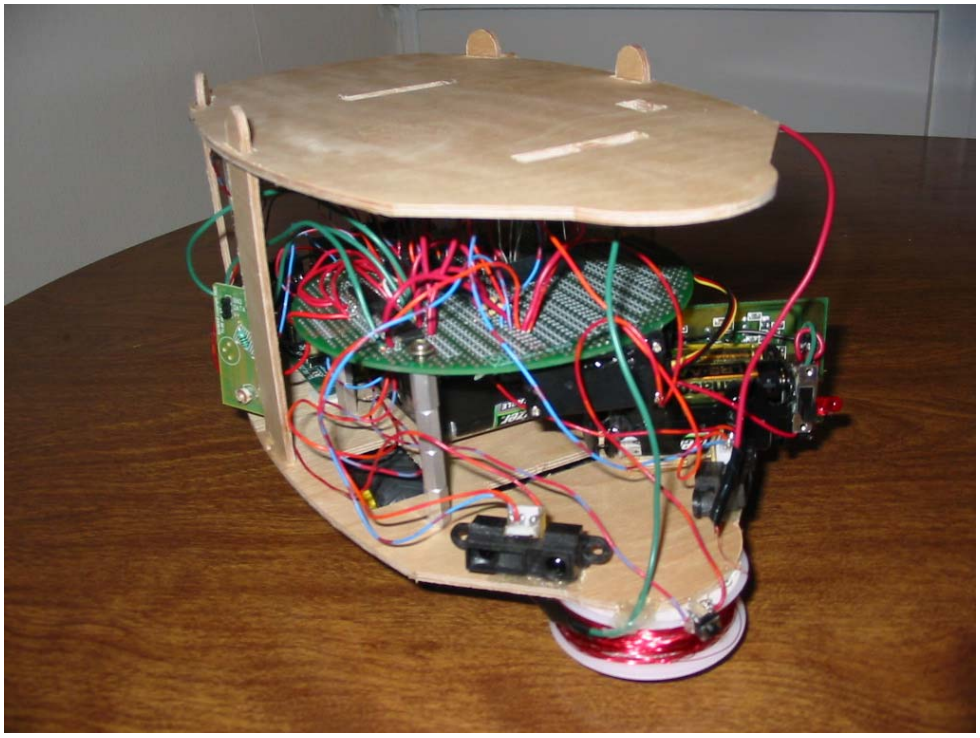
MADR : Minefield Annihilator & Detector



Scott Oetke

Final Report

EEL 5666 – Intelligent Machine Design

Summer C 2003

August 7, 2003

Table of Contents

Abstract

The purpose of this project was to create a scaled demonstration model of an actual minefield detection and elimination vehicle to provide useful experience and data for this unfortunately necessary field. The robot engages in a random walk scan of an enclosed zone and attempts to locate carbon steal mine stand-ins using a metal detection circuit. Once located, the robot deploys a sensor/battery pack that represents an explosives package, back away, and then sends a signal to the pack via radio to signify a detonation command. After being resupplied by a human, the robot then continues its search of the area.

Executive Summery

MADR was designed to be a scaled down representative of a real world automated minefield sweeper that requires minimal human intervention during this dangerous task, which can be broken down into three main segments; search, detection, and elimination. The first division, search, is where the robot defines its search pattern, containment zone, and deals with environmental hazards beyond the mines themselves. In MADR, this phase is primarily dominated by its series of three IR sensors, two bump sensors, and the main drive motors. Due to budget and time concerns, MADR uses a random search instead of one based on location knowledge, such as a GPS receiver, that a real world minesweeper would use. The second division, detection, is operated at all times to attempt to locate the mines. Once again, since this is a scaled model MADR has a single metal search loop, while a real world sweeper would use multiple search coils as well as various other sensors to detect different kinds of mines, such as ones without metallic parts. The final division, elimination, occurs once a mine has been located and marked for destruction, which is usually accomplished by using explosives to neutralize the mine. In this phase, MADR uses a servo to deploy a sensor & battery pack that is then sent a signal via RF to light an LED, symbolizing detonation. Once reloaded, MADR then continues its search for additional threats.

