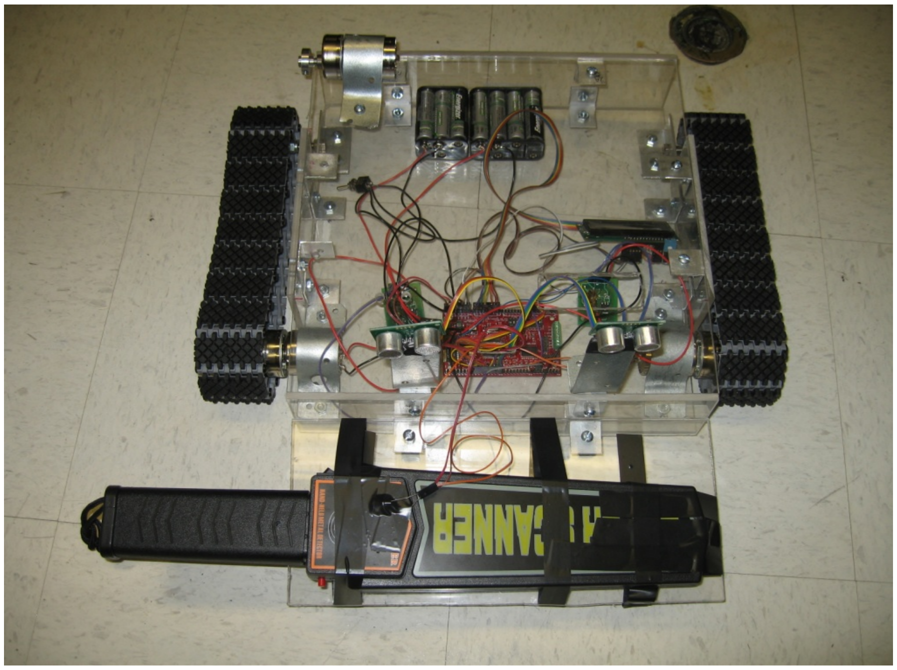
Final Report:
Auto Treasure Finder 5000

By: Philip Sherwood

April 22, 2008



EEL5666C Intelligent Machine Design Lab
Spring 2008
Professors: Dr. A. Arroyo and Dr. E. Schwartz
TA's: Adam Barnett, Mike Pridgen, Sara Keen

Table of Contents

Title Page……………………………………………………1
Table of Contents……………………………………….. 2
Abstract …………………………………..…………... 3
Executive Summary……………………………………... 4
Introduction………………………………….……...5
Integrated Systems…………………………...…………..5
Mobile Platform…………………………………...….. 6
Actuation…………………………………………………7
Sensors……………………………………….….. 8
Behaviors…………………………………….…….. 10
Experimental layout and results……………..……………...10
Conclusion…………………………………..…...11
Appendices……………………………………..……...13

Abstract

The Auto Treasure Finder 5000 is an autonomous metal detecting robot. It will search for a metal object and once found it will attempt to dig into the ground to retrieve the metal object. To achieve the metal detection a wand style metal detector is used. A CdS cell is used across an LED on the detector to interface it with the board. The robot consists of a platform made up of several Plexiglas pieces in a basic box shape held together with several right angle brackets and bolts. The body is roughly a 1 foot wide, 1 foot long and 3 inches tall. The propulsion is achieved using two plastic and rubber tank tracks on each side of the robot each powered by small dc spur gear motors. The digging mechanism is a plow made of metal wire that will dig up objects under the ground. The plow is also connected to a dc motor for motion. Once the metal is detected the robot will automatically shut down. The robot also uses a MAVRIC IIB board for processing, 2 sonars and 4 bump sensors for object avoidance and an LCD for user output.

Executive Summary

The purpose of the ATF 5000 is to locate metal buried in the ground, dig it up, and retrieve the metal object.

A MAVRIC IIB board is used to perform all of the computations and logic to guide the ATF 5000. The MAVRIC IIB is created by BDmicro and utilizes an Atmel ATMeg128 microprocessor with a 14.7 MHz clock to do millions of operations per second.

The object avoidance is accomplished using 4 bump switches and 2 sonars. The sonars were the primary object avoidance mechanism with the bump switches used as a last chance attempt to avoid an object.

The special sensor for the robot is a metal detecting wand called the super scanner. The metal detector is treated as a black box as it is unimportant to the operation of the robot how the detector works as long as it works effectively. To interface the detector with the board a CdS cell is used across an LED in the detector. The LED changes from a dimly lit green color when no metal is detected to a brightly lit red color when metal is detected. This color change is enough to allow a CdS cell to work perfectly and read into the board.

The ATF 5000 uses 2 separate battery packs for all of the power required. A set of 8 NiMH batteries are used to power the motors while a set of 6 NiMH batteries are used to power the board.

3 motors provide actuation for the robot. 2 are used for propulsion on each side of the robot while being connected to a tank track for maximum traction. The third motor is used for the digging mechanism which resembles a plow. The motor raises and lowers to plow to initiate and end the digging process. All of the motors are driven by motor drivers that were provided by Dr. Schwartz. The drivers utilize a 5 volt PWM signal from the board to create an 11 V PWM signal for driving the motors.

An LCD is used as a mechanism to provide feedback with the user about any appropriate action the robot is taking on it's own as well any relevant sensor data that may be useful for debugging purposes.

The platform was hand cut out of several pieces of Plexiglas. The basic shape is a box to house all of the components of the robot. Bolts and brackets hold together the Plexiglas pieces for a tight fit.

