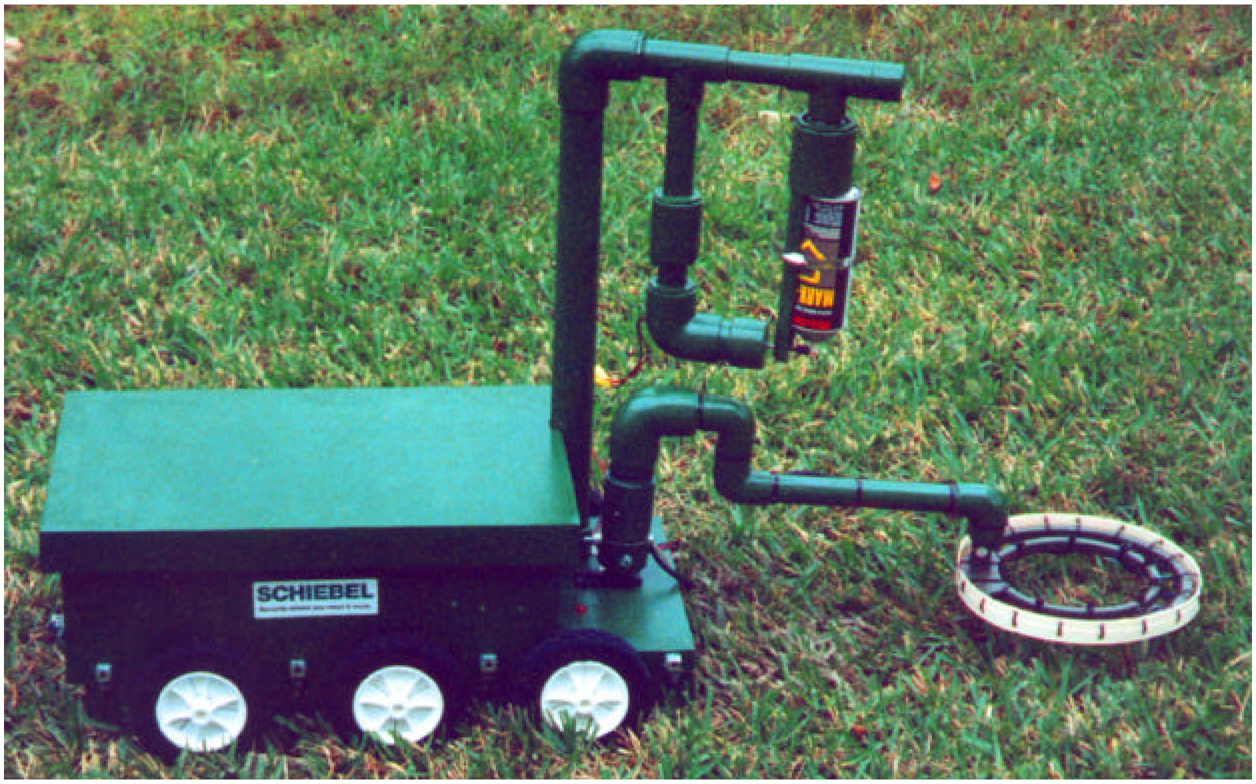


Little Ranger

Land Mine Detection and Marking Robot

Final Report

Phase I

Daniel Kucik
University of Florida
I.M.D.L.
Dr. A. Antonio Arroyo
December 3, 1998

Foreword

Little ranger was a major undertaking, involving months of researching available land mine detection methods and manufactures. Furthermore, proposals were prepared for various manufacturers while attempting to obtain a land mine detector donation. The project began the Spring 1998 semester, but as a result of the time required for researching the above mentioned items and various other delays in receiving the detector donation, the project carried over into the Fall 1998 semester.

Myself, as well as several university faculty members, feel this project has great potential, and for this reason the work started here will be continued, either by myself, or other students at the University of Florida. This is the first phase of the project, which proved that this is a sound concept with realistic goals, and it has laid the groundwork for future development.

I would like to take this opportunity to thank Hans Schiebel for making this project possible through his generous donation of the Schiebel AN-19/2 mine detector.

This document, additional pictures, and useful routines developed for this project can be found at http://www.ameriwebtek.com/dpk/.

Table of Contents

Abstract

This project investigates the feasibility of utilizing an autonomous robot to detect and mark land mines. Mine detection is achieved using a Schiebel Technologies AN-19/2 mine detector, and the marking of mines is accomplished through the use of spray paint. The obstacle avoidance sensor suite includes 11 infrared LED/detector sets and 19 bump switches. An expanded Motorola 68HC11 microprocessor programmed with Interactive C and 6811 Assembly controls functionality and behaviors.

The Little Ranger reacts to the detection of a land mine by moving the spray paint can over the mine and painting it, then the robot steers around the mine and continues the search. The obstacle avoidance routines were very limited as a result of memory restrictions. For this reason if an object is detected in an area other than the front, Little Ranger simply stops and waits for the object to be cleared. When an obstacle is found in front, the robot steers around the object and continues searching.

Executive Summary

The objective of this project is to prove the feasibility of using an autonomous robot to detect and mark land mines in a safe, accurate manner. Little Ranger's behaviors are controlled using an expanded Motorola 68HC11 EVBU board. Since all behaviors and decisions are controlled by the microprocessor, the presents of an operator is not required, which translates into saved human lives. A Schiebel Technologies AN-19/2 is utilized for detecting land mines. The AN-19/2 is used throughout the world by various armed forces, including the United States Army (designation AN/PSS-12); therefore, it is an accurate means of detection that has been thoroughly tested in "real-life" conditions. The marking of detected mines is accomplished through the use of a spray paint can, which is actuated by a solenoid. The spray paint assembly is attached to a PVC arm that extends away from the body of the robot, allowing for the robot to paint mines without running over them.

Unfortunately, as a result of unexpected memory constraints, Little Ranger's behaviors are limited. Currently the robot has the capability of searching for mines, marking the location of detected mines, avoiding mines, avoiding obstacles found in front of the robot, and stopping until obstacles in other areas can be cleared.

Introduction

Background Information

Currently more than 120 million mines are deployed in over 60 countries throughout the world$_1$. Approximately 2 million new mines are set yearly, while only 100,000 are removed$_1$. Therefore, this is still a growing problem with an overall increase of about 1.9 million new mine deployments yearly. One reason for their popularity is their low cost, only about \$3/mine$_2$, which makes them very appealing to third-world countries. The United States currently utilizes "smart" mines, which automatically deactivate after a predetermined period of time$_3$; therefore after the military action is completed there is no chance of civilians being injured. Unfortunately, other countries do not deploy these "smart" mines as a result of their higher cost, which only adds to the overall problem. Presently, more than 1 million people have been killed or maimed by mines, and this number is increasing by about 26,000 a year$_2$.

After completing a patent search at the start of this project, I was not able to locate an autonomous robot capable of detecting and marking (or removing) land mines. Additional research was completed attempting to locate companies and/or universities with similar concepts in the development stage, which currently might not be patented, but no information could be found. However, it was found that there are non-autonomous robots on the market that require operators to provide control, as well as to determine the existence of mines. After completing extensive research, it appears that this is currently the only project developing the concept of land mine detection through the use of an autonomous robot.

Project Objectives

The objectives for this phase of the project starting from most important and ending with the least important are:

1) Detect land mines.
2) Mark the location of mines.
3) Go around detected mines without detonating the mine.
4) Avoid obstacles.
5) Maintain a controlled search pattern, while incorporating adjustments for avoided mines and obstacles.

During the first phase of this project, the main goal was to prove that the concept of an autonomous robot detecting and marking land mines is feasible. This objective included completing at the very least objectives one through three. Any additional objectives that could not be completed during the first phase as a result of mechanical or hardware limitations will be completed in subsequent phases after resolving the limitations.

Integrated System

Little Ranger is comprised of several individual systems, each contributing a specific function to the overall capabilities of the robot. The platform is designed to provide support and protection to all systems and components. Mobility for the platform and the systems it carries is provided through the actuation system. The obstacle avoidance sensors provide information about objects around Little Ranger that may inhibit its actions or movements. The primary sensor is designed to detect land mines and provide their locations. Little Ranger’s functions and behaviors are managed by the control system. The sweep arm is used to attach the mine detector’s search head to the platform and to provide the head with the necessary actuation. The marking system is used to mark the location of detected land mines through the use of spray paint. Power, electrical protection, and noise suppression for all systems described above are supplied by the power system. For a summary of the systems please refer to Table 1, and for system interactions refer to Figure 1. Figure 2 shows the location of several external systems and components.

Table 1– Summary of Little Ranger's Systems

Name	Parts	Function
Mobile Platform	1) Two side panels 2) Bottom panel 3) Front panel 4) Rear panel 5) Cover 6) Control panel	Support and protect all system components.
Actuation	1) Six 6-inch lawn mower wheels 2) Six hacked 333 oz-in servos	Provide mobility to the platform.
Obstacle Avoidance Sensors	1) 11 infrared LED/detector sets 2) 19 bump (push-button) switches	Provide information about surrounding objects and possible obstacles.
Primary Sensor	1) Schiebel AN-19/2 mine detector 2) Interfacing electronics	Provide information regarding the existence of land mines.
Sweep Arm	1) PVC arm 2) 333 oz-in servo	Attach the mine detector's search head to the robot and provide actuation to the head.
Marking System	1) PVC support arm 2) Solenoid 3) Control electronics 4) Spray Paint Can	Mark the location of detected land mines.
Control System	1) Motorola 68HC11 EVBU 2) Novasoft ME11 board 3) Expansion Electronics	Control behaviors and functions.
Power System	1) Three 6-Volt motorcycle batteries 2) 8 AA batteries 3) 4 D batteries 4) Noise suppression components 5) Fuses	Provide power, protection, and noise suppression to all systems

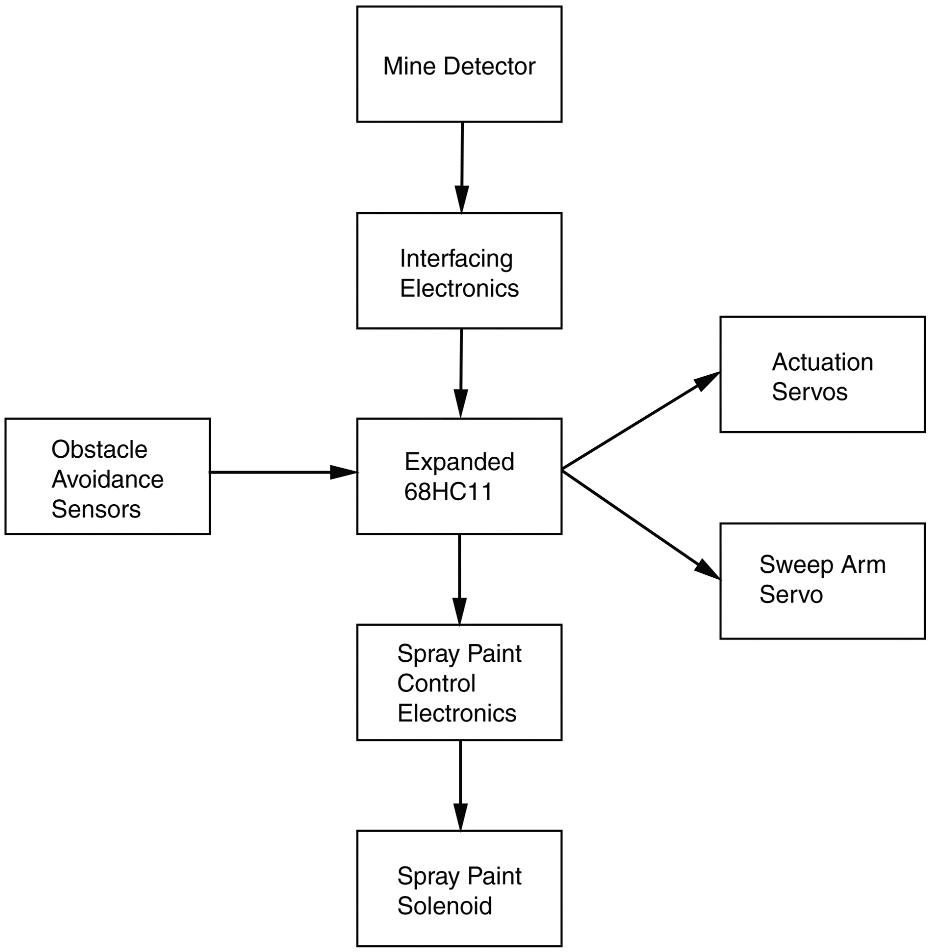


Figure 1 – System Interactions

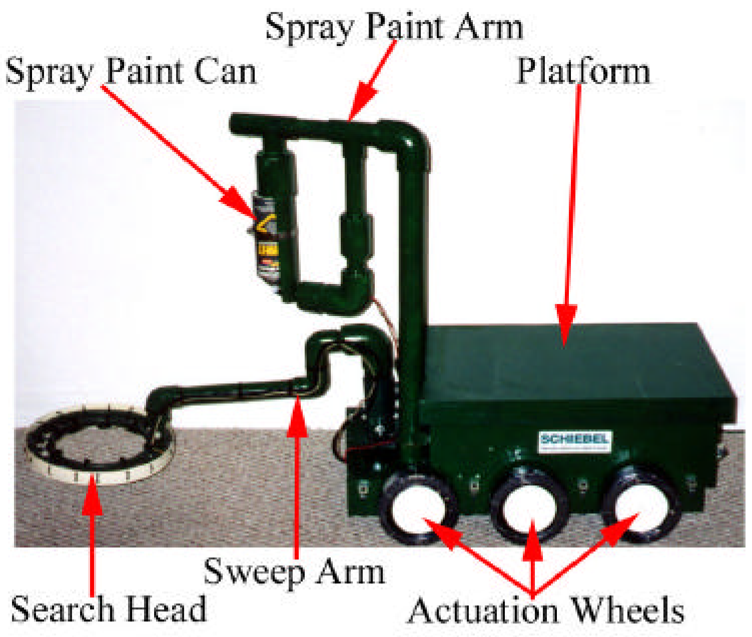


Figure 2 – Various External Components

Mobile Platform

The purpose of the mobile platform is to support and protect Little Ranger's systems and components. While designing the platform, the following requirements were considered and successfully implemented:

1) Withstand outdoor conditions.
2) Protect the electronics from the outdoor environment.
3) House the required electronic components and battery packs.
    a) Motorola 68HC11 EVBU board and the ME11 expansion board
    b) Schiebel AN-19/2 electronics unit
    c) AN-19/2 user interface
    d) mine detector signal modification circuitry
    e) bump switch interfacing board
    f) spray paint actuator control electronics
    g) 8-AA batteries for the 68HC11 board
    h) 4-D batteries for the mine detector
    i) three 6-volt motorcycle batteries for the servos and marking system
4) Stable enough not to move when the search head sweeps back and forth.
5) Support the sweep and spray paint arms.
6) Small turning radius, so that mines can be easily avoided.

The platform was built using the following materials:

- 1”x12” wood planks
- 1/2” plywood
- 1/4” plywood
- 1-1/2” x 1/2” wood strips
- 2” #8 wood screws
- 1.5” #8 wood screws

The seven parts of the platform are:

1) bottom panel
2) left side panel
3) right side panel
4) rear panel
5) front panel
6) cover
7) user interface panel

<u>Bottom Panel</u>

This part of the platform measures 12”x26”x1”. It contains a cutout at the front to house a servo for the sweep arm and a circular hole for wires to enter the body from the outside. Please refer to Figure 3.

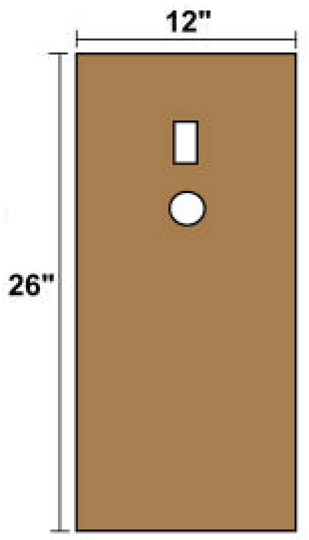


Figure 3 – Bottom Panel

Side Panels

Figure 4 depicts the side panel, which is used on both the left and right side; it measures 9”x26”x1”. Each panel contains three cutouts used to house the propulsion servos.

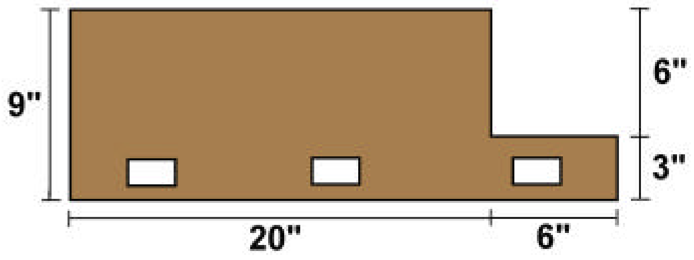


Figure 4 – Side Panels

Rear Panel

The rear panel, pictured in Figure 5, is 6.75”x12.75”x1”. There is a circular cutout for a switch that is used to quickly stop the robot without removal of the cover.

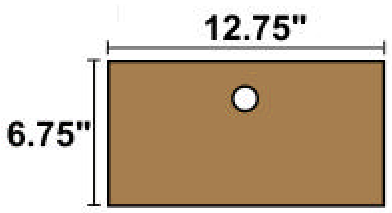


Figure 5 – Rear Panel

Front Panel

This section, seen in Figure 6, measures 6”x11.25”x1”.

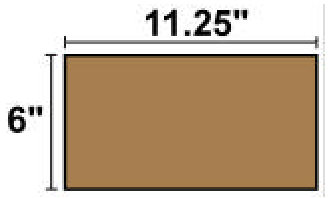


Figure 6 – Front Panel

Cover

The cover measures 21.75”x14.5”. It has a lip around it to allow for secure attachment to the rest of the body. The main part of the cover is 1/2” plywood, and the lip is made of 1-1/2”x1/2” lumber. The lip is attached using 1-1/2” #8 wood screws; pilot holes and counter-sinks were drilled for each screw to prevent the wood from splitting. The cover is attached by slipping it over the top of the side, rear, and front panels. Refer to Figure 7 for the cover design.

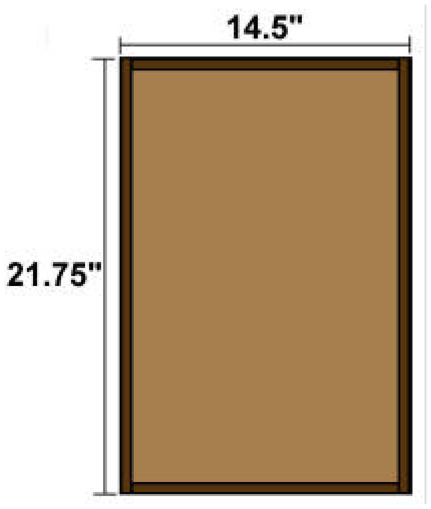


Figure 7 – Cover

User Interface Panel

Figure 8 is the user interface panel, which is constructed of 1/4” plywood and measures 11.25”x6”x1/4”. It holds the mine detector’s user interface and switches to control the power supplied to various components. Two slots, seen in Figure 9, were created inside the platform’s electronics compartment using 1-1/2”x1” wood to allow the panel to be easily installed and removed.

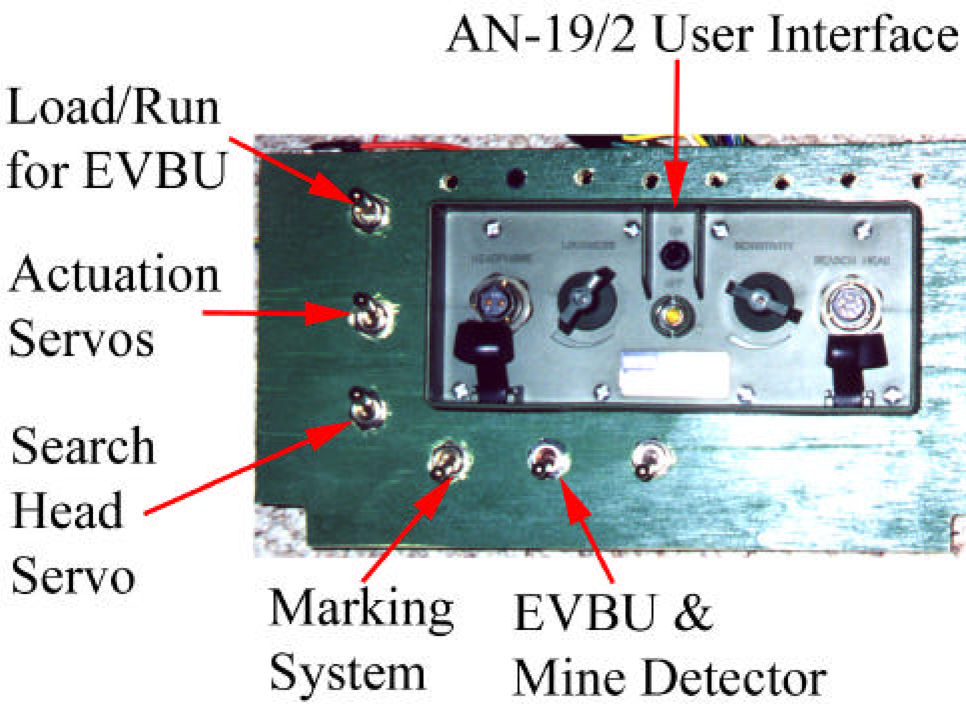


Figure 8 – User Interface Panel

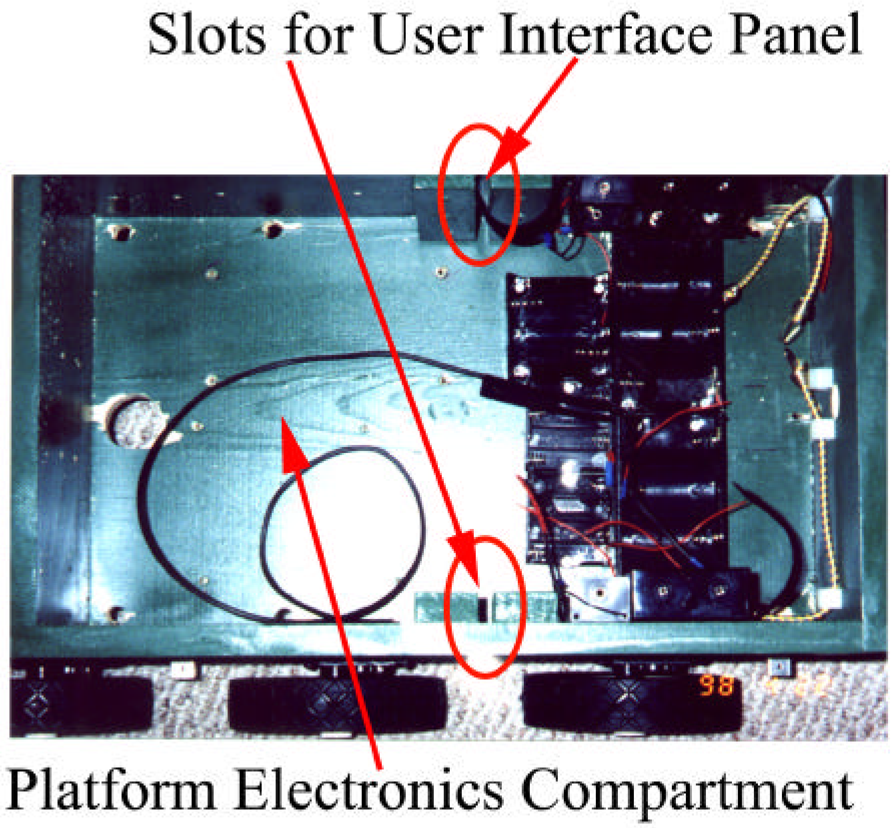

Slots for User Interface Panel
Platform Electronics Compartment

Figure 9 – Guide Slots For User Interface Panel

Assembly

The panels making up the platform were assembled, as shown in Figures 10 and 11, using 2-inch #8 wood screws spaced approximately every 2 inches. Holes and counter-sinks for the screws were drilled to prevent the wood from cracking.