Introduction

Minefields are among the most dangerous and prevalent man made features across the world. Mines are cheap to produce, difficult to detect, and often far outlast the war they were meant to fight. Instead, they usually prey on innocent people long after their intended victims have faded into memory. Due to this threat, mine demolition teams are often deployed directly into suspected minefield areas with detection equipment in an attempt to identify the location of the mines. However, this job has obvious risks to the humans involved. As such, robotic counterparts are being designed in an attempt to both eliminate the risk to the human detection teams and to increase the rate of mine destruction. MADR, or Minefield Annihilation and Detection Robot, is designed to be a scaled proof of concept vehicle to explore some of the various fundamental issues that apply to such an autonomous vehicle and to provide experience in this field. This paper describes how MADR functions and includes descriptions of its various subsystems, behaviors, experimental data, and the computer code used to drive the robot.

Integrated System

MADR's main processing and search code are based on an approximately 100Hz interrupt driven sensor scan. Its overall behaviors can be summed up with the following flowchart, with the addition that during wait cycles the metal detector has higher priority & the robot will stop if the rear of the robot comes too close to an object.

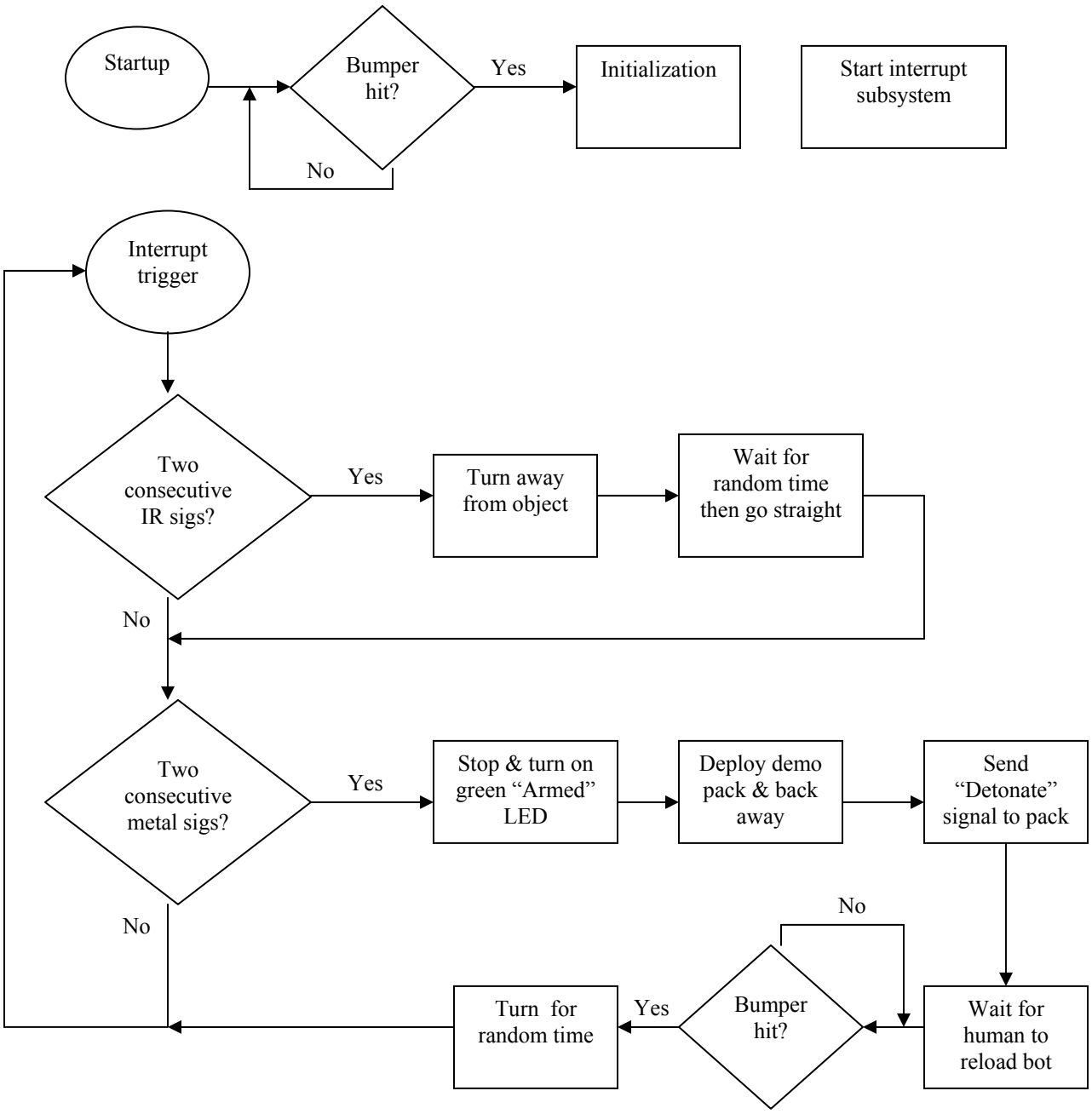


Fig. 1

Mobile Platform

The platform consists of a single, mostly oval board of plywood and a cover that is the same shape. On the rear half of the platform sits the microprocessor board (a Progressive board with an Atmega 128 core) with ample headroom for easy access to all ports and other systems. The front half primarily consists of a raised round solderable protoboard that houses the various external chips and hardware, such as the metal detection circuit and the motor driver. Below this board sits IR sensors, batteries, demolition pack, and other associated gear. On the underside of the platform rest the dual motor block, a caster riser, and the detection coil. The cover is held in place by four risers that are attached to the edges of the main plywood chassis.

Actuation