Introduction

The Auto Treasure Finder 5000 is designed with the lazy, forgetful pirate in mind. Have you ever buried all your treasure only to lose you precious treasure map? Would you rather just lay on the beach catching rays instead of shoveling away for buried booty? That's where the Auto Treasure Finder 5000 comes in. Using a metal detector it will find buried metal and then it will dig into the ground using it's robot arms to retrieve the lost loot. Plunder away!

The robot is designed to find treasure on a beach like environment with a soft ground. This environment is perfect for amateur treasure seekers who use their own metal detectors to find someone’s lost treasure underground. This paper is written to explain how the robot accomplishes it’s goal of seeking, finding, and retrieving lost treasure through the use of a mobile platform, several sensors, an integrated system and intelligent behavior algorithms.

Integrated System

The final robot required integration of a computer processor, LCD, mobile platform, two drive belts on each side, a metal detector coil, a digging device, and several sensors. The processor used for the ATF 5000 was the Atmel ATMega 128. This processor was attached interfaced using a premade MAVRIC IIB purchased from bdmicro.com. This board does all the processing and logic required for the ATF 5000 to have apparent intelligent qualities. All the sensors, actuation, and electronics are interfaced through this board using C computer code shown in the appendix.

An LCD screen is used as the primary display to the user any relevant data or words. This LCD uses a standard 16 input/output pin setup which was run in 4 bit mode. This display can display up to 32 characters on 2 lines. This display was used as the primary means for the robot to display any relevant information about the output of the sensors or the data used by the processor.

The sensors provided feedback from the environment. Most of the sensors were used for object avoidance. 2 sonars and 4 bump switches were used. The sonars are simply used as a test as to whether any object is within 1 foot of the left front or right front of the platform. If this case is true to robot will turn away from the object while continuing forward.

The actuation is done by 3 12 volt DC motors. The motors are each connected to their own motor driver. The motor drivers allow the 12 volt motors to be controlled by the board which runs at 5 volts. To accomplish this, the board supplies a 5 volt peak PWM signal which is converted to a 12 volt signal by the driver. The driver is supplied with its own 12 volt power source to use for the voltage increase. 2 motors are connected to drive tracks on the robot while 1 motor is used for a plow designed to scoop up a metal object.

The power source comes from 2 battery packs of NiMH batteries. A pack of 8 powers the motors while a pack of 6 powers the board. These are simply connected to the power ports required by the MAVRIC IIB board.

Mobile Platform

The platform is a fairly basic box shape made of Plexiglas. The propulsion system is based on tank tracks and not wheels as usual. This is useful in an environment such as the beach because tank tracks provide needed traction to move effectively through soft sand. The tracks used for the robot were purchased from lynxmotion.com and are segmented pieces of both plastic and rubber. The rubber is on the bottom to provide traction on nearly any surface. The plastic is used for the structural support of each rubber piece of the track. Each piece fits into each other with a nylon bushing in between each piece. Each track is powered by a single drive sprocket connected to a motor on each side. A freewheeling sprocket is used on each side to maintain a proper shape of the tracks for efficient travel.

The platform has several strategically placed holes for the LCD and switches for the robot. This was done to streamline the design.

The metal detector is taped on the front of a cantilever beam bolted to the main chassis. Electrical taped is used to secure the metal detector because any normal metal fastener would not allow for appropriate responses from the metal detector. This provides the easiest construction method as the detector is already in a good structure.

The digging mechanism is a plow made of wire with Plexiglas supports on each side of it. The plow is bolted to an arm attached to a motor.

All of the motors were mounted to the platform using custom made motor mounts. This was done to ensure a perfect fit and spacing on the robot. Hubs made specifically for the motors were purchased from lynxmotion.com to ensure the motors could efficiently link power to the drive train.

Nearly of the electronics are held in place using Velcro for a snug but removable fit. This proved very advantageous as the Velcro wasn't too strong as to damage the delicate electronics but was strong enough to stay put with the robot in motion. Electrical tape was also used for the wiring and to hold the LCD in place. This was the fastest method but perhaps not the most elegant solution.

The picture on the next page in Figure 1 most effectively shows the finished platform without the plow in place and without the LCD or switches in the correct place.

