    (255,255-Integer.decode(t2),255-Integer.decode(t2)));
thermoText3.setBackground(new Color
    (255,255-Integer.decode(t3),255-Integer.decode(t3)));
thermoText4.setBackground(new Color
    (255,255-Integer.decode(t4),255-Integer.decode(t4)));
thermoText5.setBackground(new Color
    (255,255-Integer.decode(t5),255-Integer.decode(t5)));
thermoText6.setBackground(new Color
    (255,255-Integer.decode(t6),255-Integer.decode(t6)));
thermoText7.setBackground(new Color
    (255,255-Integer.decode(t7),255-Integer.decode(t7)));
thermoText8.setBackground(new Color
    (255,255-Integer.decode(t8),255-Integer.decode(t8)));
```

```
              }

              else if(!(errorStream.equals(oldErrorMessage))) {
             if (ErrorTok.countTokens() < 3)
              {
                messageArea.append("\nBigFoot: " + errorStream + "\n" );
                    oldErrorMessage = errorStream;
              }
              }
              anythingChanged = true;
      }

            try {
              FileWriter erl = new FileWriter("error_log.txt", true);
              erl.write(errorStream);
              erl.write("\n");
              erl.close();
                }
            catch (IOException e) {
              System.out.println("Error -- could not open error log");
                }

    if (anythingChanged) {
      draw();
      }
    return;
  }

  //Motor control event Happened
  boolean isMotorEvent = false;
  if (ev.getActionCommand().equals("STOP")) {
    leftMotor = 50;
    rightMotor = 50;
    isMotorEvent = true;
  }else if (ev.getActionCommand().equals("FORWARD")) {
    leftMotor = 75; //MOTOR_MAX;
    rightMotor = 75; //MOTOR_MAX;
    isMotorEvent = true;
  }else if (ev.getActionCommand().equals("REVERSE")) {
    leftMotor = 25; //MOTOR_MIN;
    rightMotor = 25; //MOTOR_MIN;
    isMotorEvent = true;
  }else if (ev.getActionCommand().equals("RIGHT")) {
    leftMotor = MOTOR_MAX;
    rightMotor = MOTOR_MIN;
    isMotorEvent = true;
  }else if (ev.getActionCommand().equals("LEFT")) {
    leftMotor = MOTOR_MIN;
    rightMotor = MOTOR_MAX;
    isMotorEvent = true;
  }
  if (isMotorEvent) {
    if (((leftMotor != lOld) || (rightMotor != rOld)) ||
      (ev.getActionCommand().equals("STOP"))) {
      sendControlData();
      lOld = leftMotor;
      rOld = rightMotor;
    }
    return;
  }

  //Map Scaling event happened
  if (ev.getActionCommand().equals("Map Scaling")) {
    nowScaling = 1;
    JOptionPane.showMessageDialog(this, "Double Click and Enter 2 " +
                                "Points on the map.");
    return;
  }

  //Increasing and decreasing current wpt event happened
  if (ev.getActionCommand().equals(">")) {
    increaseCurrentWaypoint();
    return;
  }
  if (ev.getActionCommand().equals("<")) {
    decreaseCurrentWaypoint();
    return;
  }
  if (ev.getActionCommand().equals("comboBoxChanged")) {
    currentWpt = wayPointField.getSelectedIndex();
    return;
  }

  //Delete WayPoint event happened
  if (ev.getActionCommand().equals("DELETE WAYPOINT")) {
    int pos, max;
    for (pos = currentWpt; pos < (MAX_WAYPOINTS - 1); pos++) {
      route[pos] = route[pos + 1];
    }
    route[MAX_WAYPOINTS - 1] = new WayPoint();
    draw();
    return;
  }

  //Insert/override event happened
  if (ev.getActionCommand().equals("Inserting")) {
    insertWpt.setText("Overriding");
    wptInsertMode = OVERRIDE;
    return;
  }
  if (ev.getActionCommand().equals("Overriding")) {
    insertWpt.setText("Inserting");
    wptInsertMode = INSERT;
    return;
  }

  //Edit WayPoint event happened
  if (ev.getActionCommand().equals("EDIT WAYPOINT")) {
    WayPoint temp = new WayPoint();
    temp = nwpt.getWayPointData(this, route[currentWpt]);
    if (temp == null) {
      return;
    }
    route[currentWpt] = temp;
    draw();
  }

  //Save/Load route event happened
  if (ev.getActionCommand().equals("Save Route")) {
    FileDialog sr = new FileDialog(this, "Save Route", FileDialog.SAVE);
    sr.show();
    String fn = sr.getFile();
    try {
      fRouteOut = new FileOutputStream(fn);
      routeOut = new ObjectOutputStream(fRouteOut);
      for (int i=0; i < MAX_WAYPOINTS; i++) {
        routeOut.writeObject(route[i]);
      }
```