The propulsion for the robot is obtained from a twin-motor gearbox from Tamiya (Item #70097) connected to the microprocessor through a motor driver. This motor gearbox allows for a few different gear ratios through simple sprocket rearrangement. The 203:1 gear ratio was chosen for my robot primarily due to the minimal speed requirements and to ensure enough torque was available from the motors to prevent any stalling. These motors were then easily attached to the motor driver chip, which had a separate 4xAA battery pack with a common system ground to supply power for the motors, while still accepting inputs directly from the microcontroller. Finally, a simple servo mounted below the protoboard is used to deploy the demolition pack that rests in a space near the front of the machine The servo itself was a Hitec HS-311 that was fed a five volt source and a simple PWM input for position control.

Sensors

MADR consisted of various sensors that allowed the robot to execute its desired functions. These sensors are described below, each in their own section.

Special Sensor

One of the primary challenges in creating MADR was the configuration and integration of the metal detection subsystem. For the metal detector, I chose to use the On Semiconductor CS209A Proximity Detector as the core of the sensor. Attached onto this microchip is the RLC circuit that acts as the search coil, a 20K potentiometer that is used for sensitivity adjustments, and a reference capacitor. The CS209A functions by constantly feeding the reference capacitor with a constant 30µA current that is diverted away by the RLC tank during the negative half of its resonance voltage waveform. A built in voltage comparator then constantly checks the voltage across the reference capacitor with a set voltage level that is partly determined by the voltage across the potentiometer and sets the chips outputs accordingly. Whenever metal approaches the search coil the RLC circuit has its impedance changed, therefore altering how much current it diverts away from the reference capacitor, which then acts like an integrator. When enough current is allowed to charge the capacitor instead of being diverted away the voltage across the capacitor becomes high enough for the voltage comparator to flip the digital outputs. The final circuit diagram is listed as Fig. 2.