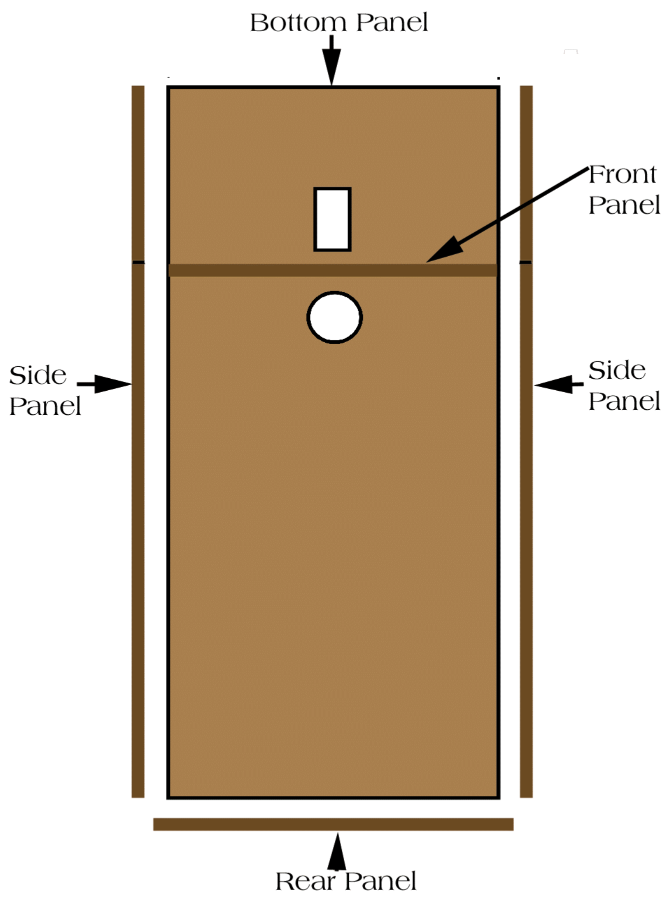


Figure 10 – Top View of Assembly

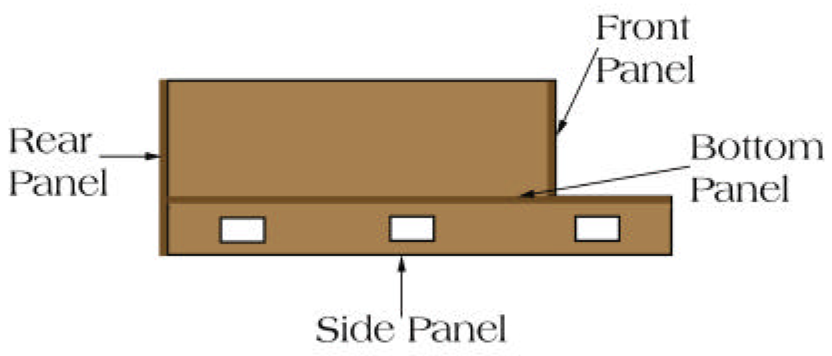


Figure 11 – Side View of Assembly

Actuation

Little Ranger's actuation system is designed to provide mobility to the platform and all components it houses and supports. This system consists of six 6-inch lawn mower wheels each attached to modified 333 oz-in servos (refer to Figure 12). Full rotation of the output gear on the servos was achieved by disconnecting the internal pot resistor from the output gear and setting it to its center position. Furthermore, the mechanical stopping mechanism (notch), seen in Figure 13 on the output gear was cut away to allow full rotation.

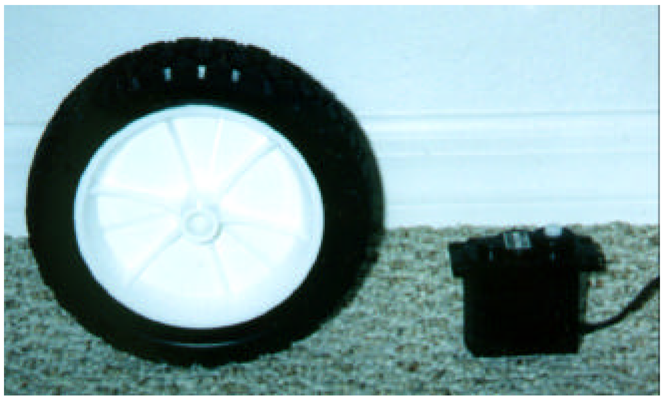


Figure 12 – Wheel and Servo

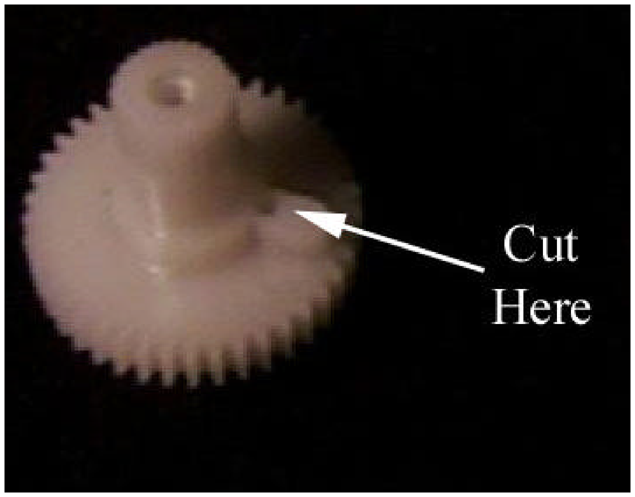


Figure 13 – Servo Output Gear Modification

The original internal servo electronics were found to be outstanding in comparison to other servos, utilizing power field effect transistors, which can easily handle the high current required by the servos to move the platform. For this reason, the original electronics were used in conjunction with pulse width modulation generated by the 68HC11 to control the servos.

All three servos on each side have a shared control line; therefore, the wheels on each side rotate at approximately the same speed all of the time. This design allows Little Ranger to turn by reversing one side while having the other side go forward (similar to a tank), which reduces the turning radius.

The actuation system was found to have one major problem, which is turning on rough surfaces. The cause appears to be a result of overly optimistic specifications for the servos and the wheels having too much traction. Little Ranger can move forward on outdoor surfaces (grass), but while attempting to turn, the servos "bog-down" and just cannot provide enough torque to complete the turn. On controlled surfaces, such as tile floors, turning is not a problem, unfortunately though, such flooring often has steel reinforcements in the concrete, which poses a problem for the mine detector.

Obstacle Avoidance Sensors

The obstacle avoidance sensors are used to provide Little Ranger with information about surrounding objects that may impede its ability to perform operations and movements. Sharp and Radio Shack infrared detectors were modified, as shown in Figures 14 and 15, to output an analog signal, which allows the strength of the IR source to be measured. The modified detectors were combined with infrared LED's to produce the sensor, which operates under the same principle as sonar. The IR signal is emitted in the direction of interest, and if an object is present, the signal is reflected back and detected by the infrared detectors. The distance to a detected object can be determined based on the output level (voltage) of the IR detectors.

Eleven IR sensor sets were positioned in various positions on Little Ranger. Figure 16 shows the positioning of three sensors on the rear, and Figure 17 shows the location of four on each side of Little Ranger.

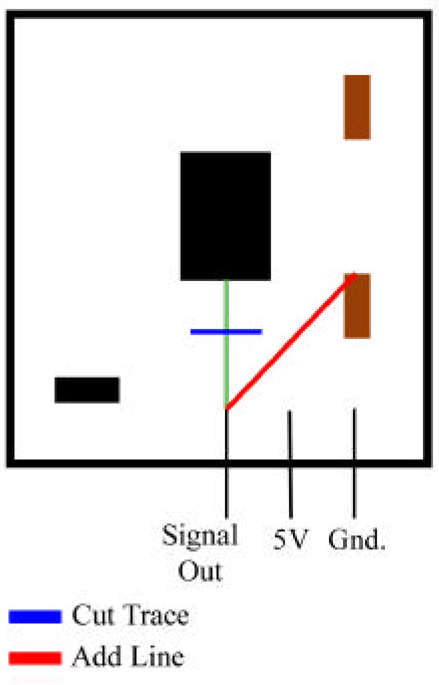


Figure 14 – Radio Shack 276-137 IR Detector Hack

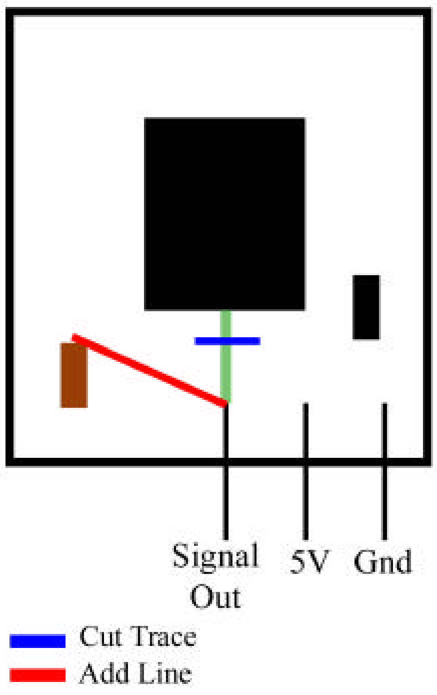


Figure 15 – Sharp GP1U58Y Detector Hack Provided by IMDL

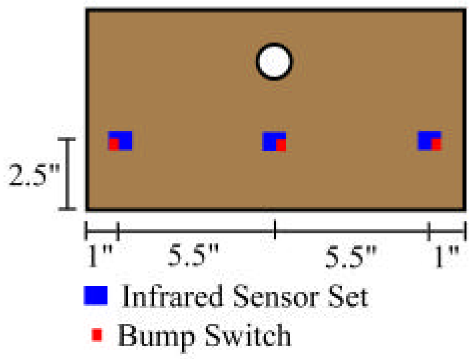


Figure 16 – Rear Panel Sensor Positioning

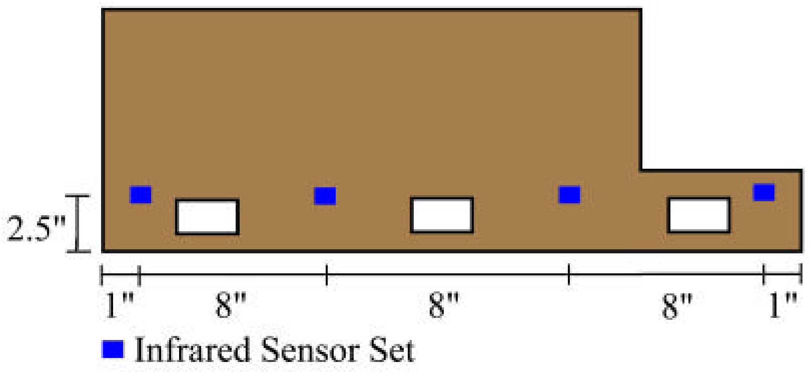


Figure 17 – Side Panel Sensor Positioning

Nineteen miniature push-button or “bump” switches were used to detect collisions. Figure 16 shows the positioning of five switches on the rear panel of Little Ranger. Fourteen bump switches were placed on the search head of the primary sensor, the locations of which are depicted in Figure 18. A flexible ring was placed around the search head which activates the switches when an object is hit in an area not containing any sensors (refer to Figure 19).

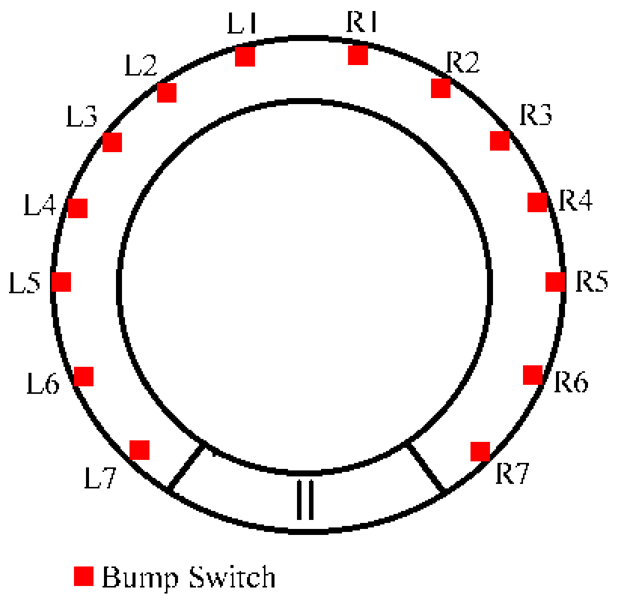


Figure 18 – Search Head Sensor Positioning

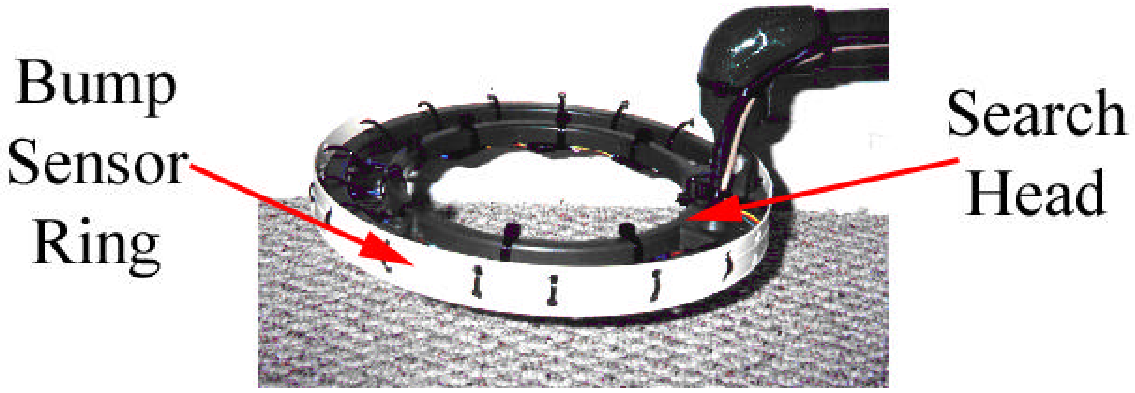


Figure 19 – Search Head Bump Sensor Ring

Primary Sensor

Little Ranger’s primary sensor is Schiebel Technologies’ AN-19/2 land mine detector, pictured in Figure 20, which is used by many armed forces throughout the world, including the United States Army (designation AN/PSS-12). The AN-19/2 is an extremely sensitive metal detector, capable of easily detecting a mine’s firing pin. The detector was obtained through a generous Schiebel Technologies donation.

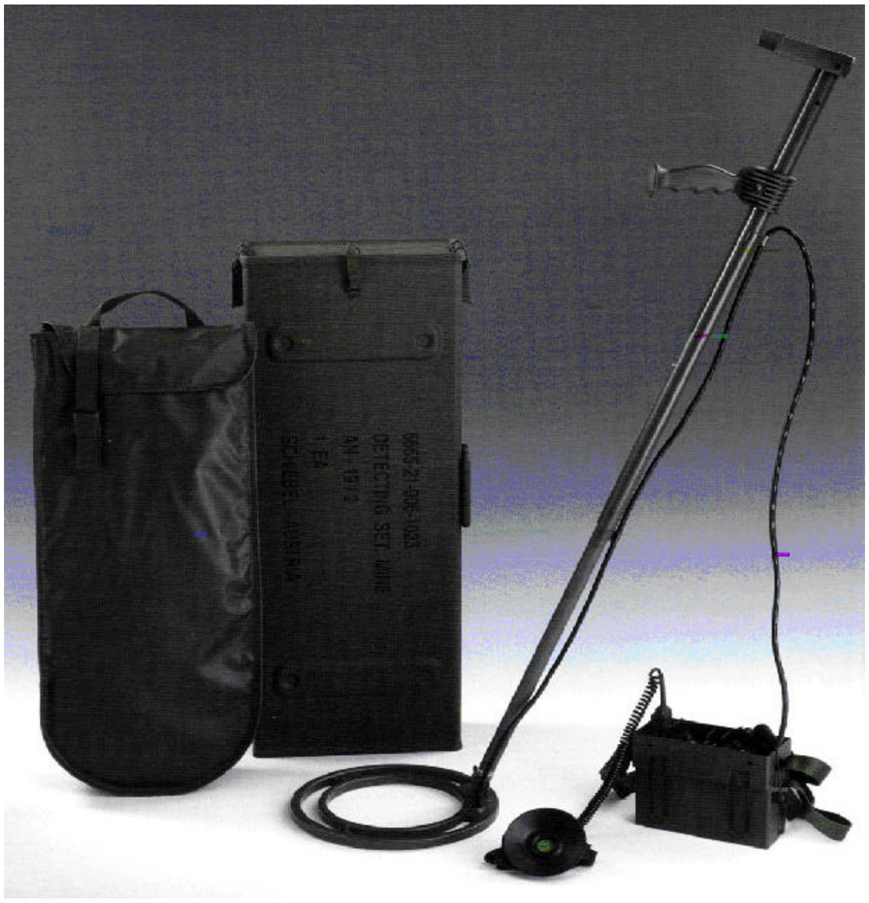


Figure 20 – Schiebel Technologies AN-19/2 Mine Detecting Set

<u>Components</u>

The mine detector consists of three main parts, the search head, electronics unit, and user interface. The search head, seen in Figure 21, is attached to the sweep arm, which sweeps over the area of interest looking for mines. The electronics unit is housed inside the platform and is connected to electronics that modify its output signal for use by the control system. The user interface, seen in Figure 22, is also located inside the platform; it contains various sensitivity controls for the detector and ports for headphone and search head connections.

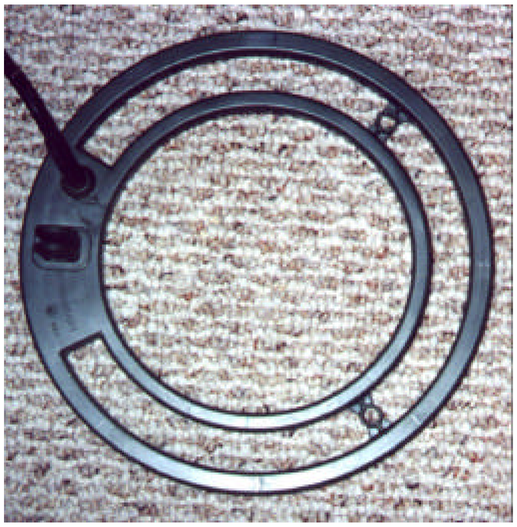


Figure 21 – AN-19/2 Search Head

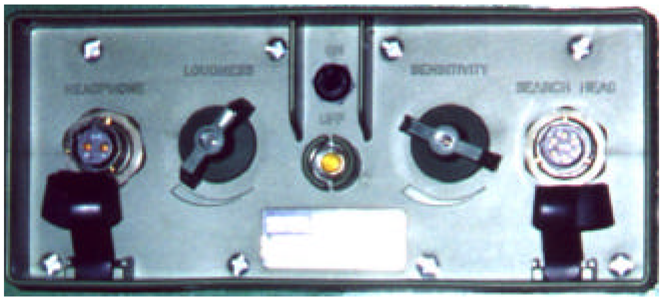


Figure 22 – AN-19/2 User Interface

Principle of Operation

The mine detector works by generating a magnetic field via a pulse generator and the transmitting coil. This causes Eddie Currents to flow in metal objects located below the coil, which causes a secondary magnetic field. The receiving coil then detects the newly generated secondary field and the information is processed to produce the mine detectors output signal. Please refer to Figure 23 for a summary of the operation.

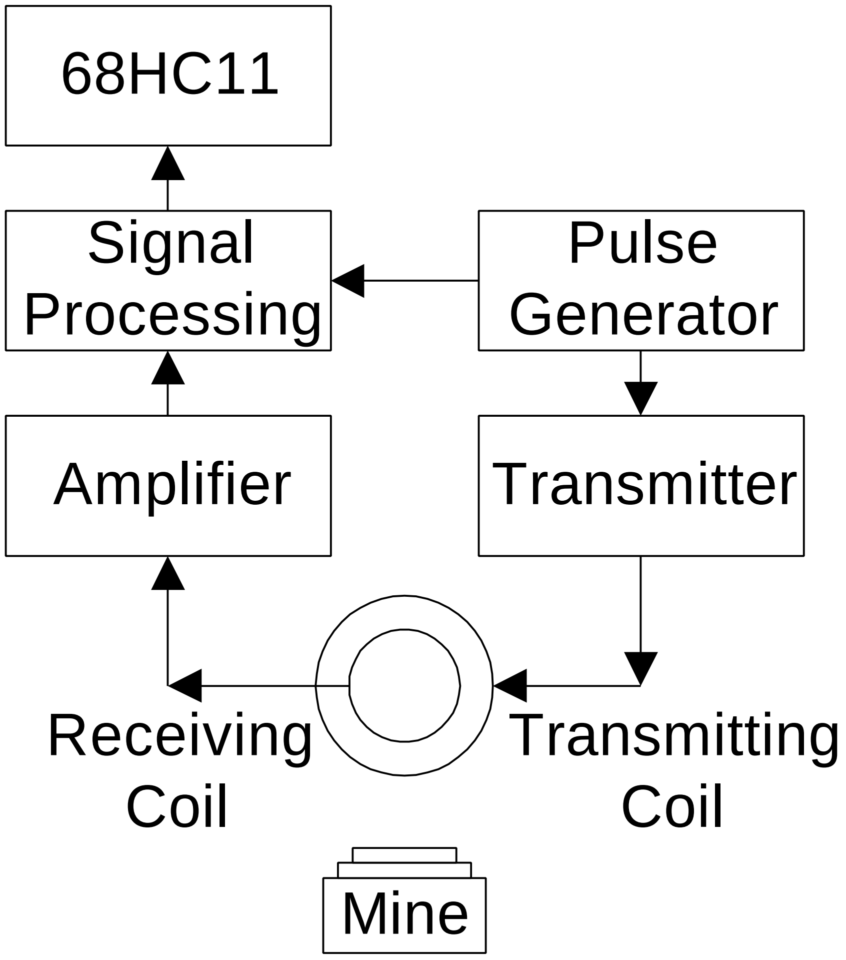


Figure 23 – Mine Detector's Principle of Operation

Modifications

The circuit used to modify the output signal is shown in Figure 24. It modifies the signal from its original waveform shown in Figure 25, to a relatively stable signal shown in Figure 26. This modification was necessary to reduce the possibility of the 68HC11 sampling at a time when the signal was at a minimum, which might result in a mine not being detected since the minimum voltage is the same as the constant voltage outputted when no mine is detected.