```
        routeOut.close();
      } catch (Exception ex) {
        messageArea.append("\nCannot Save Route.\n");
      }
      draw();
      return;
    }
    if (ev.getActionCommand().equals("Load Route")) {
      FileDialog sr = new FileDialog(this, "Load Route", FileDialog.LOAD);
      sr.show();
      String fn = sr.getFile();
      try {
        fRouteIn = new FileInputStream(fn);
        routeIn = new ObjectInputStream(fRouteIn);
        for (int i=0; i < MAX_WAYPOINTS; i++) {
          route[i] = (WayPoint)routeIn.readObject();
        }
        routeIn.close();
      } catch (Exception ex) {
        messageArea.append("\nCannot Load Route.\n");
      }
      draw();
      return;
    }

    //Send route event happened
    if (ev.getActionCommand().equals("SEND ROUTE")) {
      String routeStream = "$" + route[0].getPack() + " ";

      for (int i =1; i < MAX_WAYPOINTS; i++ ) {
        routeStream = routeStream + route[i].getPack()+ " ";
      }
      routeStream = routeStream + "\n\n\n\n\n";

      InetAddress hostAdress= null;
      //send route
      byte[] routeBytes = routeStream.getBytes();
      try {
        DatagramPacket routePack = new DatagramPacket(routeBytes,
                             routeBytes.length,
                             hostAdress.getByName(ROBOT_IP_ADRESS),
                             ROUTE_PORT);
        routeSock.send(routePack);
        navModeField.setText("AUTO");
      } catch (Exception ex) {
        messageArea.append("\nCannot Send Route.\n");
        return;
      }
    }

    // servo control events
    if (ev.getActionCommand().equals("CW")) {
      //ArmCamMap.repaint();
      serv = 1;
      sendArmData();
      Ang1Temp++;
      draw();
      return;
    }

    if (ev.getActionCommand().equals("CCW")) {
      serv = 2;
      sendArmData();
      Ang1Temp--;
      draw();
      return;
    }

    if (ev.getActionCommand().equals("Lower Up")) {
      serv = 4;
      sendArmData();
      Ang2Temp++;
      draw();
      return;
    }
    if (ev.getActionCommand().equals("Lower Down")) {
      serv = 3;
      sendArmData();
      Ang2Temp--;
      draw();
      return;
    }
    if (ev.getActionCommand().equals("Wrist Up")) {
      serv = 5;
      sendArmData();
      Ang3Temp++;
      draw();
      return;
    }
    if (ev.getActionCommand().equals("Wrist Down")) {
      serv = 6;
      sendArmData();
      Ang3Temp--;
      draw();
      return;
    }
```

```
if (ev.getActionCommand().equals("Pan Right")) {
  serv = 8;
  sendArmData();
  Ang4Temp--;
  draw();
  return;
}
if (ev.getActionCommand().equals("Pan Left")) {
  serv = 7;
  sendArmData();
  Ang4Temp++;
  draw();
  return;
}
if (ev.getActionCommand().equals("OPEN")) {
  serv = 9;
  sendArmData();
  Ang5Temp++;
  draw();
  return;
}
if (ev.getActionCommand().equals("CLOSE")) {
  serv = 10;
  sendArmData();
  Ang5Temp--;
  draw();
  return;
}


if (ev.getActionCommand().equals("View Error Log")) {
  FileDialog el = new FileDialog(this, "View Error Log", FileDialog.LOAD);
```

```
      el.show();
      String fn1 = el.getFile();
      try {
        fLogOpen = new FileInputStream(fn1);
        LogOpen = new ObjectInputStream(fLogOpen);
      } catch (Exception ex) {
        //messageArea.append("\nCannot Open Log.\n");
      }
      draw();
      return;
    }

    if (ev.getActionCommand().equals("SNAP PHOTO")) {
        snapPhoto();
        draw();
      return;
    }

  }//Action events

  public void snapPhoto()
  {
              DateFormat df = DateFormat.getDateTimeInstance
                        (DateFormat.SHORT, DateFormat.LONG);
              String str = new String();
              String photofilename = new String();
              str = df.format(new Date());
              photofilename = str.replace(':','_');
              photofilename = photofilename.replace('/','_');

              File outputFile = new File
                      (PROGRAM_LOCATION+photofilename+".jpg");
```

```
                URL url;

            BufferedImage img2 = new BufferedImage
                        (640,480, BufferedImage.TYPE_INT_ARGB);
                try {
          url = new URL(CAMERA_FILE_NAME);
          img2 = ImageIO.read(url);
        }
        catch (MalformedURLException e){
        //    messageArea.append("\nBad URL\n");
        }
        catch (java.io.IOException e){
       //     messageArea.append("\nBad File Reading\n");
        }

          try{
                ImageIO.write(img2, "JPG", outputFile);
          }
          catch(IOException e){messageArea.append("\nCannot save photo.\n");
          }
          messageArea.append("\nPhoto Saved "+photofilename+".jpg\n");

//        displayImage(img2, photofilename);

  }

// Function written to display the image in a separate window.
  public void displayImage(BufferedImage img, String filename){

                Graphics picGraph;
                ImageIcon picIcon;
                JFrame picturePage = new JFrame(filename+".jpg");
                Container c = picturePage.getContentPane();

                JLabel pic = new JLabel();

        //Put this into a JPanel
        JScrollPane picPanel = new JScrollPane(pic);
        picIcon = new ImageIcon(img);
        pic.setIcon(picIcon);
        picturePage.add(pic, BorderLayout.CENTER);
        pic.setName("P");
                picturePage.setSize(new Dimension(330,250));
                picturePage.setVisible(true);
    }


// Main. Set the GUI frame visible only.