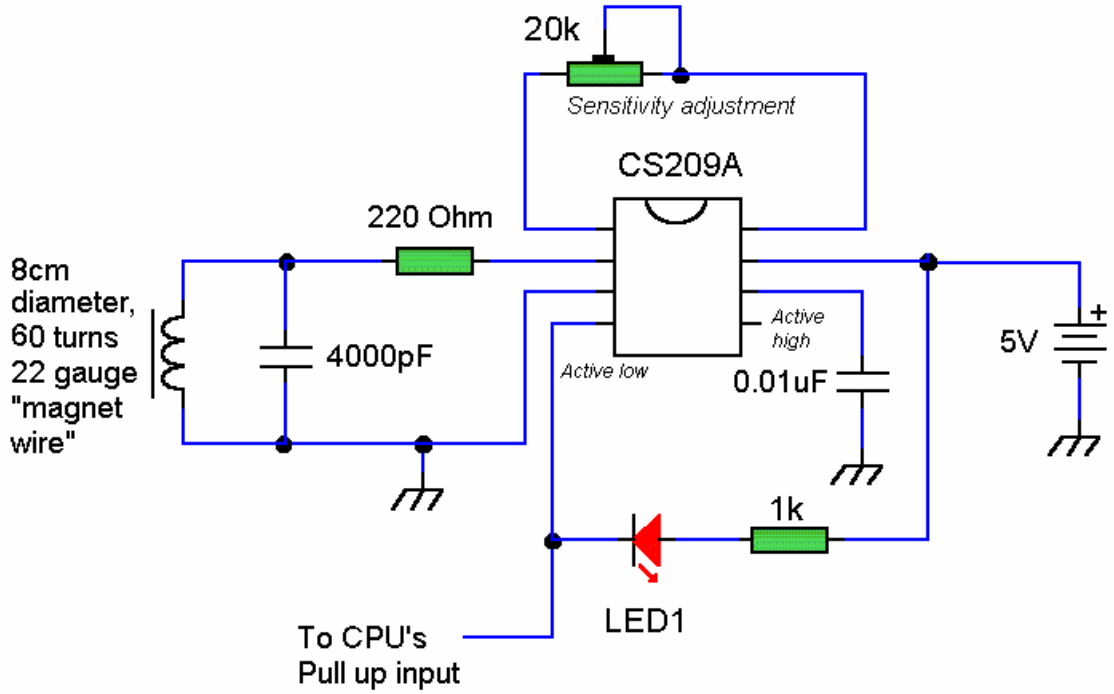


Fig.2

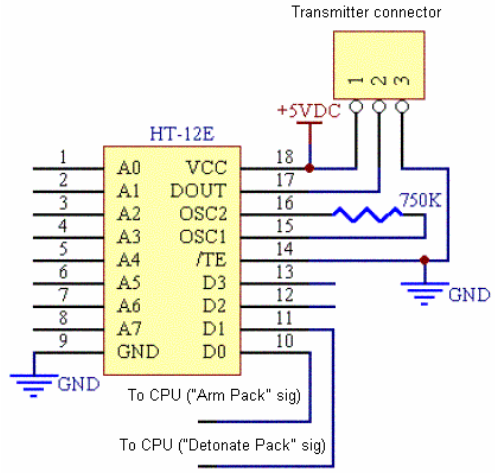
The actual procedure to use integrate the CS209A into a project primarily consists of creating an RLC circuit that has a resonant frequency within a range that the chip can detect. This feat is simplified once a reference circuit is built that can be modified and tweaked to provide the specific detection capabilities required. Also, the reference capacitor should be chosen based partly on how rapidly the user wants the circuit to change its output, though for almost all situations I was able to test the switch occurs very rapidly. Using a smaller reference capacitor allows the system to require less time to switch, but also eliminates some of the reference capacitors job of acting as a primary filter. Some changes in the ambient magnetic environment, such as steel beams in the floor and fields given off by the drive motors during switching, still triggered my detection system, so I added a simple software filter that required two consecutive positive readings from the sensor to be considered a true positive.

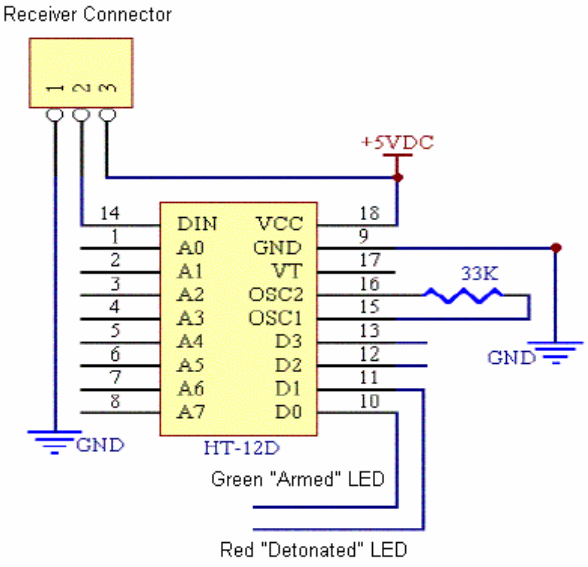
In actual use, I found this sensor to be a very practical method of metal detection with rather significant detection ranges. The primary disadvantage of this system was that a human modified potentiometer was needed to control sensitivity and detection ranges and as such I had to adjust it frequently since to create maximum sensitivity I needed to have the variable resistor set barely outside of the detection transition. Also, the fact that the chip lacked a specific analog output could limit its applications, but I do believe it would be possible to tap the non-grounded pin of the reference capacitor and use that as an analog input into an analog-to-digital converter.