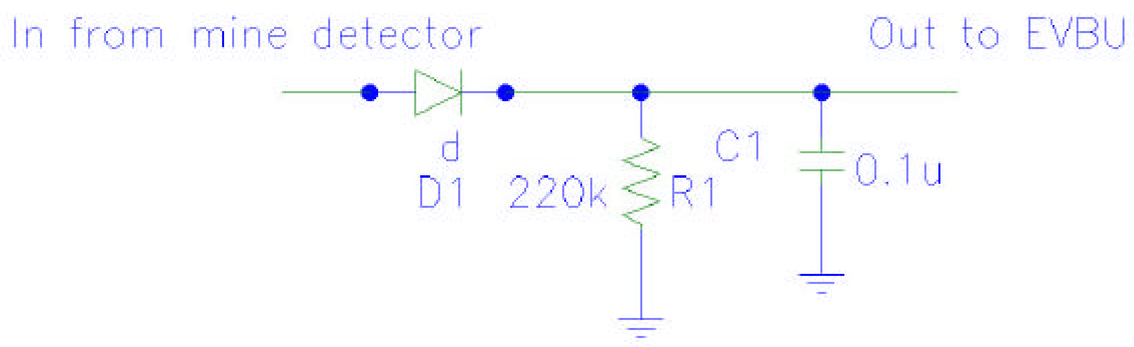


Figure 24 – Electronics to Interface AN-19/2 to 68HC11

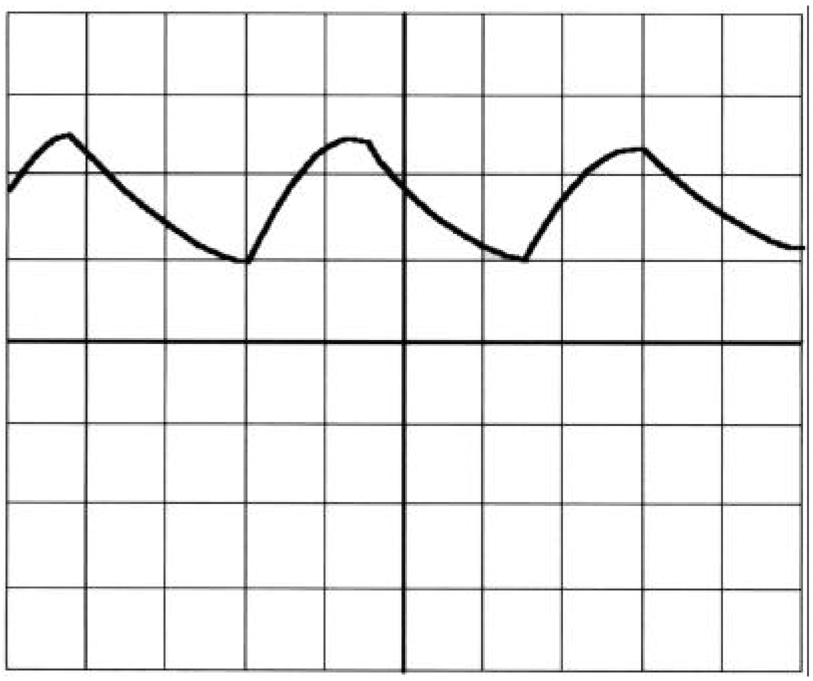


Figure 25 – AN-19/2 Output (Mine Detected) Before Modifications
Oscilloscope settings: 2V/Div and 0.2 mS/Div

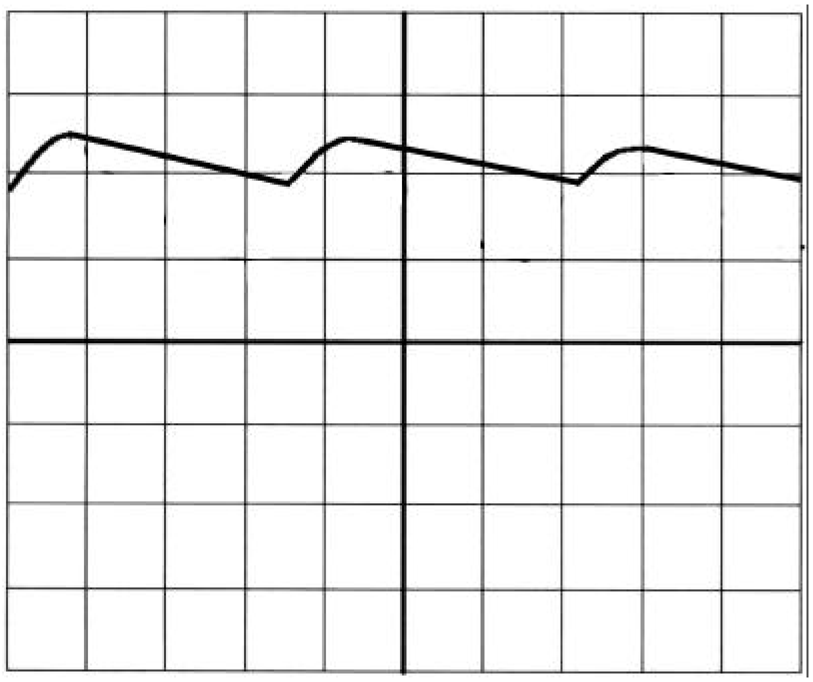


Figure 26 – AN-19/2 Output (Mine Detected) After Modifications
Oscilloscope settings: 2V/Div and 0.2 mS/Div

Control System

Little Ranger's control comes from an expanded Motorola 68HC11 EVBU board. The microprocessor's memory was expanded to 32 KB and eight 40kHz modulated outputs were added through Novasoft's ME11 expansion board. An additional 24 analog input ports and 8 digital output ports were added, please refer to Figures 27 and 28 for schematics of the input and output expansions respectively.

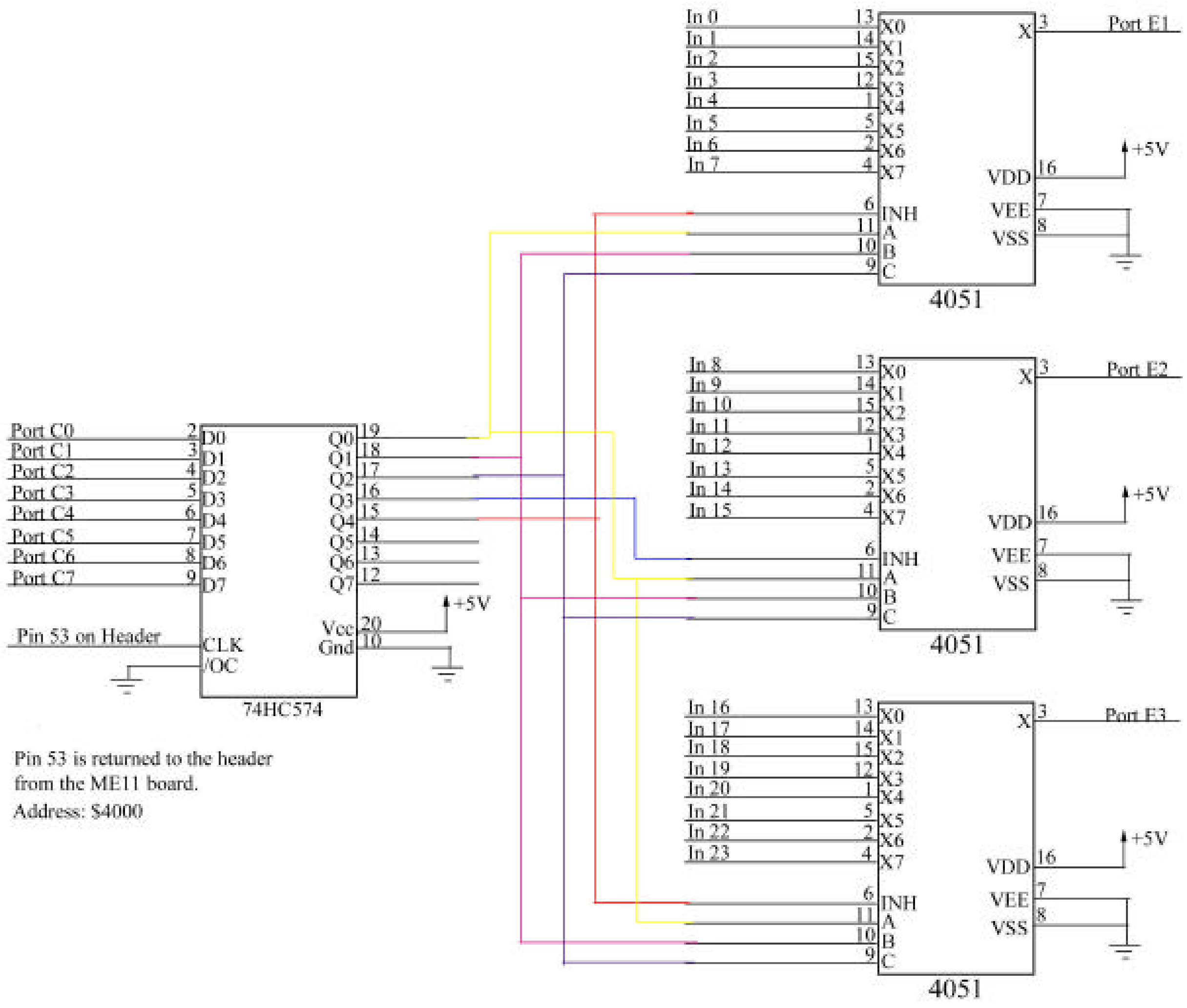


Figure 27 – Analog Input Expansion

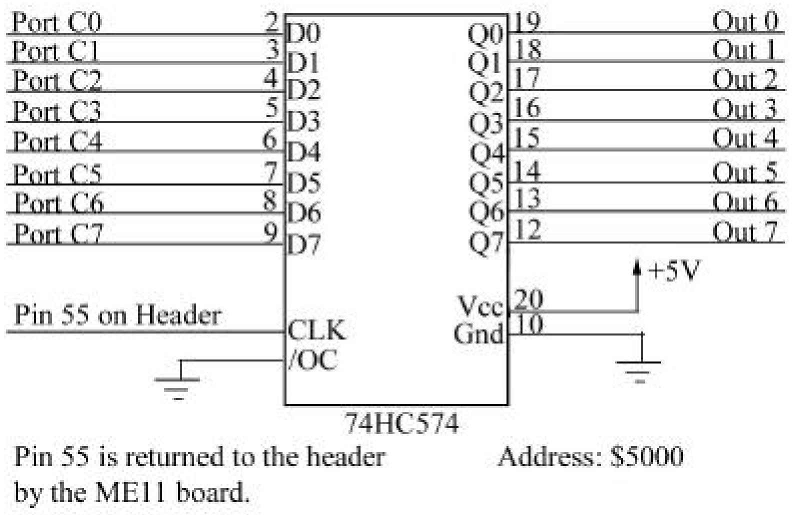
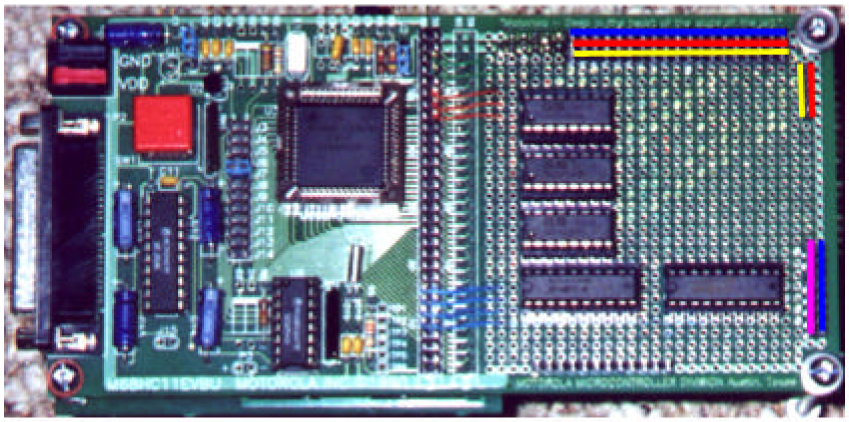


Figure 28 – Digital Output Expansion

The added analog inputs were used to collect values from the obstacle avoidance sensors. A power bus was also added so the infrared detectors could easily be connected to the EVBU without having to branch elsewhere to get the necessary power (refer to Figure 29).



Key

🟨 Analog Input
🟪 Digital Output
🟥 +5 Volts
🟦 Ground

Figure 29 – Expansion Bus

One of the digital output ports is used to control the spray paint actuation system. The additional ports can be used at a later time for robot status indication by connecting LED’s.

The 40-kHz modulated outputs are used to power the infrared LED’s used as part of the obstacle avoidance sensor suite. This modulation is needed so the 40 kHz IR detectors can detect the signal emitted from the LED’s.

Sweep Arm

The sweep arm, pictured in Figure 30, is designed to attach the primary sensor’s search head to the platform while providing the necessary actuation. The 18-inch long arm was constructed using PVC because no metal components were required, and therefore the arm would not cause false mine detector readings. Arm actuation comes from a 333 oz-in servo, which provides the required search head speed of 1 meter/sec.

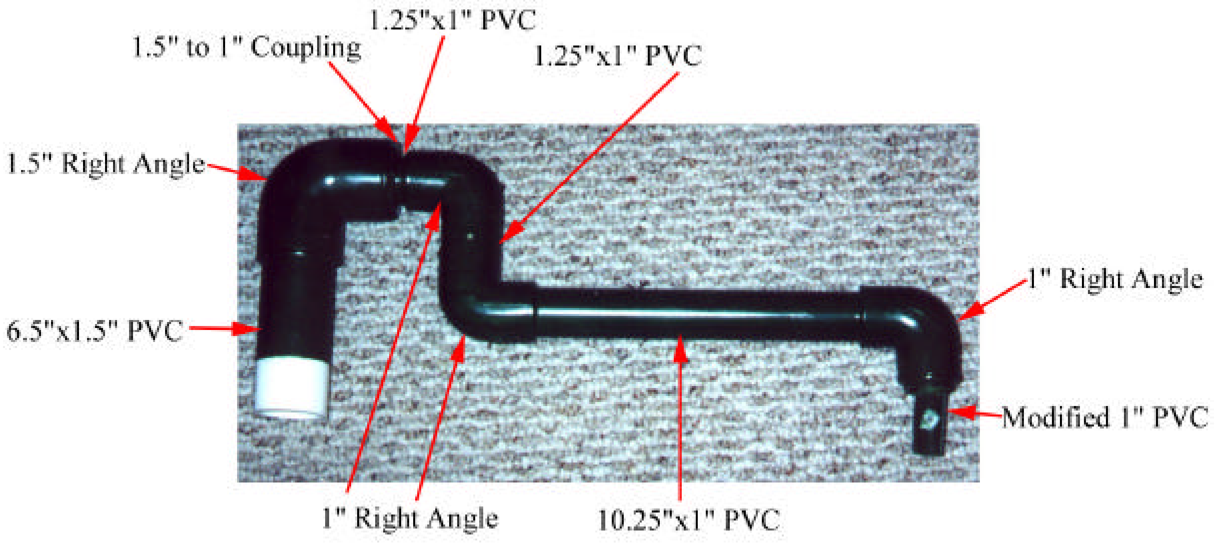


Figure 30 – Sweep Arm

One problem found with the servo is that nylon gears and horns were used, which resulted in extreme flexing at the point where the arm is connected to the servo. The required search head height is approximately 3 inches, so the original arm design was such that without flexing the head would be 5 inches above the ground, allowing 2 inches to be lost as a result of flexing. Unfortunately, the flexing was much worse than originally expected, which resulted in the search head dragging on the ground. For this reason, the reinforcements seen in Figures 31 through 33 were machined for the horn, which is used to connect the arm to the servo. With the new design, only about 1.5 inches of height is lost, and the search head is positioned about 3.5 inches above the ground.

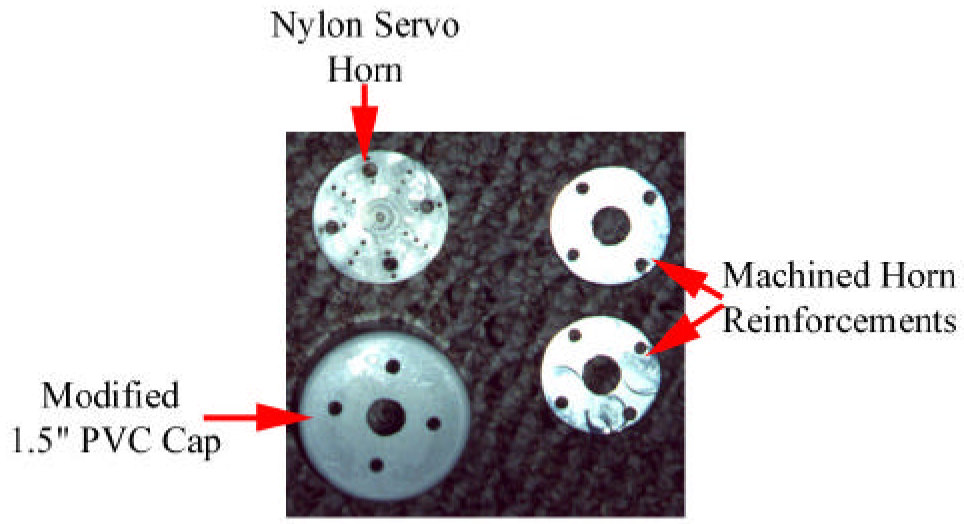


Figure 31 – Sweep Arm Servo Attachment Parts

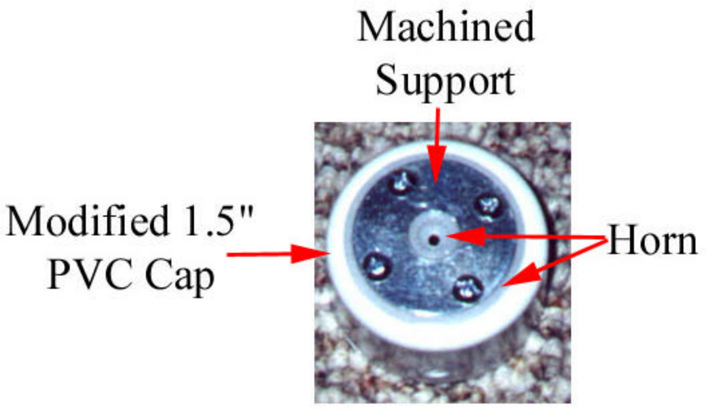


Figure 32 – Bottom View of Assembled Sweep Arm Attachment

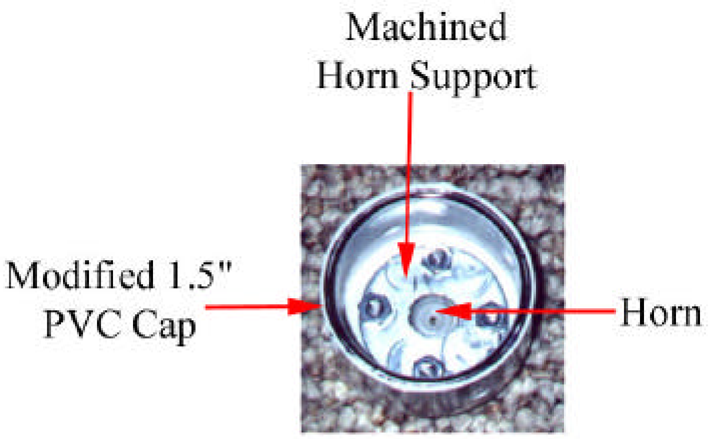


Figure 33 – Top View of Assembled Sweep Arm Attachment

An additional modification was made to allow the arm to be removed from the servo without cutting the PVC. This was accomplished by using bolts to connect the arm to the joint that is attached to the servo. Figure 34 shows the new design before it is assembled, and Figure 35 shows it after assembly. The arm is attached to the servo as shown in Figure 36.

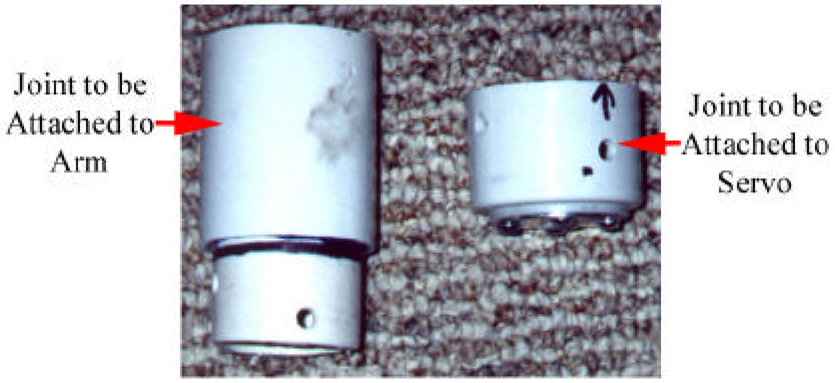


Figure 34 – Disassembled Overall Sweep Arm Support

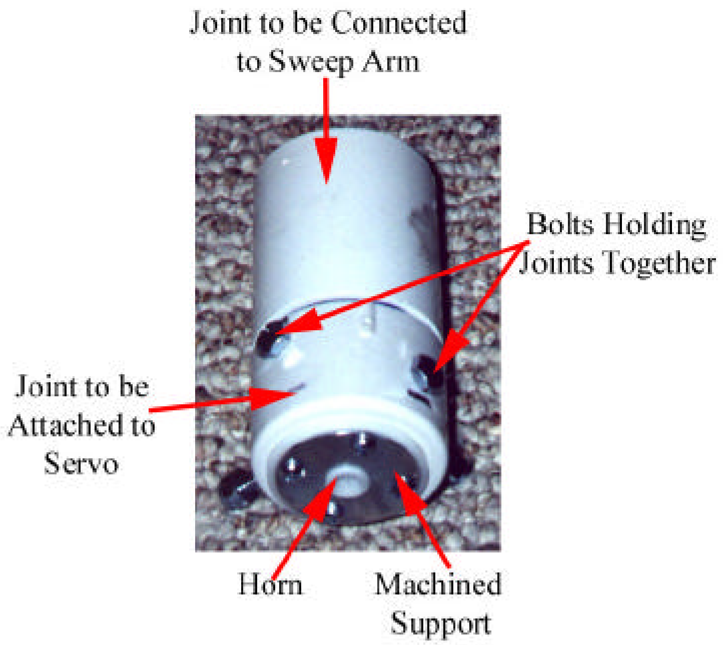

Joint to be Connected to Sweep Arm
Bolts Holding Joints Together
Joint to be Attached to Servo
Horn
Machined Support

Figure 35 – Assembled Overall Sweep Arm Support

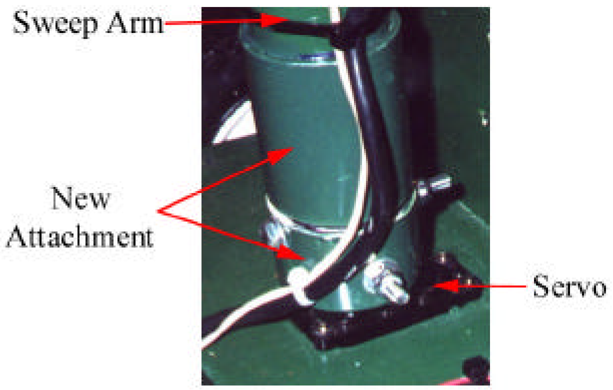


Figure 36 – Sweep Arm Attached to Servo

Marking System

Overview and Mechanical Design

This system is used to mark the location of detected land mines. Marking is done using a spray paint can attached to the PVC arm pictured in Figure 37 and is actuated using a solenoid (refer to Figure 38). Figure 39 shows the overall arm assembly with the spray paint can installed, and Figure 40 shows the attachment of the arm to Little Ranger. The spray paint can is designed to operate in the inverted position, so there are no problems with paint flow. The arm was necessary to extend the assembly away from the platform allowing Little Ranger to paint without running over mines.