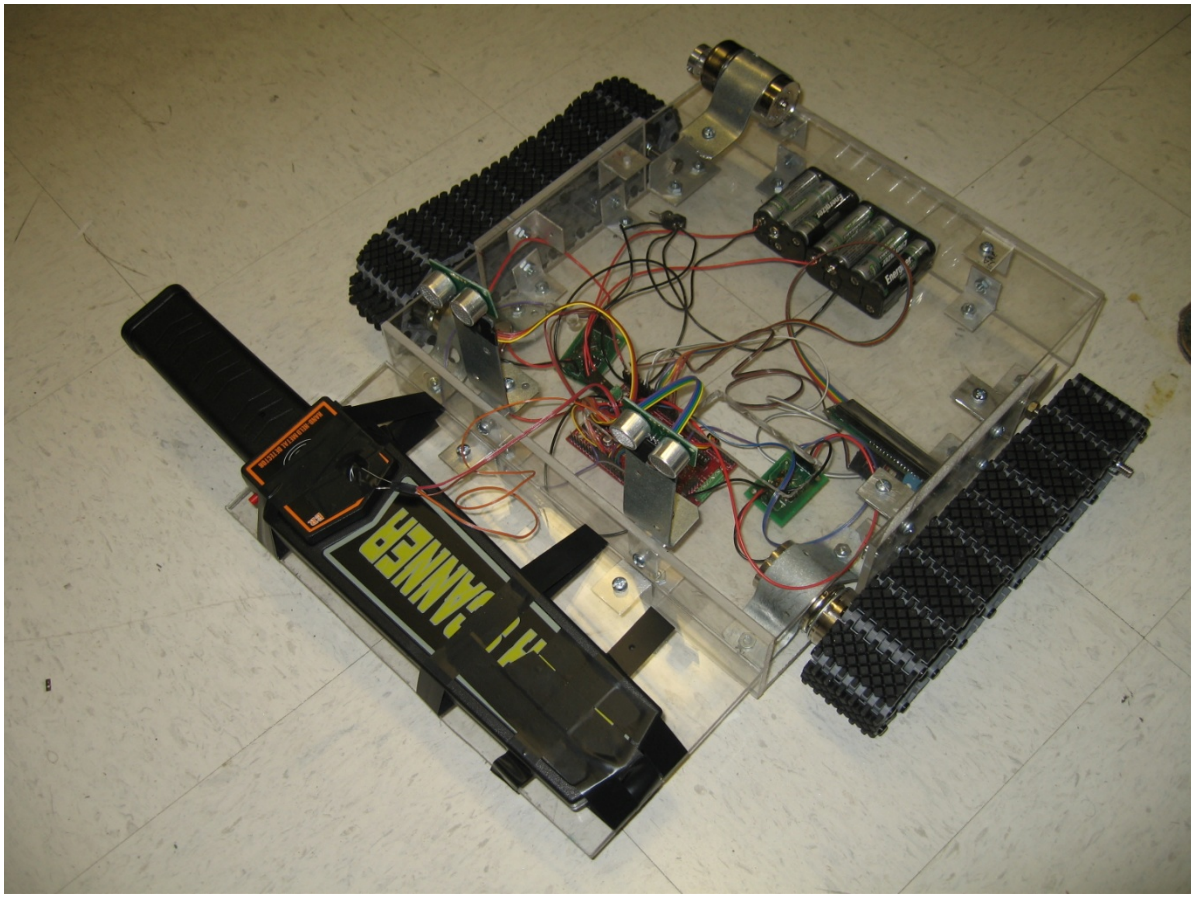


Figure 1. Main robot body without the plow.

Actuation

All of the actuation on the robot is done through 3 motors. These are basic DC spur gear motors purchased through lynxmotion.com. Two motors are used to both drive the robot forward and one is used for a plow to dig up metal. All three were controlled using motor controllers designed by the MIL lab in previous semesters.

To facilitate motion, tank tracks were used. This proved more difficult than I had first hoped for. Each track rolls on 2 sprockets. The sprocket in the rear is freewheeling while the sprocket in the front is a drive sprocket. The sprocket is made of plastic and is bolted to the motor through the use of a hub specifically designed for the motor. I then bolted the motors to the Plexiglas using custom bent metal for the motor mounts. I was not careful enough when aligning the motors to be perfectly perpendicular to the side walls. This leads to a slight angle of the tracks from the true straight line of the robot. This caused relatively severe vibrational issues that shake all of the parts on the robot when the robot is in motion.

The last motor is designed to help pick up metal with a plow. This is a very difficult task and is one I’m not sure can be done effectively with my platform. The platform needs a redesign in order to function properly. The plow was originally designed on paper to be supported by both ends. However, due to poor planning the plow could only be supported on one end. This greatly reduces the strength and digging power of the plow.

The motors were bought initially because of the high torque that was claimed by the manufacturer. However, the motors do not seem to be operating as strongly I had hoped for. The issue may be due to the motor controllers not operating as I am expecting. I may need more experimentation to figure out what needs to be changed for max torque.

The hardest part I personally faced with actuation was not having the motor controllers work until 2 weeks from the end of the class. I had found, purchased, and installed all of the correct parts from the spec sheet but the drivers still did not work. It took many hours and weeks searching for parts that would work as well as finding someone who knew how the fix them. The motor driver boards were eventually hand modified to work correctly but not after wasting several weeks.

Sensors

The ATF 5000 uses two sonar sensors, 4 bump switches, 1 Cds cell, and one metal detecting wand to gather data from the environment to operate logically.

The Auto Treasure Finder 5000 (ATF 5000) uses several sensors to obtain information about the outside world. Since robots are not equipped with biological sensors such as eyes and ears like humans, robots rely on cleverly designed electrical sensors. In order to avoid running into people or any other objects the ATF 5000 utilizes 2 sonar sensors to estimate the distance to the nearest object and will avoid anything that comes to close. The sonars used are the SRF05 sensors shown below in Figure 2 supplied by acroname.com.