RF Transmitter/Receiver

Once the demolition pack is deployed the robot required a system that could be remotely activated to show a signal to detonate was sent. For this task I decided to use an RF transmitter and receiver pair Ming RE-99 and TX-99 combined with a decoder chip (Holtec HT12D) and an encoder (Holtec HT12E). This wireless link was first used in the spring of 2003 and I borrowed heavily from the manufacturers suggestions and the previous semesters design for the link. The RF pair operates at 300MHz and are designed for short bursts of serial data at least 50 feet without adding an external antennae, though additional range can be achieved with even a basic wire antennae. These boards, which come fully assembled, require only a serial input for the transmitter and a serial decoder for the output. Instead of using my microprocessors serial connections, I purchased the Holtec encoder and decoder pair to further simplify this procedure. Finally, since my needs for the link were minimal (two bits of data that had no tight timing requirements whatsoever), I wired the transmitter to constantly send whatever data is being sent to the encoder from the mircoprocessor.

I found this wireless solution to be very easy to set up and get working in a short timeframe. While I never tested the system in any situation even approaching its full potential, it proved to be a simple, cheap, yet effective solution to MADR's humble wireless needs.





Transmitter circuit

Receiver circuit

Collision Detection Suite

The final set of sensors make up the collision detection system. For this job, I chose to use two GP2D120 30cm Analog IR sensors for the main collision avoidance. They look forward in a V shape and when their output reaches a certain hard coded hex value the robot turns away for a random amount of time and then proceeds forward again. I found that these sensors worked so well that I never actually needed a forward bump sensor for the off chance of detection miss. One problem that did crop up, however, was that since my platform was roughly an oval shape with the drive wheels at a foci the rear of the robot tended to swing wide when one motor was driving forward and the other was in reverse and as such sometimes hit walls or other objects. As such, I added another GP2D120 IR sensor on the rear facing directly away from the front of the platform and two bump sensors along the sides. When either the rear IR or bump sensors registered an object (the rear IR's threshold was set to be very close to the sensor) the robot stops its turn to prevent a collision or stalling of the motors. This addition to the collision detection suite proved to be very effective. I found the IR sensors themselves to be very reliable and provided excellent, consistent voltage readings as long as they were angled up slightly to prevent the majority of ground reflection.

Behaviors

The vehicle's main behavior is to do a random search of a confined area looking for objects to avoid and metal in front of it. The search pattern that I eventually settled on resembles a ray bouncing off objects since the robot only changes course when it detects an object. Theoretically, the robot will explore all locations within an enclosed area, except for a very small area right next to the walls and any area inside the enclosure that is too small for the robot to enter. While this random search may not be as efficient as a more explicit search pattern such as rastering the enclosure, I found it to be the best system without the extra positioning data that the more explicit patterns would require.

The metal detection subsystem gave out a digital signal, which made integrating it into the code very straightforward. The only complication I ran into was ensuring that the metal detector was being scanned very frequently even when other events were being processed. I decided against using the metal detector input as an interrupt enable and instead tested the input at the same time as when the front IR sensors were tested inside the main interrupt driven loop that operated at approximately 100Hz.

The final behavior is the detonation procedure that is executed whenever there has been two consecutive positive metal readings from the detector subsystem. First, the robot stops moving and sends out the signal to the demolition pack to arm (the first, green LED on the pack). After a hard coded time has elapsed, the robot uses the servo to push the pack from the robot. Finally, after the vehicle backs away it sends another signal to the pack to detonate (the second, red LED). The only problem I had with this procedure was that if the robot turns too fast at any point it can accidentally throw the demolition pack from the machine, but limiting its maximum speed easily prevents this.