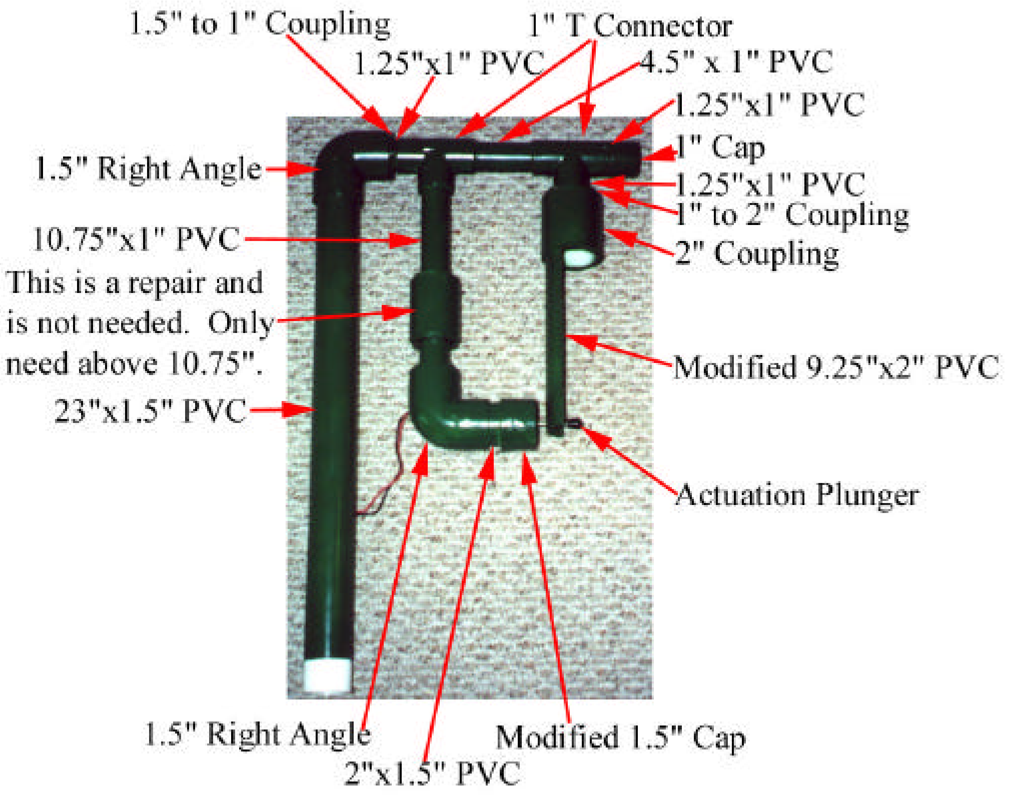


Figure 37 – Spray Paint Arm Dimensions

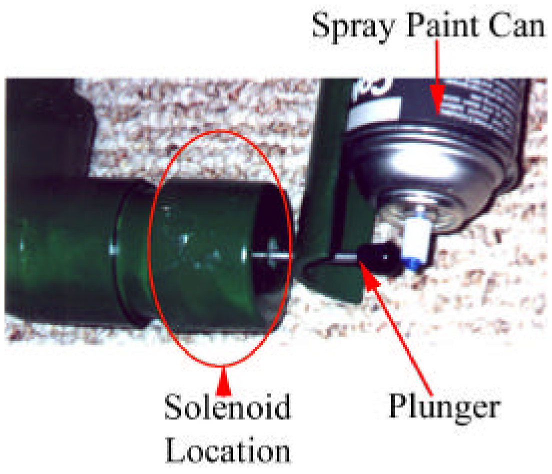


Figure 38 – Spray Paint System Components

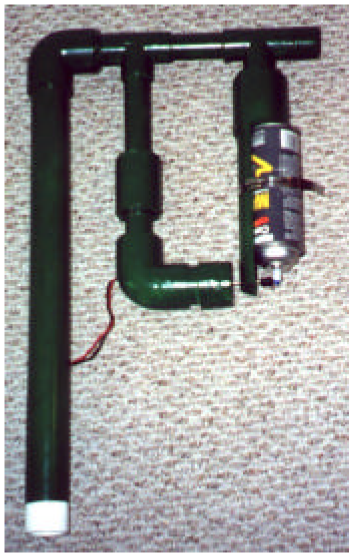


Figure 39 – PVC Spray Paint Arm

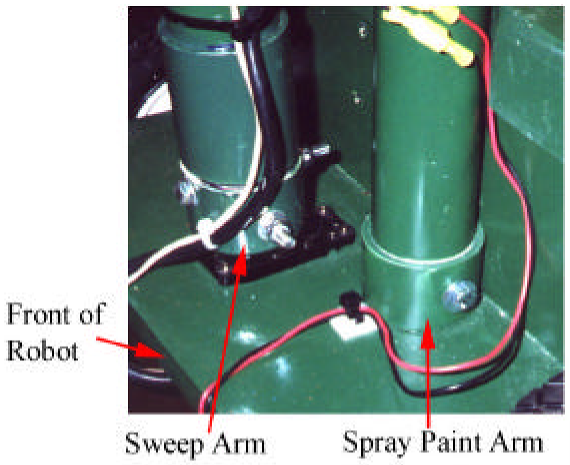


Figure 40 – Arm Attachments

Spray Paint Control

A "standard" Ford 12-volt starter solenoid was disassembled and the internal solenoid was extracted and used to actuate the spray paint can. The circuit in Figure 41 controls the solenoid used to actuate the spray paint can; it is controlled by a digital output on the Motorola EVBU board. It was necessary to use two relays because a relay capable of carrying the required current for the solenoid and having a coil voltage of only 5V could not be located. So, the first relay, which is connected to the EVBU, is used to control the second relay. The output of the second relay supplies power to the spray paint actuation solenoid. A switch was also installed on the board containing the control circuit, it is used to select between the solenoid or a 12V light mounted on the front of the robot. This feature allows the marking system to be on during indoor testing, but rather than painting over a mine (or indoor flooring), the light turns on.

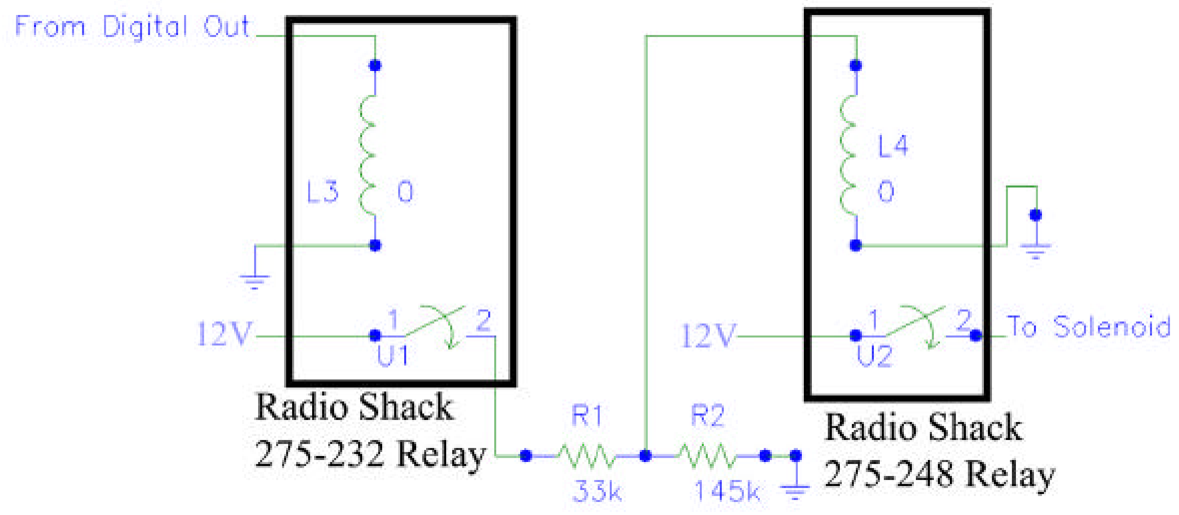


Figure 41 – Spray Paint Control Circuit

Power System

The different required component voltages and currents for Little Ranger's systems are listed in Table 2.

Table 2 – Voltage and Current Requirements for Various Components

Component	Voltage	Max Current	Batteries
68HC11 EVBU with ME11	12V	300 mA	8-AA
Schiebel AN-19/2	6V	140 mA	4-D
Spray Paint Solenoid	12V	5A	2-6V Motorcycle
Spray Paint Control Circuit	12V	400 mA	2-6V Motorcycle
333 oz-in Servos	6V/Servo	4.5A/Servo	2-6V Motorcycle

Power for the systems was provided through the use of the batteries indicated in Table 2. Each component's source was wired to a switch for easy power control and fuses were used where appropriate to provide protection.

The power supplies for the servos and spray paint solenoid were interconnected as shown in Figure 42. This connection was necessary to reduce the number of batteries required, by providing the current of two batteries for the servos and the voltage of two batteries for the spray paint solenoid, but only using three batteries to do so. Two of the batteries were wired in parallel, which allowed the same current to be available to all servos; two batteries were needed for this purpose because all servos running at max current utilized approximately 30A, which would quickly drain a single battery. The power for the spray paint solenoid utilizes one of the batteries from the servo supply, which was wired in series with a third battery, to produce the required 12V for the solenoid.

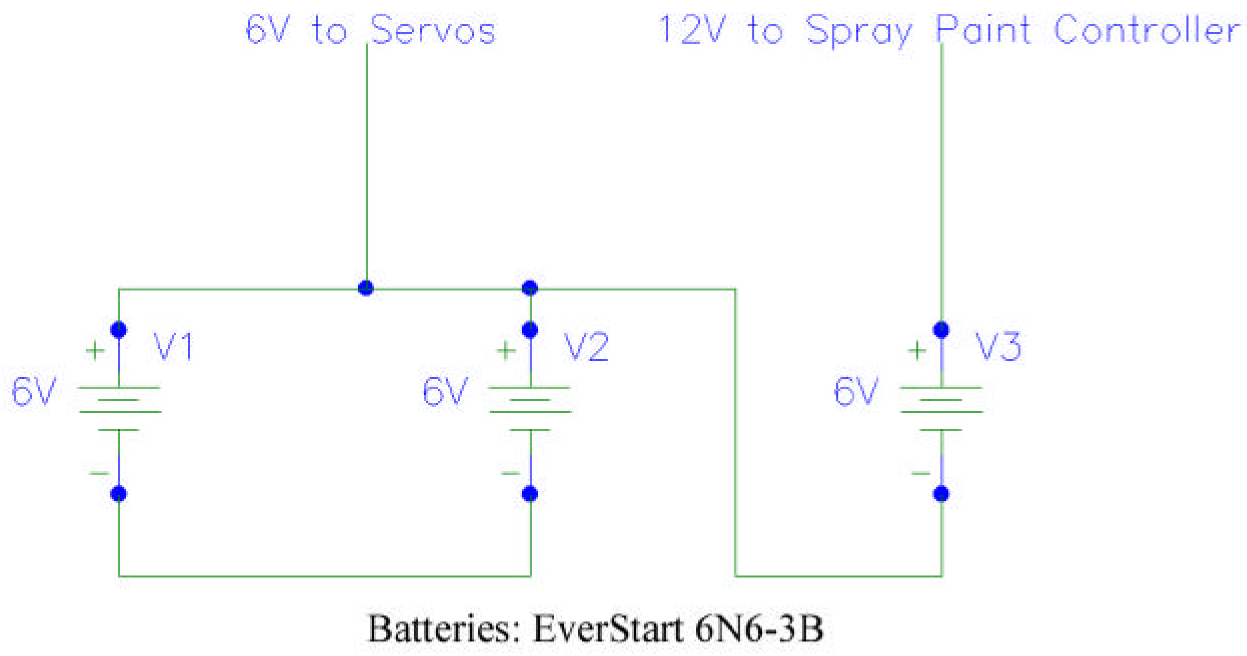


Figure 42 – Battery Organization for Servos and Marking System

All supplies were wired to a common ground, which causes some noise problems between components. In particular the mine detectors electronics unit experienced major noise problems, resulting in continuous false readings. The problem was resolved by placing a clamp-on ferrite bead on the cable connecting the search head to the electronics unit; toroids and a bypass capacitor were also placed on the mine detector’s supply lines.

Behaviors

Little Ranger has very limited memory, only 32KB, most of which is used by Interactive C and the required communications program used to connect the board to a personal computer. A large amount of memory was also required for control routines, such as those needed to control three servos. As a result of the tight memory restrictions, many behaviors were minimized. Unfortunately, I was not aware of the limitations IC placed on the 32KB prior to writing much of the code, so in Appendix A is the reduced code used on Little Ranger during Demo Day, and in Appendix B are some of the “full” routines prior to reducing their size. Overall behavior interactions can be seen in Figure 43.

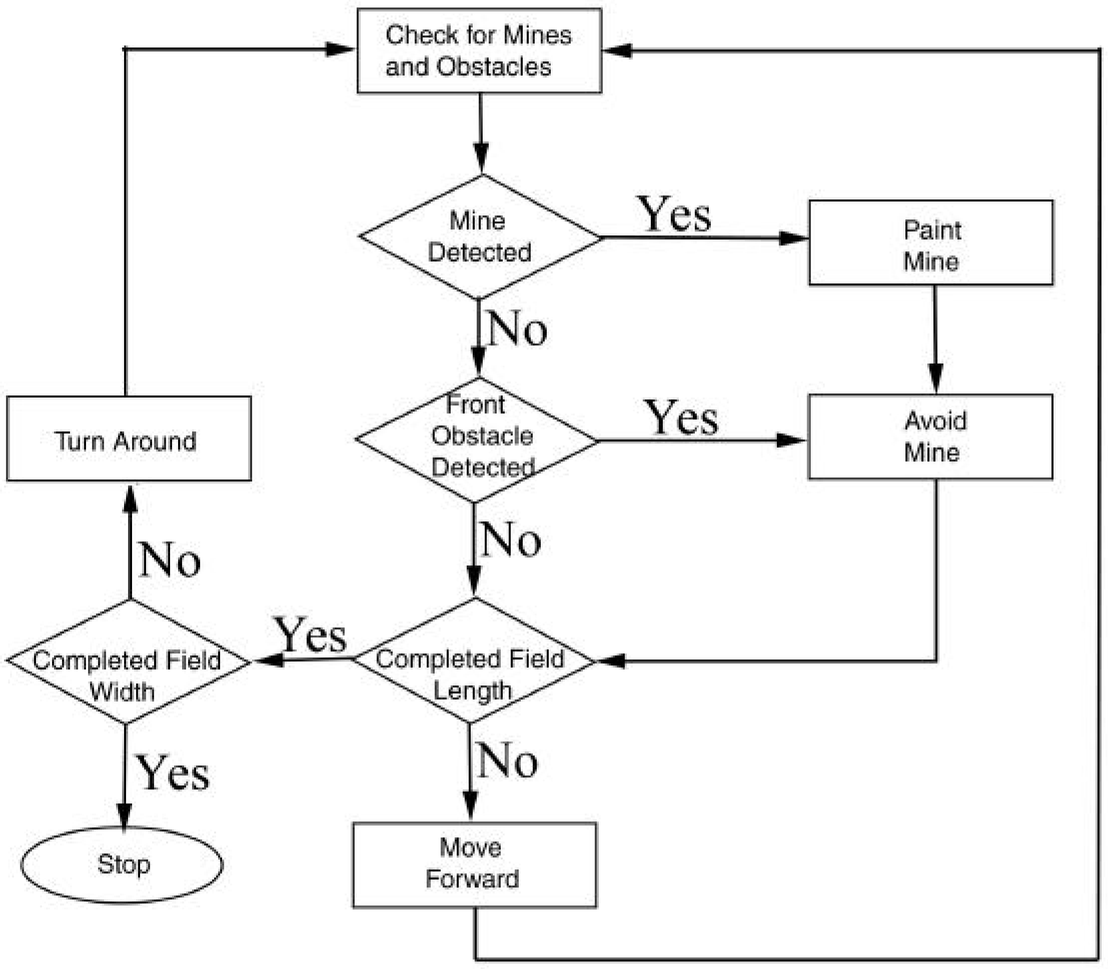


Figure 43 – Behavior Overview

<u>Search</u>

This behavior is used while searching for land mines. The search head is moved back and fourth, while the mine detector is searching for mines. During the same sweep, the bump sensors located on the search head are constantly checked. If an object is found to be obstructing the sweep, the search head stops moving in that direction and the angle of the obstacle is noted. In order to protect the search head, the newly noted angle becomes the new minimum (if the object is to the left of center), or the new maximum (if the object is to the right of center) angle. While Little Ranger is still in the same physical position, the sweep arm will not move beyond the new minimum or maximum angles. Once a mine is located, its angle is stored in an array for further processing and painting.

<u>Paint Mines</u>

Prior to painting the mines, the array containing the angles of detected mines generated by the search routine are processed. During processing it is determined in which of three sections the mines are located (refer to Figure 44 for section locations). The processing is necessary because Little Ranger's physical restrictions prevent it from turning to an exact angle. Therefore, it was necessary to paint roughly in the mine's location and utilize the coverage area of the spray paint to mark over the mine.

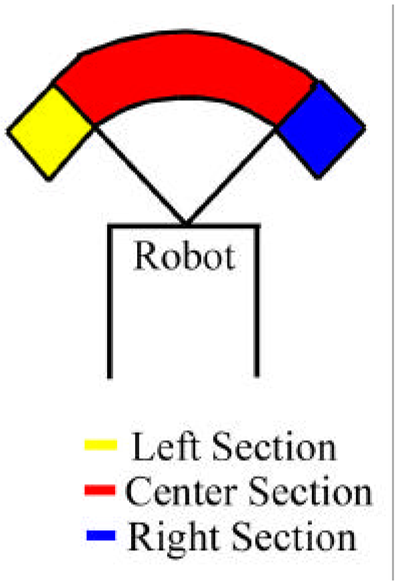


Figure 44 – Section Locations

After the processing of the angles, Little Ranger turns in the direction of the section containing the mine. Once pointed in the correct direction, it moves forward placing the spray paint can over the mine, and paints. After the painting is complete, it reverses the above procedure, and returns to its original location.

<u>Avoid Mines</u>

After a mine has been detected and painted, or an object is detected in front, Little Ranger must maneuver around it. This routine is one that was affected by the memory restrictions; therefore it assumes that the mines are not placed caddy-corner to each other.

If there are no mines located in the center section, the robot can continue moving forward since it will not run over the mine. This is because the center region is setup to be wider than the width of the robot, while the other two regions fall on their respective sides outside of the center region (refer to Figure 44).

If there is a mine in the center region, the robot starts by making a 90-degree turn to the right (start of phase 2 in Figure 45). Once the turn is complete, it moves forward approximately a distance a little greater than the width of the robot. Then, it turns to the left and checks for mines, if there is still a mine in the center section, the above mentioned procedure is repeated until there are no mines found at the center region.

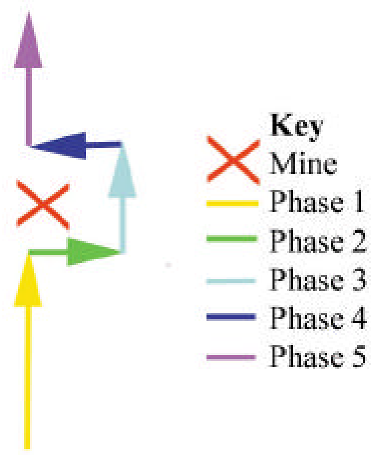


Figure 45 – Mine Avoidance Steps

Once Little Ranger locates the positions at which there are no mines in the center section (start of phase 3 in Figure 45) it moves forward while checking the left side until no mines are found in the center section. When the location at which there are no mines to the left is found (start of phase 4 in Figure 45), the robot pulls forward and starts checking to its right side. After finding the position at which there are no mines in the center section, the mine avoidance routine is completed and the robot is at the start of phase 5 in Figure 45.

Obstacle Avoidance

Obstacle avoidance is another behavior that was affected by the memory restrictions. As a result of having only a few bytes left after implementing the above behaviors and other control routines, it was necessary to have Little Ranger simply stop if an obstacle on the side or in the rear prevented it from performing a behavior. Once the obstacle is cleared the switch on the back of little ranger only needs to be toggled down then up to restart the robot. When obstacles are found in front of the robot, the avoid mine routine is used to go around the obstacle.

Experimental Layout and Results

Little Ranger was extensively tested at the sub-system level, which greatly reduced the number of problems encountered during testing. Prior to each system being integrated into the robot, it was tested, corrections were made if needed, and it was tested again, until the system performed in a proper, reliable manner. After the integration of each system, the newly installed system and the other previously installed systems where tested. This was very useful in determining if systems were interacting in a negative fashion, and if so, which system was responsible for causing the problem. This was a tedious procedure to follow, but in the long run it saved time and brought about reliable systems.

As a result of the actuation limitations described in the actuation section, experiments were limited. On controlled surfaces, Little Ranger was capable of performing all required tasks using mines simulated through the use of software. The simulated mines were necessary because the controlled surfaces contained steel reinforcements, which caused continuous false mine detector readings. All test patterns listed in Figures 46 through 52 were used, and Little Ranger successfully completed all required tasks for each given mine pattern.