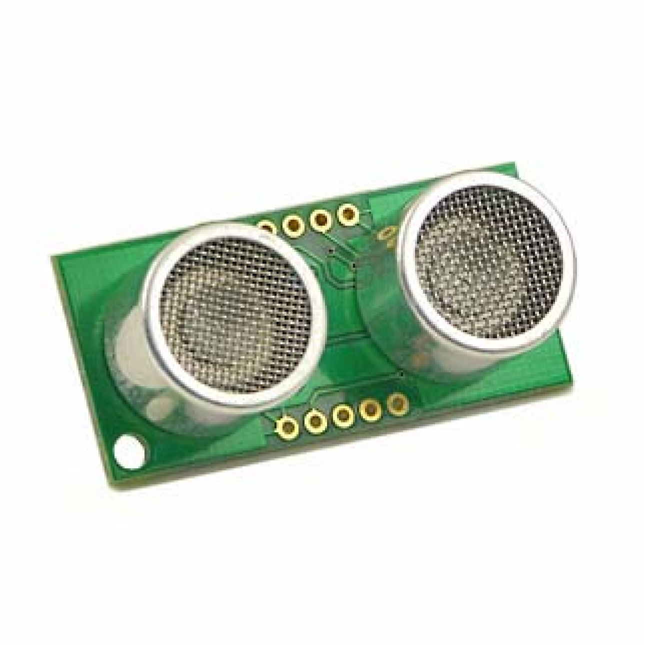


Figure 2. SRF05 Ultra-Sonic Ranger

The 2 sensors are placed in the front of the robot about 8 inches apart and with the left sonar facing slightly left by about 10 degrees and the right sonar facing slightly right by about 10 degrees. This sonar setup is designed to be able to detect objects both directly in front of the robot as well as any object slightly offset to the right or left. The sonars work by generating a short high frequency sound wave and measuring the time it takes for the wave to reflect off an object and come back to the sonar. A 10 microsecond pulse is generated and an echo pulse is received at a measured time later. The time is measured by the board in clock cycles. A correlation between clock cycle time and actual distance can then be established using experimental data. The test data is shown in the appendix. A correlation of $y = 89.57 x + 80.94$ was developed. This equation can then be used to find an object's distance in inches with a known clock cycle time by and subtracting 80.94 and then dividing the number by 89.57.

While the sonars are the primary means of object avoidance, the ATF 5000 also employs bump switches. These switches are placed at various locations along the front of the robot. They are switches that are activated when the robot physically bumps into an object. Once triggered the robot will take immediate action and reverse itself and move in a new direction. These are normally not needed as the sonar should avoid must objects before a collision occurs. However, these switches are in place for safety reasons or for an object that is did not get picked up by the sonar. The switches work by being in the normally open position or not normally connected. The board will read the switch and measure 5 volts when the switch is opened and 0V when the switch is closed. This reading is accomplished by a simple voltage divider connected to the switch.

The special sensor on the robot is what makes the ATF 5000 unique. The sensor is a wand metal detector purchased from ebay.com shown in Figure 3.

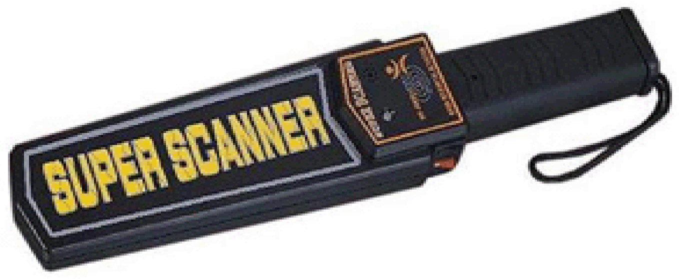


Figure 3. Wand metal detector.

The detector was designed for handheld use for scanning a person for metallic objects. These detectors are frequently seen in airports scanning for dangerous objects hidden in people's clothing. The low price and ease of use makes this a good choice for metal detection. When metal is placed near the scanner, it will make noise, vibrate, as well as light an led. The challenge is to integrate the metal detector for use in the robot. The led on the detector is used for this purpose. Since the led changes from a dim green to a bright red color when a metal object is detected, a CdS cell is placed across the led and is measured by the ADC on the MAVRIC IIB board. When the reading is lower than normal the ATF 5000 knows metal is detected. The scanner is mounted on the front of the robot horizontally on plastic so none of the metal objects in the robot affect the detector.

Behaviors

The behaviors of the ATF 5000 are relatively basic as the ultimate behavior is simply a treasure hunt. The robot simply moves forward and checks all of it's sensors to see whether it should continue forward or perform some action. Assuming it does not encounter metal the robot simply does object avoidance.

The object avoidance is a matter of first checking whether any bump switches are pressed. If so, the robot will back up and turn right to avoid the object it hit. If the bump switches are not depressed then the robot checks the sonar data. If the sonars detect an object that is within one foot of the robot it will turn accordingly. If an object is within one foot on the front right sonar it will turn left and if an object is within one foot of the front left sonar it will turn right. If it is within one foot of both sonars the robot will back up and turn right to avoid the object. This is checked in an infinite loop.

During the loop there is a check for metal. If the MAVRIC IIB board sensing metal using the metal detector it will enter into a digging mode. This robot will first stop the motors when metal is detected. Then the robot will turn 180 degrees. This turn is simply allowing the motors to turn for a predetermined amount of time. This time was found experimentally in the lab to generate a nearly 180 degree turn every time tested. The turn is designed to put the plow directly over the location where the metal detection occurred. Then the plow is lowered a predetermined amount of time to allow the plow to fully lower into the ground. This time was again found through experimentation. Then the robot moves forward for a short time as to plow the ground in front of it. The metal will hopefully be stirred up and caught in the front of the plow. Then the plow returns to it's original location again while lifting up the metal object off the ground and placing it on the robots surface.

Experimental Layout and Results

Most of the sensors for my robot did not require significant experimentation. The metal detector range was fixed to around 6 inches so further experimentation was not necessary. However, experimentation was important in a few key ways.

To generate an appropriate PWM signal by my board for my motor controllers, experimentation was used. My motor controllers needed a 20 MHz PWM signal to operate efficiently. Since, I could not figure out the appropriate value for the input compare register I simply hooked up the output of the board to an oscilloscope. Since the value of the input compare register linearly correlates to the PWM frequency I only needed to make 1 measurement of the frequency for a guessed value of input compare register. Based on the measurement the guess value for the ICR was multiplied by a factor of measured frequency over desired frequency. This proved to be a value of 368 for my robot.