// @param args The string array which is arguments passed
// @exception Exception

    public static void main(String[] args) throws Exception {
      BigFoot bf = new BigFoot();
      bf.setVisible(true);

        try {
          FileWriter erl = new FileWriter("error_log.txt", false);
          erl.write("|||");
          erl.close();
          }
        catch (IOException e) {
            System.out.println("Error -- could not open error log");
        }

        try {
            FileWriter gpsl = new FileWriter("gps_log.txt", false);
            gpsl.write(" ");
            gpsl.close();
            }
        catch (IOException e) {
            System.out.println("Error -- could not open gps log");
        }
    }//main
}//Bigfoot
```

LIST OF REFERENCES

[1] Department of Defense. Joint Robotics. http://www.jointrobotics.com, April 2007.

[2] Department of Defense. FY2005 UGV MASTER PLAN. http://www.jointrobotics.com/activities_new/2005%20JRP%20Master%20Plan.pdf, April 2007.

[3] Foster-Miller. Talon Robot Brochure. http://www.foster-miller.com/literature/documents/TALONBrochure.pdf, April 2007.

[4] Foster-Miller. Talon Robots. http://www.foster-miller.com/lemming.htm, April 2007.

[5] Federation of American Scientists. RQ-1 Predator Medium Altitude Endurance (MAE) UAV. http://www.fas.org/irp/program/collect/predator.htm, April 2007.

[6] Federation of American Scientists. UAV Tactical Control System. http://www.fas.org/irp/program/collect/uav_tcs.htm, April 2007.

[7] T. Dunbar. Demonstration of waypoint navigation for a semi-autonomous prototype surf-zone robot. Master's Thesis, Naval Postgraduate School, June 2006.

[8] B. Miller. Improvised explosive devise placement detection from a semi-autonomous ground vehicle. Master's Thesis, Naval Postgraduate School, December 2006.

[9] Superdroid Robots. IG32P 24VDC 190 RPM Gear Motor. http://www.superdroidrobots.com/shop/item.asp?itemid=374&catid=7, April 2007.

[10] NTE. NTE1914 Voltage Regulator, http://www.nteinc.com/specs/1900to1999/pdf/nte1914.pdf, April 2007.

[11] Fairchild Semiconductor. LM78XX/LM78XXA – 3 Terminal 1A Positive Voltage Regulator, http://www.fairchildsemi.com/ds/LM/LM7805.pdf, April 2007.

[12] Superdroid Robots. Electronic Magnetic Compass, http://www.superdroidrobots.com/shop/item.asp?itemid=128&catid=35, April 2007.

[13] Phillips Semiconductors. Application Note, Electronic Digital Compass Design using KMZ51 and KMZ52, AN00022, 12.

[14] J. Fraden. AIP Handbook of Modern Sensors, American Institute of Physics, New York, 1993, pp. 487-489.

[15] Superdroid Robots. SRF08 Support, http://www.superdroidrobots.com/product_info/SRF08.htm, April 2007.

[16] J. Fraden. AIP Handbook of Modern Sensors, American Institute of Physics, New York, 1993, pp. 303-305.

[17] Superdroid Robots. GP2D12. http://www.superdroidrobots.com/product_info/SharpGP2D12-15.pdf, April 2007.

[18] Savage Innovations. GP2D12. http://www.oopic.com/gp2d12.htm, April 2007.

[19] Superdroid Robots. TPA81. http://www.superdroidrobots.com/product_info/TPA81.htm, April 2007.

[20] D-Link. DCS-900. http://www.dlink.com/products/resource.asp?pid=270&rid=807&sec=0, October 2006.

[21] Garmin. WAAS. http://www.garmin.com/about/GPS/waas.html, August 2006.

[22] Garmin. GPS 16. http://www.garmin.com/manuals/425_TechnicalSpecifications.pdf, April 2007.

[23] Z-World. BL2000. http://www.zworld.com/products/bl2000/, April 2007.

[24] Dennis Clark. Programming and Customizing the OOPic Microcontroller, McGraw-Hill, 2003, p. 17.

[25] D-Link. 2.4 GHz Omni-Directional 7 dBi Indoor Antenna. http://www.dlink.com/products/resource.asp?pid=416&rid=1551&sec=0, April 2007.

[26] T. Williamson. Modeling and Implementation of PID Control for Autonomous Robots. Master's Thesis, Naval Postgraduate School, June 2007.

[27] K. Uzun. SE4015 Class Presentation (June 2003).

[28] J. Herkamp. et al. SE4015 Class Presentation (September 2006).

[29] B. Kerstens. et al. SE4015 Class Presentation (March 2007).

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

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

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

5. Physics Department
   Naval Postgraduate School
   Monterey, California