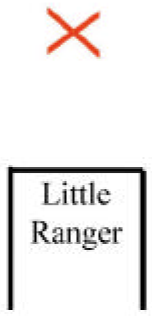


Figure 46 – Test Pattern 1

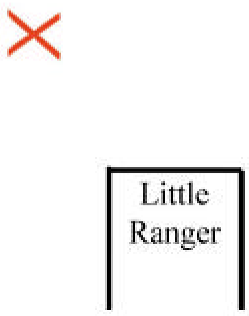


Figure 47 – Test Pattern 2

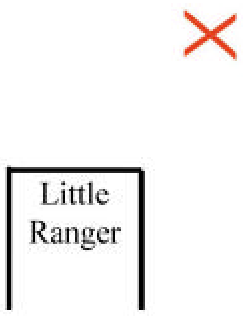


Figure 48 – Test Pattern 3

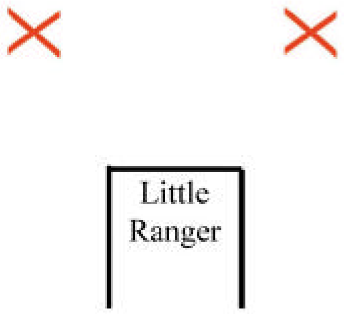


Figure 49 – Test Pattern 4

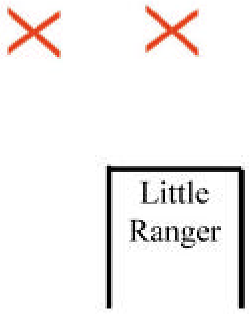


Figure 50 – Test Pattern 5

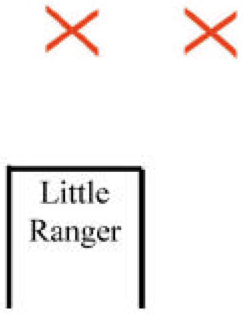


Figure 51 – Test Pattern 6

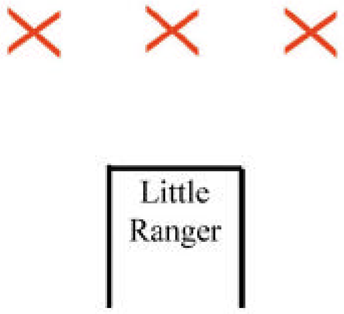


Figure 52 – Test Pattern 7

In “real-life” situations, Little Ranger successfully detected mines (steel washers) and executed the proper routines. However, as a result of its limited mobility on such surfaces, the wheels would attempt to move, but were unsuccessful. This resulted in difficulty tracking the robot’s actions while it attempted to paint multiple mines. For this reason, only the mine patterns in Figures 46 through 48 were used. Excluding the actuation limitation, Little Ranger was successful in completing all tasks.

Conclusion

During the course of this yearlong endeavor, I experienced, and resolved, many problems. Two major problems were encountered, the first being the memory restrictions mentioned in the behaviors section of this report. The second, being the overly optimistic specifications of the servos used for platform actuation. This resulted in major terrain limitations and difficulty in turning. During very controlled conditions (tile flooring), Little Ranger had no problems completing all required behaviors using software simulated mines. Unfortunately, in the case of all available controlled surfaces, there were steel reinforcements in the flooring, so the mine detector could not be used.

In outdoor "real-life" conditions, Little Ranger was capable of accurately detecting mines. But as a result of the actuation limitations, it was incapable of correctly moving on the surface, but it did attempt to perform the correct actions. The fact that mines were correctly found utilizing the mine detector and that the robot responded correctly, although its actions were impaired, indicates that all of Little Ranger's behaviors interact and execute properly. This is because the one component that was not used during the controlled testing, which was completely successful, was successfully used during outdoor testing. Therefore, bridging the gap between the simulated mines used in the controlled situation and mines detected by the AN-19/2. More extensive “real-life” testing is definitely required once the actuation problem is resolved.

Considering the discussed limitations, this project accomplished its primary goal of proving that this is a sound concept with realistic objectives. Therefore, successfully completing Phase 1 and opening the door for future work on the project.

Documentation

References

[1] Invention Finds Buried Land Mines, Gainesville Sun, May 22, 1997
http://sun-robot.nuceng.ufl.edu/land-mine/news.htm, 1/10/98

[2] CARE
http://www.care.org/newscenter/landmines/landmine.html, 1/14/98

[3] Dr. Edward Dugan, Nuclear Engineering, University of Florida
Interviewed 1/22/98

Part Sources

Schiebel AN-19/2 (DOD designation AN/PSS-12)

U.S. (Branch Office)
Schiebel Technology, Inc.
Suite 804
2127 California Street, N.W.
Washington, D.C. 20008
Telephone +1 (202) 483-8311
Facsimile +1 (202) 483-8316
Contact: Mr. Robert Fitz Carty, e-mail: roca@schiebel.com

AUSTRIA (Main Offices)
Schiebel Elektronische Geraete GmbH
Margaretenstrasse 112
A - 1050 Vienna, Austria
Telephone +43 - 1 - 546 26-0
Fax +43 - 1 - 545 2339
Contact: Mr. Hans Georg Schiebel, e-mail: gesc@schiebel.com

Jumbo CS-600 333 oz-in Servo
Hobby Shack
18480 Bandilier Circle
Fountain Valley, CA 92728
1-800-854-8471
Part Number: 444281

ME11 expansion board for the 68HC11 EVBU
Mekatronix
316 NW 17th Street, Suite A
Gainesville, FL 32603
407-672-6780

Appendix A – Demo Day Code

3 Servo Assembly file before being converted to ICB
3servo.asm

```
* Final Modifications by: Daniel Kucik
*                   Addition of 3rd servo
* File: 3servo.asm
*
* This file is designed to be compiled using interactive c's
* asm -> icb compiler.  3servo.icb must be loaded to the 68HC11 before
* 3servo.c or Little Ranger's main file.
* The combination of 3servo.icb and 3servo.c are used to provide control
* over 3 servos attached to OC1, OC2, OC3.
* See 3servo.c for usage
*
* NOTE: For the 3servo functions to work, lib_rw11.c must be modified
*       so that all lines implementing the motor( x,y) command are
*       removed.  For additional notes please see 3servo.c comments
*
* HISTORY from previous version
* icb file:  "twoservo.asm"
*
*   Anne Wright and PK Oberoi
*   Sat Jan 12 05:12:13 1992
*   Sun Jan 13 03:31:55 1992 added sweeping capabilities --  anarch
*   Mon May 31          1993 added second servo channel -- anarch
*       March 10, 1998       added third servo channel -- Dan Kucik

BASE    EQU     $1000
PORTA   EQU     $00   ; Port A data register
CFORC   EQU     $0B   ; Timer Compare Force Register
OC1M    EQU     $0C   ; Output Compare 1 Mask register
OC1D    EQU     $0D   ; Output Compare 1 Data register

* Two-Byte Registers (High,Low -- Use Load & Store Double to access)
TCNT    EQU     $0E   ; Timer Count Register
TOC1    EQU     $16   ; Timer Output Compare register 1
TOC2    EQU     $18   ; Timer Output Compare register 2
TOC3    EQU     $1A   ; Timer Output Compare register 3
TOC4    EQU     $1C   ; Timer Output Compare register 4
TI4O5   EQU     $1E   ; Timer Input compare 4 or Output compare 5 register

TCTL1   EQU     $20   ; Timer Control register 1
TCTL2   EQU     $21   ; Timer Control register 2
TMSK1   EQU     $22   ; main Timer interrupt Mask register 1
TFLG1   EQU     $23   ; main Timer interrupt Flag register 1
TMSK2   EQU     $24   ;  misc Timer interrupt Mask register 2
TFLG2   EQU     $25   ;  misc Timer interrupt Flag register 2
PACTL   EQU     $26   ; Pulse Accumulator Control register
PACNT   EQU     $27   ; Pulse Accumulator Count register
OPTION  EQU     $39   ; system configuration Options
HPRIO   EQU     $3C   ; Highest Priority Interrupt and misc.
INIT    EQU     $3D   ; RAM and I/O Mapping Register
```

```
* Interrupt Vector locations
TOINT   EQU     $DE     ; Timer Overflow
TOC5INT EQU     $E0     ; Timer Output Compare 5
TOC4INT EQU     $E2     ; Timer Output Compare 4
TOC3INT EQU     $E4     ; Timer Output Compare 3
TOC2INT EQU     $E6     ; Timer Output Compare 2
TOC1INT EQU     $E8     ; Timer Output Compare 1
RTIINT  EQU     $F0     ; Real Time Interrupt
*************************************************************************
*                           End of register equ's                       *
*************************************************************************



* program equates
REPEAT_PERIOD   EQU     20              /* number of interrupts between
pulses */

      ORG     MAIN_START

* variables

/* C variables */
variable_servo_pulse_wavetime1  FDB     3000
variable_servo_pulse_wavetime2  FDB     3000
variable_servo_pulse_wavetime3  FDB     3000
variable_servo_enable           FDB     0

/* internal variable */
servo_count                     FCB     0

subroutine_initialize_module:
*************************************************************************
*                          Start of LDXIBASE.asm                        *
* Included in this file to resolve an assembler linking problem         *
*************************************************************************

* File "ldxibase.asm"
*
* Fred Martin Thu Oct 10 19:49:38 1991
*
*
* The following code loads the X register with a base pointer to
* the 6811 interrupt vectors:  $FF00 if the 6811 is in normal mode,
* and $BF00 if the 6811 is in special mode.
*
* The file "6811regs.asm" must be loaded first for this to work.
*
*

      LDAA    HPRIO
      ANDA    #$40                  ; test SMOD bit
      BNE     *+7
      LDX     #$FF00                ; normal mode interrupts
      BRA     *+5
      LDX     #$BF00                ; special mode interrupts
```

```
************************************************************************
*                           End of ldxibase.asm                        *
************************************************************************

* X now has base pointer to interrupt vectors ($FF00 or $BF00)

* get current vector; poke such that when we finish, we go there
      LDD     TOC4INT,X              ; SystemInt on TOC4
      STD     interrupt_code_exit+1

* setup interrupt service routine as interrupt_code_start for OC4
      LDD     #interrupt_code_start
      STD     TOC4INT,X

      LDX     #BASE

* set up A7 for output
      BSET    PACTL,X $80            *Set A7 as output
      BSET    OC1M,X $80             *set OC1 to control A7

* set up A3 for output
      BSET    TCTL1,X $2             * \
      BCLR    TCTL1,X $1             * --clear OC5 output line to zero

* setup A5 for output
      BSET    TCTL1,X %00100000      * \
      BCLR    TCTL1,X %00010000      * --clear OC3 output line to zero


*initialize Servo Parameters
      LDD     #3000
      STD     variable_servo_pulse_wavetime1  /* Initialize  servo 1
      LDD     #3000
      STD     variable_servo_pulse_wavetime2  /* Initialize  servo 2
                                    period to 2000 */
      LDD     #3000
      STD     variable_servo_pulse_wavetime3  /* Initialize  servo 1

* fall through to servo_off subroutine

/*********************************************************************/
subroutine_asm_servo_off:
      LDX     #BASE

      LDD     #0
      STD     variable_servo_enable   /* disable servo */
      BCLR    OC1D,X $80             /* set OC1 to clear on match */
      BSET    TCTL1,X $02            /* set OC5 to clear on match */
      BSET    TCTL1,X %00100000      /* set OC3 to clear on match */
      BCLR    TCTL1,X %00010000      /* set OC3 to clear on match */

      RTS

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

```
/*************************************************************************/
subroutine_asm_servo_on:
      LDX     #BASE
      LDD     #1                     /* load 1 into D */
      STD     variable_servo_enable   /* enable servo */

      LDAA    #REPEAT_PERIOD          /* Reset Cycle Counter */
      STAA    servo_count

      RTS

interrupt_code_start:
      LDD     variable_servo_enable   /* check if servo is on */
      BEQ     interrupt_code_exit

      LDAA    servo_count             /* check if at beginning of cycle */
      BEQ     SetPulse

      ADDA    #-1                     /*  No: decrement cycle counter */
      STAA    servo_count

      BRA     interrupt_code_exit

SetPulse:
      LDX     #BASE

* set up for servo 1
      LDD     TCNT,X                  /* Get current timer counts */
      ADDD    variable_servo_pulse_wavetime1  /* Add the servo period and */
      STD     TOC1,X                    /* Place in TOC1 */
      LDD     TCNT,X                  /* Get current timer counts */
* for servo 3
      ADDD    variable_servo_pulse_wavetime3  /* Add the servo period and */
      STD     TOC3,X                    /* Place in TOC3 */

        BSET    TCTL1,X $30             *--setup so that sets on next match
      BSET    OC1D,X $80              /* set OC1 to set on match */
        BSET    CFORC,X $A0             /* force a match now */
      BCLR    OC1D,X $80              /* set OC1 to clear on match */
      BCLR    TCTL1,X $10             * clear on next match
      BCLR    TFLG1,X $20             * clear oc3 flag



* set up for servo 2
      BCLR    TCTL1,X $02             /* disconnect interrupt from pin */
      BSET    PORTA,X $08             /* set the pin */
      BSET    TCTL1,X $02             /* set interrupt to reset pin */

      LDD     TCNT,X                  /* Get current timer counts */
      ADDD    variable_servo_pulse_wavetime2  /* Add the servo period and */
      STD     TI4O5,X                 /* Place in TOC1 */
      BCLR    TFLG1,X $08

      LDAA    #REPEAT_PERIOD          /* Reset Cycle Counter */
      STAA    servo_count
```

```
interrupt_code_exit:
      JMP     $0000   ; this value poked in by init routine
```

Interactive C code used on Demo Day
DDay.c

```
/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                         IC Code for Little Ranger                   */
/*                                                                     */
/* Written by: Dan Kucik                                               */
/* Written for: EEL 5666 IMDL @ the University of Florida              */
/* Robot Name: Little Ranger                                           */
/* Teacher: Dr. Arroyo                                                 */
/* Compiler: Interactive C                                             */
/*                                                                     */
/* Description:  The code contained in this file is designed to run on  */
/*               Motorola's 68HC11 EVBU board with analog input and     */
/*               digital output expansions described in the report.     */
/*                                                                     */
/*               This code implements the routines needed to access the */
/*               expanded ports, servo control, and behaviors.         */
/*                                                                     */
/* Note: l_ranger.icb must be loaded prior to loading this file.       */
/*       For the servo routines to work properly lib_rw11.c must be     */
/*       modified so that all references to the "motor" command are     */
/*       removed.                                                      */
/*                                                                     */
/* Version: 3.0                                                        */
/*                                                                     */
/* Modifications: 11/28: Reduced Size                                  */
/*                lost most of obstacle avoidance (needed the memory    */
/*                12/3: Quick fix for sweep arm bump sensors -- false   */
/*                readings resulting from grass being hit              */
/*                look at start of check_sh_sensors()                   */
/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */

/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                   START OF VARIABLE DEFINITIONS                     */
/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
int ird_far_object=92;              /* defines minimum value for object */
                                    /* detected on IR detectors        */
int sh_temp=0;                      /* temp variable used by search     */
                                    /* head sensor check routines       */
int cur_found_obj=0;                /* temp used to store the current   */
                                    /* status of sensor checks         */
int sh_temp2=0;                     /* Temp  var used to hold sensor    */
                                    /* status                          */
int shb_min_value=5;                /* minimum value needed for search  */
                                    /* head bump sensors               */
float md_sensor_speed=10.0;         /* movement increment for the       */
                                    /* search head while checking for   */
                                    /* mines                           */
float md_min_angle=10.0;            /* the minimum angle allowed for    */
                                    /* for the mine detector arm       */
float md_max_angle=170.0;           /* the maximum angle allowed for    */
                                    /* for the sweep arm               */
float md_cur_pos=90.0;              /* current position of the search   */
                                    /* head                            */
float md_angle_obs[2];              /* array to hold the search arm's   */
                                    /* positions for when obstacles     */
                                    /* are found                        */
float wait1=0.1;                   /* delay time for when the search   */
                                    /* head is moved with an increment  */
                                    /* md_sensor_speed                  */
int sh_temp3=0;                    /* temp variable used while looking */
                                    /* for mines                        */
int sh_temp4=0;                    /* holds current search head bump   */
                                    /* status                           */
int md_port=7;                     /* analog port for the detector     */
int mine_min_value = 30;           /* minimum analog value for a found */
                                    /* mine                             */
float angle_of_mines[36];          /* angles of found mines            */
int num_of_mines=0;                /* number mines found               */
float cur_max_angle=160.0;         /* current maximum angle --         */
                                    /* accounting for obstacles         */
float cur_min_angle=10.0;          /* current minimum angle for the    */
                                    /* search head -- considering       */
                                    /* found obstacles                  */
float temp_angle;                   /* temp angle storage               */
int temp_cnt;                      /* position counter                 */
float angle_of_mines2[3];          /* simplified angle of mines        */

/* Start of servo actuation angles */
float right_off=88.0;              /* angle to turn off right servo    */
float left_off=91.0;               /* angle to turn off left servo     */
float right_move_forward=180.0;    /* angle for right servo to move    */
                                    /* forward                          */
float left_move_forward=30.0;      /* angle for left servo to move     */
                                    /* forward                          */
float left_reverse=150.0;           /* angle for left servo to reverse  */
float right_reverse=0.0;            /* angle for right servo to reverse */
float left_turn_left=150.0;        /* angle for left servo while        */
                                    /* turning left                     */
float right_turn_left=180.0;       /* angle for right servo while      */
                                    /* turning left                     */
float left_turn_right=30.0;        /* angle for left servo while       */
                                    /* turning right                    */
float right_turn_right=0.0;        /* angle for right servo while      */
                                   /*  turning right                    */

/* end of servo actuation angles */

/* start of timing variables */
float time_for_90degrees=0.6;      /* time to turn 90 degrees while    */
                                    /* going around mines               */
float time_turn_paint=1.0;          /* time needed to turn right to     */
                                   /*  to point to mine on right        */
float time_forward_paint=1.0;      /* time to pull forward to mine     */
float time_to_paint=3.0;           /* length of time to paint           */
float time_to_move_forward=0.15;   /* time to pull forward             */
int time_to_pass_mine=6;            /* time to move forward while        */
                                    /* avoiding a mine before checking  */
                                    /* for clear left                    */
int move_cnt;                      /* loop cnt used by move_forward     */
int move_sensor_temp;              /* holds the current bump status    */
int turn_cnt;                      /* loop counter for turning around  */
int shb_tol=3;                      /* tolerance for search head bump    */
int rb_tol=3;                      /* tolerance for rear bump          */



/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                 END OF VARIABLE DEFINITIONS                         */
/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */




/**********************************************************************/
/*                                                                    */
/* File: 3servo.c                                                     */
/* Final Revisions by: Dan Kucik                                      */
/*                                                                    */
/* This file is designed to work in conjunction with 3servo.icb, which  */
/* must be loaded before 3servo.c.  The combination of the 3servo files */
/* will provide control over 3 servos on OC1, OC3, and OC5.           */
/*                                                                    */
/* Usage: servo_on(); turns on servo control                          */
/*        servo_degx(y); controls servo x and rotates it to angle y   */
/*                       y must be a floating value (00.0)            */
/*        servo_radx(y); controls servo x and rotates it to rad angle y */
/*                       y is a float                                 */
/*        servo_off(); turns off servo control                        */
/*                                                                    */
/* History:                                                           */
/* "twoservo.c" for the 6.270 'bot                                    */
/*      servo support code for two servos                             */
/*      second servo hooked up in place of the beeper                 */
/*      must be used with "NOBEEP" pcode                              */
/* by Anne Wright                                                     */
/* Sat Jan 11, 1993                                                   */
/* March 10, 1998  3rd servo control added by Dan Kucik               */
/*                                                                    */
/* NOTE: For the 3servo functions to work, lib_rw11.c must be modified  */
/*       so that all lines implementing the motor( x,y) command are    */
/*       removed.  lib_rw11.c is located in the \ic\libs directory and  */
/*       is automatically loaded by ic.                               */
/**********************************************************************/
/* these were chosen experimentally...*/
int MIN_SERVO_WAVETIME = 1400;
int MAX_SERVO_WAVETIME = 4680;
int SERVO_RANGE = (MAX_SERVO_WAVETIME-MIN_SERVO_WAVETIME);

float rexcursion = 3.14159;
float dexcursion = 180.0;
/**********************************************************************/
/*                        servo_on                                    */
/*                                                                    */
/* This routine turns on the servos.  It must be called prior to      */
/* running the servos.                                                */
/* It makes a call to the 3servo.icb file to setup the pulse width    */
/* modulation.                                                        */
/*                                                                    */
/**********************************************************************/
```

```
void servo_on()
{
  asm_servo_on(0);
}
/**********************************************************************/
/*                              servo_off                              */
/*                                                                     */
/* This routine turns off the servos.                                  */
/*                                                                     */
/* It makes a call to the 3servo.icb file to turn off pulse width      */
/* modulation.                                                         */
/*                                                                    */
/**********************************************************************/
void servo_off()
{
  asm_servo_off(0);
}
/**********************************************************************/
/* Servo movement commands                                            */
/**********************************************************************/
/**********************************************************************/
/*                      servo1                                         */
/*                                                                     */
/* This routine is used to set the pulse width for servo 1 OC1         */
/*                                                                     */
/* It sets the variable wavetime1 which is used by the code in         */
/* 3servo.icb to set the pulse width set by the number of clock cycles  */
/*                                                                    */
/**********************************************************************/
int servo1(int period)                  /* argument in clock cycles of  */
                                    /* pulse, moves servo
*/
{
  if(period>MAX_SERVO_WAVETIME)
    return (servo_pulse_wavetime1=MAX_SERVO_WAVETIME);
  else if(period<MIN_SERVO_WAVETIME)
    return (servo_pulse_wavetime1=MIN_SERVO_WAVETIME);
  else
    return(servo_pulse_wavetime1=period);
}

/**********************************************************************/
/*                      servo_rad1                                     */
/*                                                                     */
/* This routine is used to set the pulse width for servo 1 OC1         */
/*                                                                     */
/* It sets the variable wavetime1 which is used by the code in         */
/* 3servo.icb to set the pulse width                                   */
/* To use the routine the angle must be entered in  rads               */
/*    Usage: servo_rad1(angle_in_rads)                                 */
/*          servo_on() must be run before the servos will operate      */
/*                                                                    */
/**********************************************************************/
int servo_rad1(float angle)             /* argument in radians, moves servo
*/
{
  return servo1(radian_to_pulse(angle));
```

```
}


/************************************************************************/
/*                        servo_deg1                                   */
/*                                                                      */
/* This routine is used to set the pulse width for servo 1 OC1          */
/*                                                                      */
/* It sets the variable wavetime1 which is used by the code in          */
/* 3servo.icb to set the pulse width                                    */
/* To use the routine the angle must be entered in degrees              */
/*    Usage: servo_rad1(angle_in_degrees)                               */
/*          servo_on() must be run before the servos will operate       */
/*                                                                     */
/************************************************************************/
int servo_deg1(float angle)            /* argument in degrees, moves servo
*/
{
  return servo1(degree_to_pulse(angle));
}

/************************************************************************/
/*                        servo2                                       */
/*                                                                      */
/* This routine is used to set the pulse width for servo 2 OC5          */
/*                                                                      */
/* It sets the variable wavetime2 which is used by the code in          */
/* 3servo.icb to set the pulse width set by the number of clock cycles   */
/*                                                                     */
/************************************************************************/
int servo2(int period)                 /* argument in clock cycles of  */
                                   /* pulse, moves servo
*/
{
  if(period>MAX_SERVO_WAVETIME)
    return (servo_pulse_wavetime2=MAX_SERVO_WAVETIME);
  else if(period<MIN_SERVO_WAVETIME)
    return (servo_pulse_wavetime2=MIN_SERVO_WAVETIME);
  else
    return(servo_pulse_wavetime2=period);
}