To ensure the bump switches were working a volt meter was used to measure the voltage across the switch. The switches read 5 volts open and 0 volts closed which meant they worked perfectly.

The values from the CdS cell across the metal detector LED was found with experimentation. The value had to be measured with an analog to digital converter channel so experimentation was used to both design the circuit and the logic in the processor. A resistor of 5K was used in series with the CdS cell to balance the resistance of 10K without much light and 1K with a lot of light. Using the ADC port the values measured were displayed on the LCD. It was found that when no metal was found the value was around 180 and when metal was found there was a value around 120. Thus, a threshold of 150 was used to determine if metal was found or not.

The sonars were tested most thoroughly of all the sensors for the best data. The time in clock cycles from the echo off of a sheet of paper at known distances was measured to correlate distance to clock count. This data is shown in the appendix.

Conclusion

Overall, the robot was a moderate success. The robot can complete its objectives in a very controlled environment but is not constructed or designed well enough to have real world capabilities. The robot has a decent platform but more testing and experimentation is needed to exploit the full potential of the robot.

The platform needs to be redesigned for a more effective bot. The robot is very hard to put together because each bolt must be supported on each side to screw into position because there is a nut on one side. The platform is fairly small so it is difficult to fit your hand into the robot to tighten the bolts. The motors were also not perfectly aligned which makes the robot have vibration issues. Next time I will not use custom motor mounts but mounts supplied by the motor manufacturer.

The plow does not work as I would have hoped for. If the motor drivers would have worked earlier in the semester I could have designed a better digging mechanism. The mechanism is simply not strong enough but I believe a few modifications could be done to the existing platform to allow for a stronger plow.

Also, the metal detector does not do a great job for this robot. Throughout the semester I tried to use other means of detecting metal but when I was running out of time I settled on the easiest method. The detector tends to have a very weak detection range which limits the robot.

The code written is not the best style either. It is poorly commented and not effectively written. Interrupts should be used in the future to streamline the code and allow for efficient transitions for each behavior.

The most important lesson for future students when designing a robot for this class is to make sure you stay within the time limits. The main parts of your robot should be ordered very early in the semester. My motor drivers for my robot were not working until I had 2 weeks left in class. This greatly hampered my ability to test or design any motion algorithms or mechanics of the robot. I would have simply bought drivers if I had more money to spend and I would have had the robot designed much sooner.

Appendices

Sonar Test:

	Sonar 1	Sonar 2
Object Distance (in)	Echo Time (clock cycle)	Echo Time (clock cycle)
1	135	135
2	243	234
3	338	343
4	415	414
5	517	533
6	640	628
7	725	722
8	760	798
9	905	913
10	1037	1026
11	1100	1101
12	1274	1235
18	1543	1770
24	2250	2155

Table 1. Sonar data collected by holding a sheet of paper perpendicular to each sensor alone.