/************************************************************************/
/*                        servo_rad2                                   */
/*                                                                      */
/* This routine is used to set the pulse width for servo 1 OC5          */
/*                                                                      */
/* It sets the variable wavetime2 which is used by the code in          */
/* 3servo.icb to set the pulse width                                    */
/* To use the routine the angle must be entered in  rads                */
/*    Usage: servo_rad2(angle_in_rads)                                  */
/*          servo_on() must be run before the servos will operate       */
/*                                                                     */
/************************************************************************/
int servo_rad2(float angle)            /* argument in radians, moves servo
*/
{
  return servo2(radian_to_pulse(angle));
}

/*************************************************************************/
/*                        servo_deg2                                      */
/*                                                                         */
/* This routine is used to set the pulse width for servo 2 OC5             */
/*                                                                         */
/* It sets the variable wavetime2 which is used by the code in             */
/* 3servo.icb to set the pulse width                                       */
/* To use the routine the angle must be entered in degrees                 */
/*    Usage: servo_rad2(angle_in_degrees)                                  */
/*          servo_on() must be run before the servos will operate          */
/*                                                                        */
/*************************************************************************/
int servo_deg2(float angle)            /* argument in degrees, moves servo
*/
{
  return servo2(degree_to_pulse(angle));
}


/*************************************************************************/
/*                        servo3                                          */
/*                                                                         */
/* This routine is used to set the pulse width for servo 3 OC3             */
/*                                                                         */
/* It sets the variable wavetime3 which is used by the code in             */
/* 3servo.icb to set the pulse width set by the number of clock cycles     */
/*                                                                        */
/*************************************************************************/
int servo3(int period)                 /* argument in clock cycles of  */
                                    /* pulse, moves servo
*/
{
  if(period>MAX_SERVO_WAVETIME)
    return (servo_pulse_wavetime3=MAX_SERVO_WAVETIME);
  else if(period<MIN_SERVO_WAVETIME)
    return (servo_pulse_wavetime3=MIN_SERVO_WAVETIME);
  else
    return(servo_pulse_wavetime3=period);
}


/*************************************************************************/
/*                        servo_rad3                                      */
/*                                                                         */
/* This routine is used to set the pulse width for servo 3 OC3             */
/*                                                                         */
/* It sets the variable wavetime3 which is used by the code in             */
/* 3servo.icb to set the pulse width                                       */
/* To use the routine the angle must be entered in rads                    */
/*    Usage: servo_rad3(angle_in_rads)                                     */
/*          servo_on() must be run before the servos will operate          */
/*                                                                        */
/*************************************************************************/
```

```
int servo_rad3(float angle)             /* argument in radians, moves servo
*/
{
  return servo3(radian_to_pulse(angle));
}

/*************************************************************************/
/*                        servo_deg3                                     */
/*                                                                       */
/* This routine is used to set the pulse width for servo 3 OC3           */
/*                                                                       */
/* It sets the variable wavetime3 which is used by the code in           */
/* 3servo.icb to set the pulse width                                     */
/* To use the routine the angle must be entered in degrees               */
/*    Usage: servo_rad3(angle_in_degrees)                                */
/*          servo_on() must be run before the servos will operate        */
/*                                                                       */
/*************************************************************************/
int servo_deg3(float angle)             /* argument in degrees, moves servo
*/
{
  return servo3(degree_to_pulse(angle));
}


/*************************************************************************/
/* Pulse width calculations                                              */
/*************************************************************************/

/*************************************************************************/
/*                        radian_to_pulse                                */
/*                                                                       */
/* This routine is used to convert radian angles to the required         */
/* pulse width for the given angle.                                      */
/*                                                                       */
/*************************************************************************/
int radian_to_pulse(float angle)        /* argument in radians, returns
*/
                                    /* pulse width
*/
{
  return ((int)(angle*((float)SERVO_RANGE)/rexcursion)+MIN_SERVO_WAVETIME);
}


/*************************************************************************/
/*                        radian_to_pulse                                */
/*                                                                       */
/* This routine is used to convert angles in degrees to the required     */
/* pulse width for the given angle.                                      */
/*                                                                       */
/*************************************************************************/
int degree_to_pulse(float angle)        /* argument in degrees, returns
*/
                                    /* pulse width
*/
{
  return ((int)((angle*((float)SERVO_RANGE))/dexcursion)+MIN_SERVO_WAVETIME);
}

/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                   END OF 3servo routines                            */
/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */




/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                            Analog Access Routines                   */
/*                                                                     */
/* Written by: Daniel Kucik                                            */
/* Original Creation: 3/22/98                                          */
/* Version: 2.0 (reduced size)                                         */
/* Modifications: reduced size                                         */
/*                                                                     */
/* The following routines are used to access the multiplexed analog    */
/* inputs and the remaining on-board (EVBU) analog ports.              */
/*                                                                     */
/* For use on the Motorola 68HC11 EVBU board with the analog expansion */
/* described on page 17 of the Talrik Assembly Manual (using $4000 (pin */
/* 53 on header P4 of EVBU -- brought back to header by ME11 board) as */
/* the clock for the 74HC574 (latch)).                                 */
/*                                                                     */
/* Use: analog?();  -- accesses MUX? described in the Talrik MRSX01     */
/*      circuit diagram.                                               */
/*      analog24() through analog29() access the remaining analog ports */
/*      that are available on the EVBU after the expansion.            */
/* Modified: Routine names now reflect the names of connected devices  */
/*           for Little Ranger.                                        */
/*           11/28/98: removed extra port access to reduce size        */
/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */

/* Start of routines for MUX 1 (chip U1 in diagram) (Analogs 0-7) */
int rr_ird()                        /* rear right IR detector           */
{
poke(0x4000,0b00010000);
return analog(1);
}
int rc_ird()                        /* rear center IR detector          */
{
poke(0x4000,0b00010001);
return analog(1);
}
int rl_ird()                        /* rear left IR detector            */
{
poke(0x4000,0b00010010);
return analog(1);
}
int l1_ird()                        /* left front IR detector           */
{
poke(0x4000,0b00010011);
return analog(1);
}
int l2_ird()                        /* IR detector between front and    */
```

```
                                    /* center wheels on left side       */
{
poke(0x4000,0b00010100);
return analog(1);
}
int l3_ird()                        /* IR detector between center and   */
                                    /* rear wheel on left side          */
{
poke(0x4000,0b00010101);
return analog(1);
}
int l4_ird()                        /* left rear IR detector            */
{
poke(0x4000,0b00010110);
return analog(1);
}
int r1_ird()                        /* right front IR detector          */
{
poke(0x4000,0b00010111);
return analog(1);
}
/* Start of routines for MUX 2 (chip U3 in diagram) (Analogs 8-15) */
int r2_ird()                        /* IR detector between front and    */
                                    /* center wheels on right side      */
{
poke(0x4000,0b00001000);
return analog(2);
}
int r3_ird()                        /* IR detector between center and   */
                                    /* rear wheel on right side         */
{
poke(0x4000,0b00001001);
return analog(2);
}
int r4_ird()                        /* Right rear IR detector           */
{
poke(0x4000,0b00001100);
return analog(2);
}
int run_switch()                    /* switch on back                   */
{
poke(0x4000,0b00001110);
return analog(2);
}
int MD_set_switch()                 /* switch on control panel for      */
                                    /* use while setting sensitivity    */
{
poke(0x4000,0b00001111);
return analog(2);
}
/* Start of routines for MUX 3 (chip U4 in diagram) (Analogs 16-23) */
int rear_bump()                     /* Rear bump switches               */
{
poke(0x4000,0b00010000);
return analog(3);
}
int sh_l_bump()                     /* search head bump on left         */
```

```
{
poke(0x4000,0b00010001);
return analog(3);
}
int sh_r_bump()                    /* search head bump on right        */
{
poke(0x4000,0b00010010);
return analog(3);
}
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                     End of Analog Access Routines                    */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */



/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                      check_left_sensors                              */
/*                                                                      */
/* Written by: Daniel Kucik                                             */
/* Original Creation: 3/22/98                                           */
/* Version: 2.0 (reduced size)                                          */
/*                                                                      */
/* Description: This routine checks the left side IR sensors for       */
/*              obstacles.  It returns the location and distance (close */
/*              or far) of the obstacle.                                */
/*                                                                      */
/* Returns:  0=no objects detected                                      */
/*           1=object found at left front                               */
/*           2=object found at left center                              */
/*           3=object found at left rear                                */
/*           4=object found spanning multiple sections                  */
/*                                                                      */
/* Version: 2.0                                                         */
/* Revisions: 11/28/98: reduced options to bumped object and I/R object */
/*                      detected to reduce size                         */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
int check_left_sensors()
{
      cur_found_obj=0;              /* holds the current status         */
      poke(0x7000,0b10000000);      /* turn on IR LED's                 */
      sleep(0.01);                  /* delay while turned on            */

/* checking L1 */
      if(l1_ird()>ird_far_object)   /* check if object is in range      */
      {
            cur_found_obj=1;        /* set flag if object detected      */
      }

/* Checking L2 */
      if(l2_ird()>ird_far_object)   /* check if object is in range      */
      {
            if(cur_found_obj!=1)
                                    /* if no object at front            */
            {
                  cur_found_obj=2;  /* then object at center left       */
            }
            else                    /* object was found at front        */
            {
                  return 4;         /* object spans multiple sections   */
            }
      }
/* Checking L3 */
      if(l3_ird()>ird_far_object)   /* check if object at L3             */
      {
            if(cur_found_obj!=1)
                                    /* if object was not detected at    */
                                    /* front left                       */
            {
                  cur_found_obj=2;  /* object at left center            */
            }
            else                    /* object was found at front        */
            {
                  return 4;         /* object spans multiple sections   */
            }

      }

/* Checking L4 */
      if(l4_ird()>ird_far_object)   /* if object is in range            */
      {
            if(cur_found_obj==0)
                                    /* if have not detected other objs. */
            {
                  cur_found_obj=3;  /* object found at rear             */
            }
            else                    /* other objects detected           */
            {
                  return 4;         /* object spans multiple sections   */
            }

      }

      poke(0x7000,0x00);            /* turn off IR LED's                */
      return cur_found_obj;         /* return flag for sensors          */
}

/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                        end of check_left_sensors                     */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */



/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                        check_right_sensors                           */
/*                                                                      */
/* Written by: Daniel Kucik                                             */
/* Original Creation: 3/22/98                                           */
/* Version: 2.0 (reduced size)                                          */
/*                                                                      */
/* Description: This routine checks the right side IR sensors for       */
/*              obstacles.  It returns the location of the obstacle.    */
/*                                                                      */
/* Returns:  0=no objects detected                                      */
/*           1=object found at right front                              */
/*          2=object found at right center                          */
/*          3=object found at right rear                            */
/*          4=object found spanning multiple sections               */
/*                                                                  */
/* Version: 2.0                                                     */
/* Revisions: 11/28/98: reduced size and options                    */
/*                                                                  */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
int check_right_sensors()
{
      cur_found_obj=0;              /* reset flag                      */
      poke(0x7000,0b01000000);      /* turn on IR LED's                */
      sleep(0.01);                  /* delay while LED's turn on       */
/* checking R1 */
      if(r1_ird() > ird_far_object) /* if object is in range           */
      {
            cur_found_obj=1;        /* set flag - object found -front  */
      }

/* Checking R2 */
      if(r2_ird() > ird_far_object) /* check if object is in range      */
      {
            if(cur_found_obj!=1)    /* if other objects not found      */
            {
                  cur_found_obj=2;  /* object at center                */
            }
            else                    /* object found at front           */
            {
                  return 4;         /* then spans multiple sections     */
            }
      }
/* Checking R3 */
      if(r3_ird() > ird_far_object) /* object in range                  */
      {
            if(cur_found_obj!=1)    /* object not found at front       */
            {
                  cur_found_obj=2;  /* object at right center           */
            }
            else                    /* object found at front            */
            {
                  return 4;         /* object spans multiple sections   */
            }

      }
/* Checking R4 */
      if(r4_ird() > ird_far_object) /* object in range                  */
      {
            if(cur_found_obj==0)    /* if no other objects found       */
            {
                  cur_found_obj=3;  /* object at back left              */
            }
            else                    /* other objects found              */
            {
                  return 4;         /* object spans multiple sections   */
            }

      }
      poke(0x7000,0x00);            /* turn off IR LED's                */
      return cur_found_obj;         /* return flag                      */
}

/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                      end of check_right_sensors                     */
/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */



/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                      check_rear_sensors                              */
/*                                                                     */
/* Written by: Daniel Kucik                                            */
/* Original Creation: 3/22/98                                          */
/* Tested:                                                             */
/* Version: 2.0 (reduced size and output)                              */
/*                                                                     */
/* Description: This routine is used to check the sensors (both IR and */
/*              bump) in the rear of the robot.                        */
/*                                                                     */
/*         0: nothing detected                                         */
/*         1: object on left                                           */
/*         2: object on right                                          */
/*         3: object center                                            */
/*         4: object left and center                                   */
/*         5: object right and center                                  */
/*         6: object all of rear                                       */
/*         7: bumped object                                            */
/*                                                                     */
/* a bumped object takes priority over I/R detected obstacles          */
/* Version: 2.0 (reduced size)                                         */
/* Revisions: 11/28/98: reduced size and capablilities                  */
/*                                                                     */
/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */

int check_rear_sensors()
{

      cur_found_obj=0;              /* used to hold the current status   */
      poke(0x7000,0b00010000);      /* turn on IR LED's for left side   */
      sleep(0.01);                  /* delay while LED's come on        */

/* checking Rear Right IRD */
      if(rr_ird() > ird_far_object) /* if object in range rear right    */
      {
            cur_found_obj=2;        /* set flag                         */
      }

/* Checking rear center ird */
      if(rc_ird() > ird_far_object) /* if object in range rear center   */
      {
            if(cur_found_obj==0)    /* if no other objects found        */
            {
                  cur_found_obj=3;  /* object found at center           */
            }
            else                    /* other objects found at right     */
            {
                  cur_found_obj=4;  /* object at right and center       */
            }
      }

/* Checking rear left ird */
      if(rl_ird() > ird_far_object) /* if object in range rear center   */
      {
            if(cur_found_obj==0)    /* if no other objects found        */
            {
                  cur_found_obj=1;  /* set flag -- object at left       */
            }
            else
            {
                  if(cur_found_obj==3)
                                    /* if object found at center        */
                  {
                        cur_found_obj=5;
                                    /* object center and left           */
                  }
                  else              /* objects at center and right      */
                  {
                        cur_found_obj=6;
                                    /* objects over all of rear         */
                  }
            }

      }
/* start of checking bump sensors */
      if(rear_bump()>rb_tol)        /* check if something was bumped    */
      {
            cur_found_obj=7;        /* object bumped                    */
      }
      poke(0x7000,0x00);            /* turn off IR LED's                */
      return cur_found_obj;         /* return flag                      */
}
/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                         End of check_rear_sensors()                 */
/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */


/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                       Start of check_sh_sensors                     */
/*                                                                     */
/* Written by: Daniel Kucik                                            */
/* Original Creation: 3/22/98                                          */
/* Version: 4.0: down sized version                                    */
/*                                                                     */
/* This routine checks the sensors on the mine detector's search head  */
/* for obsticals in its path.                                          */
/*                                                                     */
/*                                                                     */
/* Returned values: 0 = no objects found                               */
/*                  1 = object tripped left bump switches              */
/*                  2 = object tripped right bump switches             */
/*                      3 = object tripped both right and left bumps   */
/*                                                                      */
/* Version: 4.0                                                         */
/* Revisions: 10/4 - removed I/R detectors from search head             */
/*            11/13 - moved original returned value to sh_sensor_value  */
/*                    returned value changed to side of bump            */
/*            11/28 - down sized and reduced capablilites                */
/*                                                                      */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */

int check_sh_sensors()
{
/* ******************************************************************** */
/* ******************************************************************** */
/* ******************************************************************** */
/*                                Temp Fix for D-Day                    */
/* temp fix -- to resolve problem of grass being seen as an obstacle     */
/* 12/3/98                                                               */
return 0;
/* ******************************************************************** */
/* ******************************************************************** */
/* ******************************************************************** */

/* start of full code */
sh_temp2=0;                         /* reset flag                        */
/* start by checking left side of search head */
      if(sh_l_bump()>shb_tol)       /* check if object was hit @ left   */
      {
            sh_temp2=1;             /* set flag -- object hit at left   */
      }


      if(sh_r_bump()>shb_tol)       /* check if object was hit @ right   */
      {
            if(sh_temp2==0)         /* check if other objects not found */
            {
                  sh_temp2=2;       /* set flag object found at right   */
            }
            else                    /* object was found at left         */
            {
                  sh_temp2=3;       /* objects on both sides            */
            }

      }
            return sh_temp2;        /* return flag                      */
}


/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                        End of check_sh_sensors                        */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */




/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                        Start of center_sh                             */
/*                                                                      */
/* Written by: Daniel Kucik                                             */
/* Original Creation: 3/22/98                                           */
/* Version: 1.0     Tested                                              */
/*                                                                      */
/* This routine checks the sensors on the mine detector's search head   */
/* for obsticals in its path while moving the head to  to center         */
/* position                                                             */
/*                                                                      */
/*                                                                      */
/* Returned values: 0 = successfully moved to center                    */
/*                  1 = objects found could not center search head      */
/*                                                                      */
/* Version: 1.0                                                         */
/* Revisions: 10/4 - removed I/R detectors from search head             */
/*                                                                      */
/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
int center_sh()
{
      servo_on();
      servo_deg1(md_cur_pos);       /* set mine detector to md_cur_pos  */
      sleep(wait1);                 /* delay while servo moves          */
      if(md_cur_pos<90.0)           /* if current position 0<x<90       */
      {
            while(md_cur_pos<90.0 && check_sh_sensors()!=2 &&
check_sh_sensors()!=3)
                                    /* while no objects bumped and       */
                                    /* position<90                      */
            {
                  md_cur_pos=md_cur_pos+md_sensor_speed;
                                    /* increment position                */
                  servo_deg1(md_cur_pos);
                                    /* move servo                       */
                  sleep(wait1);     /* wait while servo moves           */
            }
            if(check_sh_sensors()==2 && check_sh_sensors()==3)
                                    /* if object is hit                 */
            {
                  servo_off();      /* turn off servos                  */
                  return 1;         /* return flag                      */
            }
            else                    /* if no object hit                 */
            {
                  md_cur_pos=90.0;  /* move to 90 degrees               */
                  servo_deg1(md_cur_pos);
                  sleep(wait1);     /* wait for movement                */
            }
      }
      if(md_cur_pos>90.0)           /* if 90<angle<180                  */
      {
            while(md_cur_pos>90.0 && check_sh_sensors()!=1 &&
check_sh_sensors()!=3)
            {
                  md_cur_pos=md_cur_pos-md_sensor_speed;
                                    /* decrement angle                  */
                  servo_deg1(md_cur_pos);
                                    /* move head                        */
                  sleep(wait1);     /* wait for movement                */
            }
            if(check_sh_sensors()==1 && check_sh_sensors()==3)
                                    /* check for hit object            */
            {
                  servo_off();      /* turn off servos                 */
                  return 1;         /* return flag                     */
            }
            else                    /* if no object was hit            */
            {
                  md_cur_pos=90.0;  /* continue moving to 90 degrees   */
                  servo_deg1(md_cur_pos);
                                    /* move arm                        */
                  sleep(wait1);     /* wait for movement               */
              return 0;             /* return completed successfully   */
            }
      }
}

float den=1.85;
float md_sensor_speed2=5.0;
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                            check_for_mines                           */
/*                                                                      */
/* Author: Daniel Kucik                                                 */
/* Robot: Little Ranger                                                 */
/* Version: 1.0                                                         */
/* Original Creation: 11/10/98                                          */
/*                                                                      */
/* This routine checks for land mines.                                  */
/* Returns: 0 = no mines and no obstacles found                         */
/*          1 = no mines and obstacles found                            */
/*          2 = mines and no obstacles found                            */
/*          3 = mines and obstacles found                               */
/*          4 = could not complete sweep because positions of obstacles */
/*              could not be determined from moving_sh_sensors_chk()    */
/*                                                                      */
/*          mine_angle holds the angle of the sweep arm at which the    */
/*                     mines were found.                                */
/*                     An angle of -1 in the array means that no mines  */
/*                     where found in those positions in the array      */
/*                                                                      */
/* Version: 1.0                                                         */
/* Modifications:                                                       */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
int check_for_mines()
{
      servo_on();                   /* turn on servos                  */
      servo_deg1(md_cur_pos);       /* set mine detector to md_cur_pos */
      sleep(wait1);                 /* wait for movement               */
      sh_temp3=0;                   /* reset flag                      */
      md_angle_obs[0]=-1.0;         /* set to safe value               */
      md_angle_obs[1]=-1.0;         /* set to safe value               */
      num_of_mines=0;               /* reset number of mines to 0      */

      if(center_sh()!=0)
                                    /* center search head and check    */
                                    /* no obstacles prevent movement to */
```

```
                              /* center                         */
      {
      servo_off();            /* turn off servos                */
      return 4;               /* return flag                    */
      }
/* now the search head is positioned at 90 degrees (center) */
/* checking 90 degrees to md_max_angle */
      servo_on();             /* turn on servos                 */
      sh_temp4=0;             /* reset temp variable to store   */
                              /* current bump status            */
      sh_temp=0;
      while(sh_temp4==0 && md_cur_pos<md_max_angle)
                              /* while less than the max angle  */
                              /* and no objects have been hit   */
      {
            md_cur_pos=md_cur_pos+md_sensor_speed2;
                              /* increment current position      */
            servo_deg1(md_cur_pos); /* move arm                 */
            sleep(wait1/den); /* wait for move                  */
                              /* move arm                       */
            if(analog(7)>mine_min_value)
                              /* if found mine                  */
            {
                  angle_of_mines[num_of_mines]=md_cur_pos;
                              /* store the angle                */
                  num_of_mines++; /* increment the number of mines */
            }
            if(check_sh_sensors()!=0)
                              /* if obstacle is hit              */
          {
                  sh_temp4=1; /* set flag                       */
          }
      }
      if(sh_temp4==1)         /* if hit object                  */
      {
            md_angle_obs[0]=md_cur_pos;
                              /* store angle object was hit at  */
            sh_temp3=1;       /* set flag                       */
            cur_max_angle=md_cur_pos;
                              /* set current max angle          */
      }
      else                    /* if no obstacles were hit       */
      {
            md_angle_obs[0]=md_max_angle;
                              /* store max angle                */
            cur_max_angle=md_cur_pos;
                              /* set max angle                  */
      }
      center_sh();
      servo_on();
      sh_temp4=0;
      while(sh_temp4==0 && md_cur_pos>md_min_angle)
                              /* while greater than min angle   */
                              /* and no obstacles have been hit */
      {
                md_cur_pos=md_cur_pos-md_sensor_speed2;
                              /* decrement current position     */
            servo_deg1(md_cur_pos); /* move arm                        */
                sleep(wait1/den);   /* wait for movement               */
            if(analog(md_port)>mine_min_value)
                                    /* if found mine                   */
            {
                  angle_of_mines[num_of_mines]=md_cur_pos;
                                    /* store angle of mine             */
                  num_of_mines++;   /* increment mine counter          */
            }

          if(check_sh_sensors()!=0)
                                    /* if found obstacle               */
          {
                  sh_temp4=1;       /* set flag                        */
          }