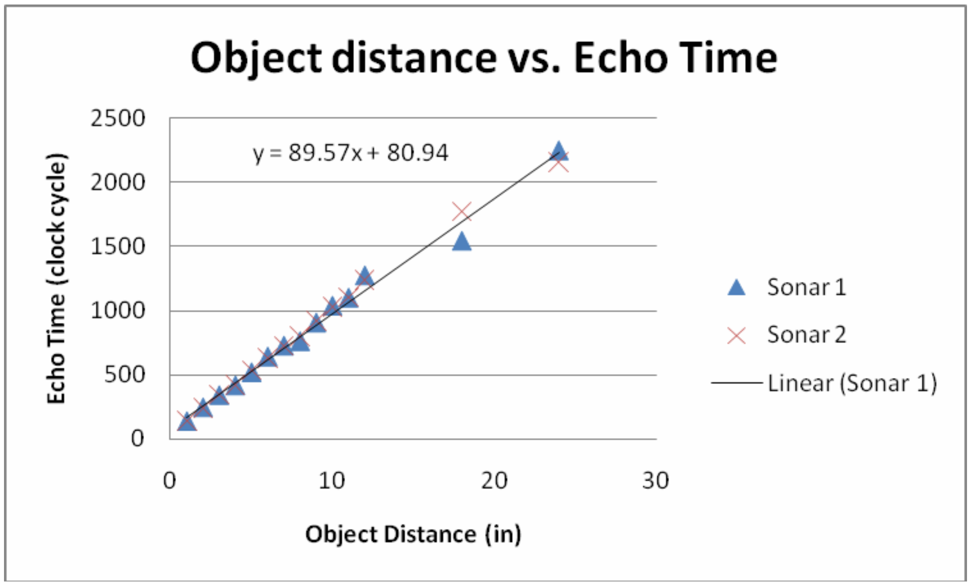


Figure 4. Sonar distance correlation.

```
// Final code: Philip Sherwood
// Spring 2008: IMDL
//Auto Treasure Finder 2000
```

```
#include <avr/io.h>

void long_delay();
void short_delay();
void lcd_init();                    // sets lcd in 4 bit mode, 2-line mode, with cursor on and set to blink
void lcd_cmd();                              // use to send commands to lcd
void lcd_disp();                    // use to display text on lcd
void lcd_displong();
void lcd_clear();                   // use to clear LCD and return cursor to home position
void lcd_row(int row);              // use to put the LCD at the desired row

int get_Sonarleft();                         // use to read value left sonar module
int get_Sonarright();                        // use to read value right sonar module
void config_adc(void);
void init_adc( void );
void        pwm_init(void);

/* IMPORTANT!

Before using this code make sure your LCD is wired up the same way mine is, or change the code to
match the wiring of your own LCD.  My LCD is connected to PortC of the At-Mega128 in the following manner:

PortC bit 7         :           LCD data bit 7 (MSB)
PortC bit 6         :           LCD data bit 6
PortC bit 5         :           LCD data bit 5
PortC bit 4         :           LCD data bit 4 (LSB)

PortC bit 3         :           (not connected)
PortC bit 2         :           LCD enable pin (clock)
PortC bit 1         :           LCD R/W (read / write) signal
PortC bit 0         :           LCD RS (register select) pin

Also remember you must connect a potentiometer (variable resistor) to the vcc, gnd, and contrast pins on the LCD.
The output of the pot (middle pin) should be connected to the contrast pin.  The other two can be on either pin.

*/



void long_delay()           // delay for 10000 clock cycles
{
        long int ms_count = 0;
        while (ms_count < 10000)
        {
                ms_count = ms_count + 1;
        }
}
void short_delay()                  // delay for 10000 clock cycles
{
        long int ms_count = 0;
        while (ms_count < 1000)
        {
                ms_count = ms_count + 1;
        }
}
```

```
void lcd_cmd( unsigned int myData )
{

        /* READ THIS!!!

        The & and | functions are the BITWISE AND and BITWISE OR functions respectively.  DO NOT
        confuse these with the && and || functions (which are the LOGICAL AND and LOGICAL OR functions).

        The logical functions will only return a single 1 or 0 value, thus they do not work in this scenario
        since we need the 8-bit value passed to this function to be preserved as 8-bits
        */


        unsigned int temp_data = 0;

        temp_data = ( myData | 0b00000100 );        // these two lines leave the upper nibble as-is, and set
        temp_data = ( temp_data & 0b11110100 );   // the appropriate control bits in the lower nibble
        PORTC = temp_data;
        long_delay();
        PORTC = (temp_data & 0b11110000);                    // we have written upper nibble to the LCD

        temp_data = ( myData << 4 );                         // here, we reload myData into our temp. variable and
shift the bits
                                                                                    // to the left 4
times.  This puts the lower nibble into the upper 4 bits

        temp_data = (temp_data & 0b11110100);     // temp_data now contains the original
        temp_data = (temp_data | 0b00000100);       // lower nibble plus high clock signal

        PORTC = temp_data;                                                   // write the data to PortC
        long_delay();
        PORTC = (temp_data & 0b11110000);                    // re-write the data to PortC with the clock signal low
(thus creating the falling edge)
        long_delay();

}

void lcd_disp(unsigned int disp)
{

        /*

        This function is identical to the lcd_cmd function with only one exception.  This least significant bit of
        PortC is forced high so the LCD interprets the values written to is as data instead of a command.

        */

        unsigned int temp_data = 0;

        temp_data = ( disp & 0b11110000 );
        temp_data = ( temp_data | 0b00000101 );
        PORTC = temp_data;
        //short_delay();
        PORTC = (temp_data & 0b11110001);
        short_delay();                                                  // upper nibble
        temp_data = (disp << 4 );
        temp_data = ( temp_data & 0b11110000 );
        temp_data = ( temp_data | 0b00000101 );
        PORTC = temp_data;
        short_delay();
        PORTC = (temp_data & 0b11110001);
        short_delay();                                  // lower nibble

}
void lcd_displong(unsigned int disp)
{

        /*

        This function is identical to the lcd_cmd function with only one exception.  This least significant bit of
        PortC is forced high so the LCD interprets the values written to is as data instead of a command.

        */

        unsigned int temp_data = 0;

        temp_data = ( disp & 0b11110000 );
        temp_data = ( temp_data | 0b00000101 );
        PORTC = temp_data;
        long_delay();
        PORTC = (temp_data & 0b11110001);
        long_delay();                                   // upper nibble

        temp_data = (disp << 4 );
        temp_data = ( temp_data & 0b11110000 );
        temp_data = ( temp_data | 0b00000101 );
        PORTC = temp_data;
        long_delay();
        PORTC = (temp_data & 0b11110001);
        long_delay();                                   // lower nibble

}


void lcd_init()
{
        lcd_cmd(0x33);          // writing 0x33 followed by
        lcd_cmd(0x32);          // 0x32 puts the LCD in 4-bit mode

        lcd_cmd(0x28);          // writing 0x28 puts the LCD in 2-line mode

        lcd_cmd(0x0F);          // writing 0x0F turns the display on, curson on, and puts the cursor in blink
mode

        lcd_cmd(0x01);          // writing 0x01 clears the LCD and sets the cursor to the home (top left) position

        //LCD is on... ready to write

}
```

```