      }
      if(sh_temp4==1)               /* if obstacle was hit             */
      {
            md_angle_obs[1]=md_cur_pos;
            cur_min_angle=md_cur_pos;
                                    /* store angle at which it was hit  */
            sh_temp3=1;             /* obstale was hit                 */
      }
      else                          /* if no obstacle was hit          */
      {
      md_angle_obs[1]=md_min_angle;
                                    /* store min angle                 */
          cur_min_angle=md_cur_pos;
                                    /* set min angle                   */
      }

      center_sh();                  /* center sweep arm to 90 degrees   */
      if(sh_temp3==0 && num_of_mines>0)
            return 2;
      if(sh_temp3==1 && num_of_mines>0)
            return 3;
      if(sh_temp3==0 && num_of_mines==0)
            return 0;
      if(sh_temp3==1 && num_of_mines==0)
            return 1;

      return sh_temp3;
}


/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                               End of check_for_mines()                  */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */



/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                              manage_angles                             */
/*                                                                        */
/* Author: Daniel Kucik                                                   */
/* Robot: Little Ranger                                                   */
/* Version: 1.0                                                        */
/* Original Creation: 11/28/98                                         */
/*                                                                     */
/* This routine checks the angles of the found mines and simplifies    */
/* them so that they can be used for painting.                         */
/* The new angles are in angle_of_mines2                               */
/* Version: 1.0                                                        */
/* Modifications:                                                      */
/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */

void manage_angles()
{
      temp_cnt=0;
      angle_of_mines2[0]=-1.0;      /* reset to a default value          */
      angle_of_mines2[1]=-1.0;      /* reset to a default value         */
      angle_of_mines2[2]=-1.0;      /* reset to a default value         */

      while(temp_cnt<num_of_mines)
                                    /* check all mine angles            */
      {
            temp_angle=angle_of_mines[temp_cnt];
                                    /* store current mine angle         */
                if(temp_angle<40.0 && temp_angle!=-1.0)
                                    /* if falls on left side            */
            {
                  angle_of_mines2[0]=25.0;
                                    /* store simplified angle           */
            }
                if(temp_angle>140.0 && temp_angle!=-1.0)
                                    /* if falls on right side            */
            {
                  angle_of_mines2[2]=150.0;
                                    /* store simplified angle           */
            }
                if(temp_angle>=40.0 && temp_angle<=140.0 && temp_angle!=-1.0)
                                    /* if at center                     */
            {
                  angle_of_mines2[1]=90.0;
                                    /* store simplified angle           */
            }
            temp_cnt++;             /* increment position count         */
      }
}

/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                            End of manage_angles                      */
/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */


/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                               paint_mines                            */
/*                                                                     */
/* Author: Daniel Kucik                                                */
/* Robot: Little Ranger                                                */
/* Version: 1.0                                                        */
/* Original Creation: 11/28/98                                         */
/*                                                                     */
/* This routine paints found land mines.  It requires that            */
/* manage_angles() is run prior to execution.                         */
/* Version: 1.0                                                       */
/* Modifications:                                                     */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */

void paint_mines()
{
      servo_on();                   /* turn on servos                   */
        if(angle_of_mines2[0]!=-1.0)
                                    /* if angle found to left of robot  */
      {
/* set up angles for turning to left */
/* note this is a fix had cur_max_angle -- grass was interfering */
            md_cur_pos=md_max_angle;
/* end fix */

      servo_deg1(md_cur_pos);
      servo_deg2(right_turn_left);
      servo_deg3( left_turn_left);
      sleep(time_turn_paint);
/* pull forward to mine */
      servo_deg2(right_move_forward);
      servo_deg3( left_move_forward);
      sleep(time_forward_paint);
/* turn stop moving */
      servo_deg2(right_off);
      servo_deg3( left_off);
/* turn on spary paint  */
          poke(0x5000,0xff);
          sleep(time_to_paint);
/* turn off spray paint */
          poke(0x5000,0x00);
/* interchaning angles to go backwards */
          servo_deg3( left_reverse);
          servo_deg2(right_reverse);
          sleep(time_forward_paint);
/* interchanging angles to turn back */
          servo_deg3( left_turn_right);
          servo_deg2(right_turn_right);
          sleep(time_turn_paint);
          servo_deg2(right_off);
          servo_deg3( left_off);
      }
      stop_check();           /* ****************************** */
      if(angle_of_mines2[1]!=-1.0)
                                    /* if mine in front                 */
      {
/* note this is a fix had cur_max_angle -- grass was interfering */
            md_cur_pos=md_max_angle;
/* end fix */

            servo_deg1(md_cur_pos);
            servo_deg2(right_turn_left);

/* set servo angles to move forward */
          servo_deg2(right_move_forward);
          servo_deg3(left_move_forward);
          sleep(time_forward_paint);
/* stop movement angles */
          servo_deg2(right_off);
          servo_deg3(left_off);
/* turn on spray paint */
          poke(0x5000,0xff);
          sleep(time_to_paint);
/* turn off spray paint */
          poke(0x5000,0x00);
/* interchaning angles to go backwards */
          servo_deg3(left_reverse);
          servo_deg2(right_reverse);
          sleep(time_forward_paint);
/* stop servos  */
          servo_deg2(right_off);
          servo_deg3(left_off);
      }
      stop_check();           /* ************************************ */
      if(angle_of_mines2[2]!=-1.0)
                                    /* if mine to right                 */
      {
/* set angles to turn right */
/* note this is a fix had cur_max_angle -- grass was interfering */
            md_cur_pos=md_max_angle;
/* end fix */

            servo_deg1(md_cur_pos);
            servo_deg2(right_turn_left);
            md_cur_pos=cur_max_angle;
            servo_deg1(md_cur_pos);

          servo_deg2(right_turn_right);
          servo_deg3(left_turn_right);
          sleep(time_turn_paint);
/* set angles to move forward */
            servo_deg2(right_move_forward);
            servo_deg3(left_move_forward);
          sleep(time_forward_paint);
/* stop moving */
          servo_deg2(right_off);
          servo_deg3(right_off);
/* spary paint on */
          poke(0x5000,0xff);
          sleep(time_to_paint);
/* spray paint off */
          poke(0x5000,0x00);
/* interchaning angles to go backwards */
          servo_deg3(left_reverse);
          servo_deg2(right_reverse);
          sleep(time_forward_paint);
/* interchanging angles to turn back */
            servo_deg3(left_turn_left);
            servo_deg2(right_turn_left);
          sleep(time_turn_paint);
/* stop moving */
          servo_deg2(right_off);
          servo_deg3(right_off);
      }
      center_sh();
}

/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                              End of  paint_mines                      */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */




/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                            move_forward                               */
/*                                                                      */
/* Author: Daniel Kucik                                                 */
/* Robot: Little Ranger                                                 */
/* Version: 1.0                                                         */
/* Original Creation: 11/28/98                                          */
/*                                                                      */
/* This routine moves the robot forward after the front has been found  */
/* to have no mines.                                                    */
/*                                                                      */
/* Returns: 0: movement forward was successful                          */
/*          1: movement was prevented by an obstacle                    */
/* Version: 1.0                                                         */
/* Modifications:                                                       */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
int move_forward()
{
      center_sh();                  /* center the search head            */
      move_cnt=0;                  /* loop counter                      */
      move_sensor_temp=0;          /* resent sensor flag                */
        while(move_cnt<5 && move_sensor_temp==0)
      {
/* set servos to move forward */
            servo_deg2(right_move_forward);
            servo_deg3(left_move_forward);
            sleep(time_to_move_forward);
/* checking for bumped obstacles        */
            if(check_sh_sensors()!=0)
            {
                  move_sensor_temp=1;
            }
                move_cnt++;
      }
/* turn off servos */
      servo_deg2(right_off);
      servo_deg3(left_off);
/* if bumped obstacle return flag */
      if(move_sensor_temp==1)
            return 1;
      else
            return 0;
}

/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                              End of move_forward                      */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */



/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                             turn_around                                */
/*                                                                        */
/* Author: Daniel Kucik                                                   */
/* Robot: Little Ranger                                                   */
/* Version: 1.0                                                           */
/* Original Creation: 11/28/98                                            */
/*                                                                        */
/* This routine turns (to left) the robot around after sweeping an area */
/*                                                                        */
/* Returns: 0: if no obstacles were found to obstruct the  turn_around   */
/*          1: obstacles prevented the turn around                        */
/*                                                                        */
/* Version: 1.0                                                           */
/* Modifications:                                                         */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */

/* turn to left */
int turn_around()
{
      center_sh();                  /* center the search head            */
      turn_cnt=0;                  /* reset counter                       */
      move_sensor_temp=0;           /* resent sensor flag                 */
        servo_on();
/* turn on servos */
        servo_deg2(right_turn_left);
        servo_deg3(left_turn_left);

        while(turn_cnt<5 && move_sensor_temp==0)
      {
                sleep(time_for_90degrees);
            if(check_sh_sensors()!=0)
            {
                  move_sensor_temp=1;
            }
                turn_cnt++;
      }
        move_forward();
        move_forward();
        move_forward();
      turn_cnt=0;                   /* reset counter                      */
      move_sensor_temp=0;          /* resent sensor flag                  */
        servo_on();
/* turn on servos */
        servo_deg2(right_turn_left);
        servo_deg3(left_turn_left);

        while(turn_cnt<5 && move_sensor_temp==0)
      {
                sleep(time_for_90degrees);
            if(check_sh_sensors()!=0)
            {
                  move_sensor_temp=1;
            }
                turn_cnt++;
      }

      servo_deg2(right_off);
      servo_deg3( left_off);
      if(move_sensor_temp==1)
            return 1;
      else
            return 0;
}

/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                               End of  turn_around                     */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */

/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                            go_around_mine_forward                     */
/*                                                                      */
/* Author: Daniel Kucik                                                 */
/* Robot: Little Ranger                                                 */
/* Version: 1.0                                                         */
/* Original Creation: 11/28/98                                          */
/*                                                                      */
/* This routine is used to go around mines after painting them.         */
/* manage_angles must be run prior to execution                         */
/*                                                                      */
/* Returns: 0 = successful forward                                      */
/*          1 = failed to go forward                                    */
/* Version: 1.0                                                         */
/* Modifications:                                                       */
/*                                                                      */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
int go_around_mine_forward()
{
      check_for_mines();
      manage_angles();
        pass_count=0;
      if(angle_of_mines2[1]==-1.0)
      {
            while(pass_count<time_to_pass_mine)
            {
                  if(move_forward()==1)
                        /* if could not move forward */
                  {
                                servo_deg2(right_off);
                                servo_deg3( left_off);
                        return 1;
                  }
                  else
                  {
                        check_for_mines();
                        manage_angles();
                        if(angle_of_mines2[1]!=-1.0)
                                {
                                        servo_deg2(right_off);
                                        servo_deg3(left_off);
                                return 1;

                        }
                  }
                  pass_count++;
            }
            return 0;

      }

}

/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                         End of  go_around_mine_forward               */
/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */


int avoid_sensor_temp=0;
/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                           go_around_mine_right                       */
/*                                                                     */
/* Author: Daniel Kucik                                                */
/* Robot: Little Ranger                                                */
/* Version: 1.0                                                        */
/* Original Creation: 11/28/98                                         */
/*                                                                     */
/* This routine is used to go around mines after painting them.        */
/* manage_angles must be run prior to execution                        */
/*                                                                     */
/* Returns: 0 = successful turn                                        */
/*          1 = failed to go turn                                      */
/* Version: 1.0                                                        */
/* Modifications:                                                      */
/*                                                                     */
/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
int go_around_mine_right()
{
      /* setup servos to turn right */
      servo_deg2(right_turn_right);
      servo_deg3(left_turn_right);
      servo_on();
      /* reset counters and flags */
      avoid_cnt=0;
      avoid_sensor_temp=0;
        while(avoid_cnt<5 && avoid_sensor_temp==0)
/* turn while there are no obstacles and count has not run out         */
      {
            sleep(time_for_90degrees);

            if(check_sh_sensors()!=0)
/* if obstacle hit turn off servos and set flag */
            {
                  servo_deg2(right_off);
                  servo_deg3(left_off);
                  avoid_sensor_temp=1;
            }
```

```
/* if no obstacles hit turn and inc. count */

            avoid_cnt++;
      }
      servo_deg2(right_off);
      servo_deg3(left_off);
      return avoid_sensor_temp;

}

/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                        End of go_around_mine_right                    */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */



/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                          go_around_mine_left                          */
/*                                                                      */
/* Author: Daniel Kucik                                                 */
/* Robot: Little Ranger                                                 */
/* Version: 1.0                                                         */
/* Original Creation: 11/28/98                                          */
/*                                                                      */
/* This routine is used to go around mines after painting them.         */
/* manage_angles must be run prior to execution                         */
/*                                                                      */
/* Returns: 0 = successful turn                                         */
/*          1 = failed to go turn                                       */
/* Version: 1.0                                                         */
/* Modifications:                                                       */
/*                                                                      */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
int go_around_mine_left()
{
      /* setup servos to turn left */
      servo_deg2(right_turn_left);
      servo_deg3(left_turn_left);
      servo_on();
      /* reset counters and flags */
      avoid_cnt=0;
      avoid_sensor_temp=0;
        while(avoid_cnt<5 && avoid_sensor_temp==0)
/* turn while there are no obstacles and count has not run out          */
      {
            sleep(time_for_90degrees);

            if(check_sh_sensors()!=0)
/* if obstacle hit turn off servos and set flag */
            {
                  servo_deg2(right_off);
                  servo_deg3(left_off);
                  avoid_sensor_temp=1;
            }
/* if no obstacles hit turn and inc. count */

            avoid_cnt++;
      }
      servo_deg2(right_off);
      servo_deg3(left_off);
      return avoid_sensor_temp;

}

/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                         End of go_around_mine_left                  */
/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */



/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                           go_around_check_left                      */
/*                                                                     */
/* Author: Daniel Kucik                                                */
/* Robot: Little Ranger                                                */
/* Version: 1.0                                                        */
/* Original Creation: 11/28/98                                         */
/*                                                                     */
/* This routine is used to go around mines after painting them.        */
/* manage_angles must be run prior to execution                        */
/*                                                                     */
/* Returns: 0 = successful turn (no mines found to left)               */
/*          1 = failed to go turn (mines found)                        */
/* Version: 1.0                                                        */
/* Modifications:                                                      */
/*                                                                     */
/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
int go_around_check_left()
{
      if(go_around_mine_left()==0)
      {
            if(check_for_mines()==0)
            {
                  return 0;
            }
            else
            {
                  go_around_mine_right();

                  return 1;
            }
      }
      else
      {
            go_around_mine_right();

            return 1;
      }
}

/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                         End of go_around_check_left                 */
/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                           go_around_check_right                      */
/*                                                                     */
/* Author: Daniel Kucik                                                */
/* Robot: Little Ranger                                                */
/* Version: 1.0                                                        */
/* Original Creation: 11/28/98                                         */
/*                                                                     */
/* This routine is used to go around mines after painting them.        */
/* manage_angles must be run prior to execution                        */
/*                                                                     */
/* Returns: 0 = successful turn (no mines found to left)               */
/*          1 = failed to go turn (mines found)                        */
/* Version: 1.0                                                        */
/* Modifications:                                                      */
/*                                                                     */
/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
int go_around_check_right()
{
      if(go_around_mine_right()==0)
      {
            if(check_for_mines()==0)
            {
                  return 0;
            }
            else
            {
                  go_around_mine_left();

                  return 1;
            }
      }
      else
      {
            go_around_mine_left();

            return 1;
      }
}

/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                        End of go_around_check_right                  */
/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */




int first_time_right=0;
int turn_check_status;
int avoid_cnt;
int pass_count;


/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                           go_around_mine                             */
/*                                                                      */
/* Author: Daniel Kucik                                                 */
/* Robot: Little Ranger                                                 */
/* Version: 2.0                                                         */
/* Original Creation: 11/28/98                                          */
/*                                                                      */
/* This routine is used to go around mines after painting them.         */
/* manage_angles must be run prior to execution                         */
/*                                                                      */
/* Returns: 0 = successful "go around"                                  */
/*          1 = failed to go aroud                                       */
/* Version: 2.0                                                         */
/* Modifications: 12/2 setup up so that turns to the right since        */
/*                       that area has already be checked and painted   */
/*                                                                      */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
int go_around_mine()
{
      servo_deg2(right_off);
      servo_deg3(left_off);
      servo_on();
        first_time_right=1;

      if(cur_max_angle==md_max_angle)
                              /* if no obstacles found to right       */
      {
            turn_check_status=0;
            if(go_around_check_right()==0)
                              /* check for successful right turn      */
            {
                  while(turn_check_status==0)
                  {

                        if(go_around_mine_forward()==0)
                        {
                              first_time_right++;
                              if(go_around_check_left()==0)
                              {
                                    turn_check_status=1;

                              }
                        }

                  }

                  turn_check_status=0;
                  while(turn_check_status==0)
                  {

                        if(go_around_mine_forward()==0)
                        {
                              if(go_around_check_left()==0)
                              {
                                    turn_check_status=1;

                              }
                        }
                  }
                        while(first_time_right>0)
                  {
                        go_around_mine_forward();
                        first_time_right--;
                  }
                  go_around_check_right();
                  return 0;
            }
            else
            {
                        servo_deg2(right_off);
                        servo_deg3(left_off);
                  return 1;
            }

      }
        else
        {
                 servo_deg2(right_off);
                 servo_deg3(left_off);
                 return 1;
        }
}



/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                        End of go_around_mine                         */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */


/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                          stop_check                                   */
/*                                                                      */
/* Author: Daniel Kucik                                                 */
/* Robot: Little Ranger                                                 */
/* Version: 1.0                                                         */
/* Original Creation: 11/11/98                                          */
/*                                                                      */
/* This routine checks the switch in the rear of little ranger to see   */
/* if it has been toggled down to stop the robot, if so it pauses        */
/* until it is toggled again to the up position.                         */
/*                                                                      */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
void stop_check()
{
      while(run_switch()<100)
      {
            servo_off();
            sleep(wait1);
      }
      servo_on();
}
```

```
/* stop check2 is the same as stop_check execpt switch must go off,     */
/* on, off, on – used to pause while cover is being placed on after     */
/* download and reset                                                   */
void stop_check2()
{
        while(run_switch()>100)
        {
                servo_off();
                sleep(wait1);
        }
        stop_check();
}

/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                         End of stop_check                            */
/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */



/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                           run()                                      */
/*                                                                     */
/* Author: Daniel Kucik                                                */
/* Robot: Little Ranger                                                */
/* Version: 1.0                                                        */
/* Original Creation: 11/11/98                                         */
/*                                                                     */
/* This routine operates the robot through the execution of the other  */
/* routines – acts as a “main()”                                        */
/*                                                                     */
/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */

void run()
{
      int pos=0;
        int length=20;
      int width=10;
      int length_count=0;
      int width_count=0;
        poke(0x5000,0x00);
      while(width_count<width)
      {
            length_count=0;
            while(length_count<length)
            {
                  stop_check();
                  move_forward();
                  stop_check();
                        if(check_for_mines()!=0)
                  {
                        manage_angles();
                        stop_check();
                        paint_mines();
                        stop_check();
                        go_around_mine();
                  }
                  length_count++;
            }
                turn_around();
            width_count++;
       }

}

/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                         End of run                                   */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */

void main()
{
        stop_check2();
        run();
}
```

Appendix B – Routines Before Sized Reduction

The code contained in this appendix could not be used on Little Ranger as a result of the memory limitations described in the Behaviors section of the report. These routines were written prior to knowing about the memory restrictions, so they were reduced to their minimum size and capabilities (seen in Appendix A) to free-up memory. I am including them here, in their original form, because they may be useful to those who continue work on the project, or to other students working on their own robotics projects. The provided routines have all been tested, and work properly. There are additional routines that were written, but could not be tested (not enough memory on the EVBU) which are not included because of their unknown characteristics.

```
int ird_far_object=92;
      /* defines the IR detector value for when a far object is detected */
int ird_tol=2;
      /* defines the tolerence for use with the IR detectors (uses +-) */
int cur_found_obj=0;

int sh_sensor_value=0;
int sh_temp2=0;
int shb_min_value=5;

/* Variables for rear sensors */
int rb_tol=4;                       /* tolerence for rear bump switches  */
int rb_right1=127;                  /* value if bump switch 1 is hit     */
int rb_right2=130;                  /* value if bump switch 1,4 is hit   */
int rb_right3=131;                  /* value if bump switch 1,2,4 is hit */
int rb_left1=46;                    /* value if bump switch 3 is hit     */
int rb_left2=61;                    /* value if bump switch 3,5 is hit   */
int rb_left3=65;                    /* value if bump switch 3,5,2 is hit */
int rb_center1=6;                   /* value if bump switch 2 is hit     */
int rb_center2=16;                  /* value if bump switch 2,4 is hit   */
int rb_center3=36;                  /* value if bump switch 2,4,5 is hit */
int rb_center4=11;                  /* value if bump switch 4 is hit
*/
int rb_center5=23;                  /* value if bump switch 5 is hit     */
int rb_center6=32;                  /* value if bump switch 4,5 is hit    */
/* End of variables for rear sensors */

/* Variables for sh sensors */
int shb_tol=2;                      /* tolerence for rear bump switches */
int shb_left1=46;                   /* Bump Value for R1, */
int shb_left2=81;                   /* Bump Value for R2, */
int shb_left3=40;                   /* Bump Value for R3, */
int shb_left4=105;                  /* Bump Value for R1, R2, */
int shb_left5=72;                   /* Bump Value for R1, R3, */
int shb_left6=97;                   /* Bump Value for R2, R3, */
int shb_left7=117;                  /* Bump Value for R1, R2, R3, */
int shb_right1=45;                  /* Bump Value for R1, */
int shb_right2=99;                  /* Bump Value for R2, */
int shb_right3=38;                  /* Bump Value for R3, */
int shb_right4=102;                 /* Bump Value for R1, R2, */
int shb_right5=71;                  /* Bump Value for R1, R3, */
int shb_right6=98;                  /* Bump Value for R2, R3, */
int shb_right7=116;                 /* Bump Value for R1, R2, R3, */
/* end of sh sensor values */



/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                     check_left_sensors                                */
/*                                                                       */
/* Written by: Daniel Kucik                                              */
/* Original Creation: 3/22/98                                            */
/* Version: 1.0                                                          */
/*                                                                       */
/* Description: This routine checks the left side IR sensors for        */
/*              obstacles.  It returns the location and distance (close */
/*              or far) of the obstacle.                                 */
/*                                                                       */
/* Returns xy:  x=proximity                                              */
/*                1=close                                                */
/*                2=far                                                  */
/*              y=location of object (front, center, back)               */
/*                1=front of robot                                       */
/*                2=center of robot                                      */
/*                3=back (on side) of robot                              */
/*                4=spans multiple sections-assume whole side blocked   */
/*                                                                       */
/* Output: 0 : nothing detected                                          */
/*         11: object detected on left (front) (close)                   */
/*         12: object detected on left (center) (close)                  */
/*         13: object detected on left (back) (close)                    */
/*         14: object detected on left (spans mult. secs.) (close)       */
/*         21: object detected on left (front) (far)                     */
/*         22: object detected on left (center) (far)                    */
/*         23: object detected on left (back) (far)                      */
/*         24: object detected on left (spans mult. secs.) (far)         */
/*                                                                       */
/*         It is assumed that if an object is found that spans multiple */
/*         sections, with one section being close and the other being   */
/*         far that the whole object is considered close (returns 14)   */
/*                                                                       */
/* Version: 1.0                                                          */
/* Revisions:                                                            */
/*                                                                       */
/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
int check_left_sensors()
{
      cur_found_obj=0;
            /* used to hold the current status */
      poke(0x7000,0b10000000);
            /* turn on IR LED's for left side */
/* checking L1 */
      if(l1_ird() > (ird_far_object - ird_tol))
            /* if object in range front left */
      {
            if (l1_ird() > (ird_far_object+ird_tol))
            /* if closer than a far object */
            {
                  cur_found_obj=11;
                        /* object close on left front */
            }
            else
            {
                  cur_found_obj = 21;
                        /* object far on left front */
            }
      }

/* Checking L2 */
      if(l2_ird() > (ird_far_object - ird_tol))
                        /* if object in range middle left */
      {
            if (l2_ird() > (ird_far_object + ird_tol))
                        /* if closer than a far object */
            {
                  if(cur_found_obj==11 || cur_found_obj==21)
                  {
                        cur_found_obj=14;
                        /* object spans mult. sections */
                  }

                  else
                  {
                        cur_found_obj=12;
                        /* object close on left center */
                  }
            }
/* now dealing with far objects */
            else
            {
                  if (cur_found_obj==21)
                        /* check if found far object before */
                  {
                        cur_found_obj = 24;
                        /* if so, then spans multiple sections */
                  }

                  else
                  {
                        if (cur_found_obj==11)
                        /* check if found close object before */
                        {
                              cur_found_obj = 14;
                        /* if so, then spans multiple sections (assumes both
close) */
                        }


                        else
                        {
                              cur_found_obj = 22;
                        /* if not, then only one left center */
                        }
                  }
            }
      }
/* Checking L3 */
      if(cur_found_obj != 14)
                  /* check if need to continue checking left side */
                  /* don't need to if objects found that span mult secs. */
      {

            if(l3_ird() > (ird_far_object - ird_tol))
                        /* if object in range middle left */
            {
                  if (l3_ird() > (ird_far_object + ird_tol))
                        /* if closer than a far object */
                  {
                        if(cur_found_obj==11 || cur_found_obj==21 ||
cur_found_obj==24)
                        /* check if found object close front */
                        {
                              cur_found_obj=14;
                        /* close object spans mult. sections */
                        }
                        else
                        {
                              cur_found_obj=12;
                              /* object close on left center */
                        }
                  }
                  else
                  {
                        if (cur_found_obj==21)
                              /* check if found far object before on front */
                        {
                              cur_found_obj = 24;
                              /* if so, then spans multiple sections */
                        }
                        else
                        {
                              if(cur_found_obj == 11)
                              {
                                    cur_found_obj=14;
                              }
                              else
                              {
                                    cur_found_obj = 22;
                              /* if not, then only one at left center */
                              }
                        }
                  }
            }

      }



/* Checking L4 */
```

```
      if(cur_found_obj != 14)
                  /* check if need to continue checking left side */
                  /* don't need to if objects found that span mult secs.*/
      {

            if(l4_ird() > (ird_far_object - ird_tol))
                        /* if object in range middle left */
            {
                  if (l4_ird() > (ird_far_object + ird_tol))
                        /* if closer than a far object */
                  {
                        if(cur_found_obj>0)
                              /* check if found object close front */
                        {
                              cur_found_obj=14;
                              /* close object spans mult. sections */
                        }
                        else
                        {
                              cur_found_obj=13;
                              /* object close on left back */
                        }
                  }
                  else
                  {
                        if (cur_found_obj>20)
                              /* check if found far object before on front */
                        {
                              cur_found_obj = 24;
                              /* if so, then spans multiple sections */
                        }
                        else
                        {
                              if(cur_found_obj>10)
                              {
                                    cur_found_obj=14;
                              }
                              else
                              {
                                    cur_found_obj = 23;
                              /* if not, then only one at left back */
                              }
                        }
                  }
            }
      }
      return cur_found_obj;
      poke(0x7000,0x00);
}

/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                      end of check_left_sensors                       */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */



/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                      check_right_sensors                             */
/*                                                                      */
/* Written by: Daniel Kucik                                             */
/* Original Creation: 3/22/98                                           */
/* Version: 1.0                                                         */
/*                                                                      */
/* Description: This routine checks the right side IR sensors for       */
/*              obstacles.  It returns the location and distance (close */
/*              or far) of the obstacle.                                */
/*                                                                      */
/* Returns xy:  x=proximity                                             */
/*                1=close                                               */
/*                2=far                                                 */
/*              y=location of object (front, center, back)              */
/*                1=front of robot                                      */
/*                2=center of robot                                     */
/*                3=back (on side) of robot                             */
/*                4=spans multiple sections-assume whole side blocked   */
/*                                                                      */
/* Output: 0 : nothing detected                                         */
/*         11: object detected on right (front) (close)                 */
/*         12: object detected on right (center) (close)                */
/*         13: object detected on right (back) (close)                  */
/*         14: object detected on right (spans mult. secs.) (close)     */
/*         21: object detected on right (front) (far)                   */
/*         22: object detected on right (center) (far)                  */
/*         23: object detected on right (back) (far)                    */
/*         24: object detected on right (spans mult. secs.) (far)       */
/*                                                                      */
/*         It is assumed that if an object is found that spans multiple */
/*         sections, with one section being close and the other being   */
/*         far that the whole object is considered close (returns 14)   */
/*                                                                      */
/* Version: 1.0                                                         */
/* Revisions:                                                           */
/*                                                                      */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
int check_right_sensors()
{
      cur_found_obj=0;
            /* used to hold the current status */
      poke(0x7000,0b01000000);
            /* turn on IR LED's for right side */
/* checking L1 */
      if(r1_ird() > (ird_far_object - ird_tol))
            /* if object in range front right */
      {
            if (r1_ird() > (ird_far_object+ird_tol))
            /* if closer than a far object */
            {
                  cur_found_obj=11;
                        /* object close on right front */
            }
            else
            {
                  cur_found_obj = 21;
                        /* object far on right front */
            }
      }

/* Checking L2 */
      if(r2_ird() > (ird_far_object - ird_tol))
                        /* if object in range middle right */
      {
            if (r2_ird() > (ird_far_object + ird_tol))
                        /* if closer than a far object */
            {
                  if(cur_found_obj==11 || cur_found_obj==21)
                  {
                        cur_found_obj=14;
                        /* object spans mult. sections */
                  }

                  else
                  {
                        cur_found_obj=12;
                        /* object close on right center */
                  }
            }
/* now dealing with far objects */
            else
            {
                  if (cur_found_obj==21)
                        /* check if found far object before */
                  {
                        cur_found_obj = 24;
                        /* if so, then spans multiple sections */
                  }

                  else
                  {
                        if (cur_found_obj==11)
                        /* check if found close object before */
                        {
                              cur_found_obj = 14;
            /* if so, then spans multiple sections (assumes both close) */
                        }


                        else
                        {
                              cur_found_obj = 22;
                        /* if not, then only one right center */
                        }
                  }
            }
      }
/* Checking L3 */
      if(cur_found_obj != 14)
                  /* check if need to continue checking right side */
                  /* don't need to if objects found that span mult secs. */
      {

            if(r3_ird() > (ird_far_object - ird_tol))
```

```
                        /* if object in range middle right */
            {
                  if (r3_ird() > (ird_far_object + ird_tol))
                        /* if closer than a far object */
                  {
                        if(cur_found_obj==11 || cur_found_obj==21 ||
cur_found_obj==24)
                        /* check if found object close front */
                        {
                              cur_found_obj=14;
                        /* close object spans mult. sections */
                        }
                        else
                        {
                              cur_found_obj=12;
                              /* object close on right center */
                        }
                  }
                  else
                  {
                        if (cur_found_obj==21)
                              /* check if found far object before on front */
                        {
                              cur_found_obj = 24;
                              /* if so, then spans multiple sections */
                        }
                        else
                        {
                              if(cur_found_obj == 11)
                              {
                                    cur_found_obj=14;
                              }
                              else
                              {
                                    cur_found_obj = 22;
                              /* if not, then only one at right center */
                              }
                        }
                  }
            }

      }



/* Checking L4 */
      if(cur_found_obj != 14)
                  /* check if need to continue checking right side */
                  /* don't need to if objects found that span mult secs.*/
      {

            if(r4_ird() > (ird_far_object - ird_tol))
                        /* if object in range middle right */
            {
                  if (r4_ird() > (ird_far_object + ird_tol))
                        /* if closer than a far object */
                  {
                        if(cur_found_obj>0)
                              /* check if found object close front */
                        {
                              cur_found_obj=14;
                              /* close object spans mult. sections */
                        }
                        else
                        {
                              cur_found_obj=13;
                              /* object close on right back */
                        }
                  }
                  else
                  {
                        if (cur_found_obj>20)
                              /* check if found far object before on front */
                        {
                              cur_found_obj = 24;
                              /* if so, then spans multiple sections */
                        }
                        else
                        {
                              if(cur_found_obj>10)
                              {
                                    cur_found_obj=14;
                              }
                              else
                              {
                                    cur_found_obj = 23;
                              /* if not, then only one at right back */
                              }
                        }
                  }
            }
      }
      return cur_found_obj;
      poke(0x7000,0x00);
}

/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                      end of check_right_sensors                       */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */



/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                      check_rear_sensors                              */
/*                                                                      */
/* Written by: Daniel Kucik                                             */
/* Original Creation: 3/22/98                                           */
/* Tested:                                                              */
/* Version: 1.0                                                         */
/*                                                                      */
/* Description: This routine is used to check the sensors (both IR and  */
/*              bump) in the rear of the robot.                         */
/*                                                                      */
/* Output: xy                                                           */
/*         x = proximity of object                                          */
/*             1 = bumped object                                            */
/*             2 = close object                                             */
/*             3 = far object                                               */
/*         y = location of object                                           */
/*             1 = object on left                                           */
/*             2 = object on right                                          */
/*             3 = object in center                                         */
/*             4 = object on left and center                                */
/*             5 = object on right and center                               */
/*             6 = object all of rear                                       */
/*                                                                          */
/*          0: nothing detected                                             */
/*         11: bumped object on left                                        */
/*         12: bumped object on right                                       */
/*         13: bumped object center                                         */
/*         14: bumped object left and center                                */
/*         15: bumped object right and center                               */
/*         16: bumped object all of rear                                    */
/*         21: close object on left                                         */
/*         22: close object on right                                        */
/*         23: close object center                                          */
/*         24: close object left and center                                 */
/*         25: close object right and center                                */
/*         26: close object all of rear                                     */
/*         31: far object on left                                           */
/*         32: far object on right                                          */
/*         33: far object center                                            */
/*         34: far object left and center                                   */
/*         35: far object right and center                                  */
/*         36: far object all of rear                                       */
/*                                                                          */
/* Version: 1.0                                                             */
/* Revisions:                                                               */
/*                                                                          */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */

int check_rear_sensors()
{

      cur_found_obj=0;            /* used to hold the current status      */
      poke(0x7000,0b00010000);        /* turn on IR LED's for left side   */
/* checking Rear Right IRD */
      if(rr_ird() > (ird_far_object - ird_tol))
                                    /* if object in range rear right      */
      {
            if (rr_ird() > (ird_far_object+ird_tol))
                                    /* if closer than a far object  */
            {
                  cur_found_obj=22;
                                    /* object close on rear right   */
            }
            else
            {
                  cur_found_obj = 32;
                                    /* object far on rear right           */
            }
      }

/* Checking rear center ird */
      if(rc_ird() > (ird_far_object - ird_tol))
                              /* if object in range rear center   */
      {
            if (rc_ird() > (ird_far_object + ird_tol))
                              /* if closer than a far object  */
            {
                  if(cur_found_obj==22 || cur_found_obj==32)
                  {
                        cur_found_obj=25;
                              /* object right and center           */
                  }

                  else
                  {
                        cur_found_obj=23;
                              /* object close on left center  */
                  }
            }
/* now dealing with far objects */
            else
            {
                  if (cur_found_obj==32)
                              /* check if found far object          */
                              /* on right before                    */
                  {
                        cur_found_obj = 35;
                              /* if far on right and center   */
                  }

                  else
                  {
                        if (cur_found_obj==22)
                              /* check if found close object  */
                              /* before on right                   */
                        {
                              cur_found_obj = 25;
                              /* object on right and center   */
                              /* (assumes both close)              */
                        }


                        else
                        {
                              cur_found_obj = 33;
                              /* far object at center              */
                        }
                  }
            }
      }

/* Checking rear left ird */
      if(rl_ird() > (ird_far_object - ird_tol))
                              /* if object in range rear center    */
      {
            if (rl_ird() > (ird_far_object + ird_tol))
                                    /* if closer than a far object  */
            {
                  if(cur_found_obj==23 || cur_found_obj==33)
                                    /* check for center obj found   */
                  {
                        cur_found_obj=24;
                                    /* object on left and center        */
                  }

                  else
                  {
                        if(cur_found_obj!=0)
                        {
                              cur_found_obj=26;
                                    /* all of rear                      */
                        }
                        else
                        {
                              cur_found_obj=21;
                        }

                  }
            }
/* now dealing with far objects */
            else
            {
                  if (cur_found_obj==33)
                                    /* check if found far object */
                                    /* before at center */
                  {
                        cur_found_obj = 34;
                                    /* object spans left and center */
                  }

                  else
                  {
                        if (cur_found_obj==23)
                                    /* check if found close object */
                                    /* before at center */
                        {
                              cur_found_obj = 24;
                                    /* object spans left and center */
                                    /* (assumes both close) */
                        }


                        else
                        {
                              if(cur_found_obj>30)
                                    /* have found other far */
                              cur_found_obj = 36;
                                    /* if not, then only one left */
                                    /* center */
                              else
                              {
                                    if(cur_found_obj!=0)
```

```
                                {
                                      cur_found_obj=26;
                                }
                                else
                                {
                                      cur_found_obj=31;
                                }
                          }
                    }
              }
        }
      }
/* start of checking bump sensors */
      if(rear_bump()>rb_tol)          /* check if something was bumped */
      {
            /* checking for bump on left side */
            if(rear_bump()>=rb_left1-rb_tol && rear_bump()<=rb_left1+rb_tol)
                                /* check if falls in range of rb_left1 */
            {
                  cur_found_obj=11;       /* object bumped on left */
            }
            if(rear_bump()>=rb_left2-rb_tol && rear_bump()<=rb_left2+rb_tol)
                                /* check if falls in range of rb_left2 */
            {
                  cur_found_obj=11;       /* object bumped on left */
            }
            if(rear_bump()>=rb_left3-rb_tol && rear_bump()<=rb_left3+rb_tol)
                                /* check if falls in range of rb_left3 */
            {
                  cur_found_obj=14;    /* object bumped on left and center */
            }



            /* checking for bump on right side */
            if(rear_bump()>=rb_right1-rb_tol &&
rear_bump()<=rb_right1+rb_tol)
                          /* check if falls in range of rb_right1 */
            {
                  cur_found_obj=12;       /* object bumped on right */
            }
            if(rear_bump()>=rb_right2-rb_tol &&
rear_bump()<=rb_right2+rb_tol)
                          /* check if falls in range of rb_right2 */
            {
                  cur_found_obj=12;       /* object bumped on right */
            }
            if(rear_bump()>=rb_right3-rb_tol &&
rear_bump()<=rb_right3+rb_tol)
                          /* check if falls in range of rb_right3 */
            {
                  cur_found_obj=15;   /* object bumped on right and center */
            }
            /* checking for bump at center */
            if(rear_bump()>=rb_center1-rb_tol &&
rear_bump()<=rb_center1+rb_tol)
                              /* check if falls in range of rb_center1 */
            {
                  cur_found_obj=13;       /* object bumped at center */
            }
            if(rear_bump()>=rb_center2-rb_tol &&
rear_bump()<=rb_center2+rb_tol)
                              /* check if falls in range of rb_center2 */
            {
                  cur_found_obj=13;       /* object bumped at center */
            }
            if(rear_bump()>=rb_center3-rb_tol &&
rear_bump()<=rb_center3+rb_tol)
                              /* check if falls in range of rb_center3 */
            {
                  cur_found_obj=13;       /* object bumped at center */
            }
            if(rear_bump()>=rb_center4-rb_tol &&
rear_bump()<=rb_center4+rb_tol)
                              /* check if falls in range of rb_center4 */
            {
                  cur_found_obj=13;       /* object bumped at center */
            }
            if(rear_bump()>=rb_center5-rb_tol &&
rear_bump()<=rb_center5+rb_tol)
                              /* check if falls in range of rb_center5 */
            {
                  cur_found_obj=13;       /* object bumped at center */
            }
            if(rear_bump()>=rb_center6-rb_tol &&
rear_bump()<=rb_center6+rb_tol)
                              /* check if falls in range of rb_center6 */
            {
                  cur_found_obj=13;       /* object bumped at center */
            }

/* now handling the case for which the entire rear hits an object */
            if(cur_found_obj>20)
                  /* if no bump values have been recorded yet */
            {
                  cur_found_obj=16;
            }
      }
      return cur_found_obj;
}
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                        End of check_rear_sensors()        */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */


/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                      Start of check_sh_sensors                         */
/*                                                                        */
/* Written by: Daniel Kucik                                               */
/* Original Creation: 3/22/98                                             */
/* Version: 3.0                                                        */
/*                                                                     */
/* This routine checks the sensors on the mine detector's search head   */
/* for obsticals in its path.                                          */
/*                                                                     */
/*                                                                     */
/* Returned values: 0 = no objects found                               */
/*                  1 = object tripped left bump switches              */
/*                  2 = object tripped right bump switches             */
/*                  3 = object tripped both right and left bumps       */
/*                                                                     */
/* sh_sensor_value: 0 = no objects found                               */
/*                  1 = object found in front of search head (1 object) */
/*                  2 = object found to right of search head (1 object) */
/*                  3 = object found to left of search head  (1 object) */
/*                  4 = multiple objects found (center and left)       */
/*                  5 = multiple objects found (center and right)      */
/*                  6 = multiple objects found ( right,center,left)     */
/*                  7 = multiple objects found (right and left)        */
/*                                                                     */
/* Requires: Analog access routine                                     */
/*                                                                     */
/*            sh_check()                                               */
/*                                                                     */
/* Version: 2.0                                                        */
/* Revisions: 10/4 - removed I/R detectors from search head            */
/*            11/13 - moved original returned value to sh_sensor_value  */
/*                    returned value changed to side of bump           */
/*                                                                     */
/* Note on the left sensors 1,2,3 are considered the front             */
/*             right sensors 1,2,3 are considered the front            */
/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */

int check_sh_sensors()
{
sh_temp=0;
sh_temp2=0;
/* start by checking left side of search head */
      if(sh_l_bump()>shb_tol)      /* check that object was hit */
      {
            if(sh_l_bump()>=shb_min_value)
            {
                  sh_temp2=1;
            }
            if(sh_l_bump()>=shb_left1-shb_tol &&
sh_l_bump()<=shb_left1+shb_tol)
                  sh_temp=1;
            if(sh_l_bump()>=shb_left2-shb_tol &&
sh_l_bump()<=shb_left2+shb_tol)
                  sh_temp=1;
            if(sh_l_bump()>=shb_left3-shb_tol &&
sh_l_bump()<=shb_left3+shb_tol)
                  sh_temp=1;
            if(sh_l_bump()>=shb_left4-shb_tol &&
sh_l_bump()<=shb_left4+shb_tol)
                  sh_temp=1;
            if(sh_l_bump()>=shb_left5-shb_tol &&
sh_l_bump()<=shb_left5+shb_tol)
                  sh_temp=1;
            if(sh_l_bump()>=shb_left6-shb_tol &&
sh_l_bump()<=shb_left6+shb_tol)
                  sh_temp=1;
            if(sh_l_bump()>=shb_left7-shb_tol &&
sh_l_bump()<=shb_left7+shb_tol)
                  sh_temp=1;
            if(sh_temp!=1)
                  sh_temp=3;
      }


      if(sh_r_bump()>shb_tol)     /* check that object was hit */
      {

            if(sh_r_bump()>=shb_min_value)
            {
                  if(sh_temp2==1)
                  {
                        sh_temp2=3;
                  }
                  else
                  {
                        sh_temp2=2;
                  }
            }
            if(sh_r_bump()>=shb_right1-shb_tol &&
sh_r_bump()<=shb_right1+shb_tol)
            {
                  if(sh_temp==3)
                  {
                        sh_temp=4;
                  }
                  else
                  {
                        sh_temp=1;
                  }
            }
            if(sh_r_bump()>=shb_right2-shb_tol &&
sh_r_bump()<=shb_right2+shb_tol)
            {
                  if(sh_temp==3)
                  {
                        sh_temp=4;
                  }
                  else
                  {
                        sh_temp=1;
                  }
            }
            if(sh_r_bump()>=shb_right3-shb_tol &&
sh_r_bump()<=shb_right3+shb_tol)
            {
                  if(sh_temp==3)
                  {
                        sh_temp=4;
                  }
                  else
                  {
                        sh_temp=1;
                  }
            }
            if(sh_r_bump()>=shb_right4-shb_tol &&
sh_r_bump()<=shb_right4+shb_tol)
            {
                  if(sh_temp==3)
                  {
                        sh_temp=4;
                  }
                  else
                  {
                        sh_temp=1;
                  }
            }
            if(sh_r_bump()>=shb_right5-shb_tol &&
sh_r_bump()<=shb_right5+shb_tol)
            {
                  if(sh_temp==3)
                  {
                        sh_temp=4;
                  }
                  else
                  {
                        sh_temp=1;
                  }
            }
            if(sh_r_bump()>=shb_right6-shb_tol &&
sh_r_bump()<=shb_right6+shb_tol)
            {

                  if(sh_temp==3)

                  {

                        sh_temp=4;
                  }
                  else
                  {
                        sh_temp=1;
                  }
            }
            if(sh_r_bump()>=shb_right7-shb_tol &&
sh_r_bump()<=shb_right7+shb_tol)
            {
                  if(sh_temp==3)
                  {
                        sh_temp=4;      /* object at left and center */
                  }

                  else
                  {
                        sh_temp=1;      /* object at center */
                        }
                }
                if(sh_temp==3)
                        sh_temp=7;              /* object on left and right */
                        if(sh_temp==0)
                                sh_temp=2;              /* object to right only */

        }
                sh_sensor_value=sh_temp;
                return sh_temp2;
}


/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/*                      End of check_sh_sensors                          */
/* +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
```