void lcd_string(char *a)
{

	/*

	This function writes a string to the LCD.  LCDs can only print one character at a time so we need to
	print each letter or number in the string one at a time.  This is accomplished by creating a pointer to
	the beginning of the string (which logically points to the first character).  It is important to understand
	that all strings in C end with the "null" character which is interpreted by the language as a 0.  So to print
	an entire string to the LCD we point to the beginning of the string, print the first letter, then we increment
	the pointer (thus making it point to the second letter), print that letter, and keep incrementing until we reach
	the "null" character".  This can all be easily done by using a while loop that continuously prints a letter and
	increments the pointer as long as a 0 is not what the pointer points to.

	*/

	while (*a != 0)
	{
		lcd_disp((unsigned int) *a);		// display the character that our pointer (a) is pointing to
		a++;								// increment a
	}
	return;

}
void lcd_stringlong(char *a)
{

	/*

	This function writes a string to the LCD.  LCDs can only print one character at a time so we need to
	print each letter or number in the string one at a time.  This is accomplished by creating a pointer to
	the beginning of the string (which logically points to the first character).  It is important to understand
	that all strings in C end with the "null" character which is interpreted by the language as a 0.  So to print
	an entire string to the LCD we point to the beginning of the string, print the first letter, then we increment
	the pointer (thus making it point to the second letter), print that letter, and keep incrementing until we reach
	the "null" character".  This can all be easily done by using a while loop that continuously prints a letter and
	increments the pointer as long as a 0 is not what the pointer points to.

	*/

	while (*a != 0)
	{
		lcd_displong((unsigned int) *a);	// display the character that our pointer (a) is pointing to
		a++;								// increment a
	}
	return;

}


void lcd_int(int value)
{

	/*
	This routine will take an integer and display it in the proper order on
	your LCD.  Thanks to Josh Hartman (IMDL Spring 2007) for writing this in lab
	*/

	int temp_val;
	int x = 10000;				// since integers only go up to 32768, we only need to worry about
							// numbers containing at most a ten-thousands place

	while (value / x == 0)	// the purpose of this loop is to find out the largest position (in decimal)
	{						// that our integer contains.  As soon as we get a non-
zero value, we know
		x/=10;				// how many positions there are int the int and x will be
properly initialized to the largest
	}						// power of 10 that will return a non-zero value when
our integer is divided by x.


	while (x >= 1)				// this loop is where the printing to the LCD takes place.  First,
we divide
	{							// our integer by x (properly
initialized by the last loop) and store it in
		temp_val = value / x;		// a temporary variable so our original value is preserved.
Next we subtract the
		value -= temp_val * x;		// temp. variable times x from our original value.  This will
"pull" off the most
		lcd_displong(temp_val+ 0x30);	// significant digit from our original integer but leave all the
remaining digits alone.
								// After this, we add a hex 30 to our
temp. variable because ASCII values for integers
		x /= 10;				// 0 through 9 correspond to hex numbers 30 through
39.  We then send this value to the
	}							// LCD (which understands ASCII).
Finally, we divide x by 10 and repeat the process
								// until we get a zero value (note:
since our value is an integer, any decimal value
	return;						// less than 1 will be truncated to a 0)

}

void lcd_clear()		// this function clears the LCD and sets the cursor to the home (upper left) position
{
	lcd_cmd(0x01);

	return;
}

void lcd_row(int row)	// this function moves the cursor to the beginning of the specified row without changing
{					// any of the current text on the LCD.

	switch(row)
	{

		case 0: lcd_cmd(0x02);
		case 1:   lcd_cmd(0xC0);

	}

	return;
```

```
}

int get_Sonarleft()
{

        int n = 0;

        PORTA = (PINA & 0xFE);          // these two lines create a rising edge
        PORTA = (PINA | 0x01);  // on PortA pin 1

        while (n < 40)
        {
                //waste enough clock cycles for at least 10us to pass
                n += 1;
                n++;
        }

        PORTA = (PINA & 0xFE);          // force PortA pin 7 low to create a falling edge
                                                // this sends out the trigger

        while (!(PINA & 0x02))
        {
                // do nothing as long as echo line is low
        }

        n = 0;  //re-use our dummy variable for counting

        while (PINA & 0x02)
        {
                n += 1;         // add 1 to n as long as PortA pin 0 is high
        }

        //when we get here, the falling edge has occured

        return n;

}

int get_Sonarright()
{

        int n = 0;

        PORTA = (PINA & 0xEF);          // these two lines create a rising edge
        PORTA = (PINA | 0x10);  // on PortA pin 4

        while (n < 40)
        {
                //waste enough clock cycles for at least 10us to pass
                n += 1;
                n++;
        }

        PORTA = (PINA & 0xEF);          // force PortA pin 7 low to create a falling edge
                                                // this sends out the trigger
```

```
            while (!(PINA & 0x20))
            {
                        // do nothing as long as echo line is low
            }

            n = 0;     //re-use our dummy variable for counting

            while (PINA & 0x20)
            {
                        n += 1;                  // add 1 to n as long as PortA pin 0 is high
            }

            //when we get here, the falling edge has occured

            return n;

}

int main(void)
{

   long i;
   int analogLow = 0;
   int analogHigh = 0;

   config_adc();

   init_adc(); // initialize ADC converter

   DDRC = 0xFF;                          // set portC to output  (could also use DDRC = 0b11111111)
   DDRE = 0xFF;                          // set portE to output
   DDRB = 0xFF;                          // set portB to output

            long Sonarleft,Sonarright = 0;

            DDRA = 0x99;    // 2 sonar are connected with trigger echo ground power
                                                      // times 2 from PortA.7 to PortA.0

            PORTA = (PINA & 0x00);             // clear port A
            PORTA = (PINA | 0x88);  // power both sonars
            PORTE = (PINE & 0x00);// clear port E
            PORTE = (PINE | 0b01001111);     // give motors forward directions





   lcd_init();     // set lcd in 4 bit mode, 2-line mode, with cursor on and set to blink


   lcd_stringlong("Your LCD works.");     // if your LCD is wired up correctly, you will see this text
                                                // on it when you power up your Micro-controller board.
   lcd_row(1);
   lcd_stringlong("DeStRoY hUmAnS");
            for (i = 0; i < 450; i++)
              {
                long_delay();   //delay to read LCD (humans reading)
              }

            pwm_init();
            OCR1A = 365;
            OCR1B = 365;


   while(1)

   {
   OCR1C = 0;

              analogLow = ADCL; // read ACD low register
              analogHigh = ADCH; // read ACD high register
              PORTB ^= 0x01;
                     lcd_clear();
                     lcd_string("Finding Treasure");
                     Sonarleft = get_Sonarleft();
                     short_delay();
                     Sonarright = get_Sonarright();

            if ((!(PIND & 0b00001000))||(!(PIND & 0b00000100))||(!(PIND & 0b00000010))||(!(PIND &
0b00000001))){
            OCR1A = 0;
            OCR1B = 0;
            lcd_clear();
            lcd_stringlong("ouch");
                            PORTE = (PINE & 0b00111111);   // clear port E motion
                     PORTE = (PINE | 0b10000000);     // give motors reverse directions
                     OCR1A = 200;
                     OCR1B = 365;
                     for (i = 0; i < 75; i++)
              {
                long_delay();   //delay to read LCD (humans reading)
              }
              PORTE = (PINE & 0b00111111); // clear port E motion
              PORTE = (PINE | 0b01000000);
              OCR1A = 365;
              OCR1B = 365;

            }

                     if (Sonarleft < 2000){
                     OCR1B = 0;
                     lcd_clear();
                     }
                     else{
                     OCR1B = 365;
                     }
                     if (Sonarright < 2000){
                     OCR1A = 0;
                     lcd_clear();
                     }
                     else{
                     OCR1A = 365;
```

```
        }
        if (Sonarleft < 2000 && Sonarright < 2000){
        // reverse
        lcd_clear();
        lcd_string("Backup!");
        PORTE = (PINE & 0b00111111);   // clear port E motion
        PORTE = (PINE | 0b10000000);     // give motors reverse directions
        OCR1A = 200;
        OCR1B = 365;
        }
        else{
        PORTE = (PINE & 0b00111111);   // clear port E motion
        PORTE = (PINE | 0b01000000);     // give motors forward directions
        }

if (analogLow < 150){

OCR1A = 0;
OCR1B = 0;
OCR1C = 0;
lcd_clear();
lcd_stringlong("You hit metal");
lcd_row(1);
lcd_stringlong("I am self aware!");
for (i = 0; i < 10; i++)
{
  long_delay();   //delay to read LCD (humans reading)
}
PORTE = (PINE & 0b00111111); // clear port E motion
PORTE = (PINE | 0b11000000);   // give motors spin directions
OCR1A = 365;
OCR1B = 365;

for (i = 0; i < 100; i++)
{
  long_delay();   //delay to read LCD (humans reading)
}
OCR1A = 0;
OCR1B = 0;
for (i = 0; i < 100; i++)
{
  long_delay();   //delay to read LCD (humans reading)
}
OCR1C = 365;
for (i = 0; i < 20; i++)
{
  long_delay();   //delay to read LCD (humans reading)
}
OCR1C = 0;
for (i = 0; i < 100; i++)
{
  long_delay();   //delay to read LCD (humans reading)
}
PORTE = (PINE & 0b00111111); // clear port E motion
PORTE = (PINE | 0b10000000);   // give motors reverse directions
OCR1A = 365;
```

```
            OCR1B = 365;
            for (i = 0; i < 100; i++)
            {
              long_delay();   //delay to read LCD (humans reading)
            }
            OCR1A = 0;
            OCR1B = 0;
            PORTE = (PINE & 0b11011111); // clear port E plow
            PORTE = (PINE | 0b00100000);   // give motors reverse directions
            OCR1C = 365;
            for (i = 0; i < 20; i++)
            {
              long_delay();   //delay to read LCD (humans reading)
            }
            OCR1C = 0;
            lcd_clear();
            lcd_stringlong("Treasure found!");
            lcd_row(1);
            lcd_stringlong("Shutting down");
            while(1){
            }
            // end of demo ;)

            }

   }

   return 0;

}


void init_adc( void )
{
  ADCSRA |= 0b01000000;   // start free running conversions
}

void config_adc(void)
{
   DDRF = 0b00000000; // set port F to all input
                // Note: when JTAGEN fuse is set, F4 - F7 don't work
   PORTF = 0x00;      // make sure pull up resistor is not enabled

   ADMUX = 0b01000000; // 5V reference, select channel0 (pin F0)
   ADCSRA |= 0b10100111; // turn on ADC, don't start conversions
                  // free funning
                  // divide clock by 128

}

void       pwm_init(void)
{//jump pins B5, B6, B7
            TCCR1A = 0b10101000;  // ......00:P&FC PWM // 11......,..11....:OC1A,OC1B inv
            TCCR1B = 0b00010001;  // ...10...:P&FC PWM // .....010:bclk/8 (~8kHz)
            DDRB = 0xFF;                    // set PWM pins as outputs, PB5 (OC1A) & PB6 (OC1B)
    ICR1=368;
```

```
        TCNT1=0x0000;
        OCR1A = 0x0000;                    // 1/2 duty, ~2.5v dc level, motor 1 on PB5 (OC1A)
        OCR1B = 0x0000;
        OCR1C = 0x0000;                    // 0/256 duty, ~2.5v dc level, motor 2 on PB6 (OC1B)
} // pwm_init
```