Experimental Layout and Results

During the construction of my special sensor, I frequently tested its sensitivity by adjusting the potentiometer to a position where the system barely does not send a positive detection signal to the processor and then testing the distance certain metallic objects would trigger the circuit. During this testing I found that it was important to make sure that the position of metallic objects, such as keyboards, mice, the system batteries, and even my watch, were kept constant between tests. Even then, the detection threshold ranges tended to vary by a decent amount. However, since the detection coil rests just 1cm above the ground, such variations were not a design concern since I was always well within the detection range during actual use. The following table shows the metal detector's maximum detection range of a variety of objects in two different modes; one with the demolition pack present and one without. This made a difference since the demolition pack rests very near the search coil and as such the turnoff resistance of the circuit was different in each case. Additionally, since each object tested contained different metals and in different amounts (such as the aluminum can, which is significantly larger & more massive than a quarter), it is important to not use the data as a direct test of detectability of various metals.

	Search Sensitivity		***Maximum Sensitivity***	
	Vertical	***Horizontal***	***Vertical***	***Horizontal***
Carbon Steel	3	1.7	4.5	3.3
Quarter	1.3	0.4	3	2
Penny	1.5	0.5	3.9	2.4
Aluminum Can	3.3	1.9	7.5	6.3

Conclusion

I feel that MADR represents a good demonstration of a metallic minesweeper, but of course is not a real function robot. In addition to being made larger, more robust, and having actual explosives, additional sensors such as GPS to allow a far more efficient search procedure and laser range finders instead of IR sensors would allow it to function outside in the sun better. Also, the vehicle would most likely use a track system instead of wheels to let it travel over a diverse set of terrain. Finally, the robot would need additional sensors to handle mines at different locations & different constructions, such as all wooden bombs. That said, I believe MADR represents a good first step to learning more about this specific area of robotics.

Given the opportunity to start over, I believe the only major changes I would have done would be to make it a tracked vehicle and to replace the IR sensors with ones capable of functioning outside (since IR sensors are very easily confused by the sun). This would allow far more interesting testing situations, as well as add new depth and realism to the project. Finally, given additional time and money I would add a simple GPS receiver so the robot could intelligently raster an area determined by software, not actual walls, and as such be able to more confidently say when an area has been completely searched, an ability MADR currently sorely lacks.

Documentation

Metal detector circuit:

http://www.mitedu.freeserve.co.uk/Circuits/Misc/metaldetector.htm

Metal detector chip:

http://www.onsemi.com/

RF Transmitter:

http://www.rentron.com/remote_control/TX-99.htm

RF Receiver:

http://www.rentron.com/remote_control/RE-99.htm

Various components, including IR sensors:

http://www.digikey.com

Main drive gearbox:

http://www.pololu.com/products/tamiya/0061/

Appendices – Program code

```
#include <io.h>
#include <interrupt.h>
#include <sig-avr.h>

typedef unsigned char ubyte;
typedef char sbyte;

#define L_MOTOR_P 0    // Values on output bus A
#define L_MOTOR_N 1
#define R_MOTOR_P 2
#define R_MOTOR_N 3
#define RF_RED    6
#define RF_GREEN  7

#define MAX_L_VAL 18
#define MAX_R_VAL 67
#define MAX_L_TURN 28
#define MAX_R_TURN 64

#define L_EYE 3               // IR sensors on A2D bus
#define R_EYE 1
#define REAR_EYE 2

#define L_BUMP 3       // Bump sensor inputs into Port E
#define R_BUMP 4
#define METAL_DETECT 6 // Metal detector input into Port E

static volatile ubyte *uart_data_ptr;
static volatile ubyte uart_counter;
static volatile ubyte IR_count, metal_count, in_metal_routine;

void Metal_Detection_Routine(void);
void InitMotorPWM(void);
void InitDeployPWM(void);
unsigned char ReceiveByte( void );
void TransmitByte( unsigned char data );
void uart_string(ubyte*, ubyte);
void uart_ubyte(ubyte);
ubyte ADC_getreading(ubyte);
void InitADC(void);
void InitUART(void);
void InitCount3(void);
void Wait(int);
void RandomDir(void);

void InitMotorPWM(void)
{

        outp(0x6d,OCR0);       // Sets the maximum speed of the drive motors
        outp(0x60,OCR2);

        TCCR0 = ( (0<<WGM1) | (1<<WGM0) | (0<<CS0) | (0<<CS1) | (1<<CS2) | (1<<COM1) |
(0<<COM0) ); // No prescaler
        outp(0,TCNT0);
        TCCR2 = ( (0<<WGM1) | (1<<WGM0) | (0<<CS0) | (0<<CS1) | (1<<CS2) | (1<<COM1) |
(0<<COM0) ); // No prescaler
        outp(0,TCNT2);

        PORTA = ( (1<<L_MOTOR_P) | (1<<R_MOTOR_P) | (0<<L_MOTOR_N) | (0<<R_MOTOR_N) ); //
Start out at forward throtle

}

void InitDeployPWM(void)
{

        outp(0x81,TCCR1A); //8 bit PWM//
```

```
        outp(0x04,TCCR1B); //prescale clock by 256//
        outp(0x00,OCR1AH); //leave empty (top of count)
        outp(MAX_R_TURN,OCR1AL); //CENTER_TURN is neutral (1.5mS)
        Wait(4);
        outp(MAX_L_TURN,OCR1AL); //CENTER_TURN is neutral (1.5mS)

        // This 2ms flip is needed to prevent shaking of the servo from starting at the
final destination

}

void InitADC(void)
{
        outp(0x00,DDRF);       // A to D port, so all of Port F is input
        outp((1<<ADEN) | (1<<ADPS2) | (ADPS1),ADCSR);
}

void InitCount3(void)
{
        ETIMSK = (1<<TOIE3);
        TCNT3 = 0;
        TCCR3A = 0;
        TCCR3B = 2;
}

void InitUART(void)
{

        // 19200 baud, 8 data bits, Xon/Xoff, 1 stop bit, no Parity

        UBRR1L = 47;
        UCSR1B = ( (1<<RXEN) | (1<<TXEN) | (1<<RXCIE) | (1<<TXCIE) );
}

SIGNAL(SIG_UART1_TRANS)
{
        uart_data_ptr++;

        if (--uart_counter)
                outp(*uart_data_ptr, UDR1);
}


SIGNAL(SIG_UART1_RECV)
{
        ubyte uart1_in;
        uart1_in = inp(UDR1);

        uart_ubyte(0x0A);
        uart_ubyte(0x0D);

        uart_string("  : ADC port value",18);
        uart_ubyte(ADC_getreading(0));
}

void uart_ubyte(ubyte data)
{
        while (!(UCSR1A & (1<<UDRE))){}
        outp(data,UDR1);
}


void uart_string(ubyte *buf, ubyte size)
{
        if (!uart_counter) {
                uart_data_ptr  = buf;
                uart_counter   = size;
                outp(*buf, UDR1);
        }
}
```

```
SIGNAL(SIG_OVERFLOW3)                          // Ticks at ~100Hz
{

        cli();                                 // Disable interrupts until this one is done

        ubyte L_EyeReading, R_EyeReading;

        L_EyeReading = ADC_getreading(L_EYE); // Read the Left & Right IR sensor values
        R_EyeReading = ADC_getreading(R_EYE);
        ubyte rand = TCNT1L ^ TCNT0;           // "Random" value

        if (L_EyeReading>0x35) {
                IR_count++;

                if (IR_count >= 2) {           // If there are two consecutive positive
readings
                        outp(0x0E,PORTB);
                        PORTA = ( (0<<L_MOTOR_P) | (1<<L_MOTOR_N) | (1<<R_MOTOR_P) |
(0<<R_MOTOR_N) );
                        Wait(rand);            // Turn away for a random amount of time
                }
        }

        else if (R_EyeReading>0x35) {
                IR_count++;

                if (IR_count >= 2) {           // If there are two consecutive positive
readings
                        outp(0x0D,PORTB);
                        PORTA = ( (1<<L_MOTOR_P) | (0<<L_MOTOR_N) | (0<<R_MOTOR_P) |
(1<<R_MOTOR_N) );
                        Wait(rand);            // Turn away for a random amount of time
                }
        }

        else {
                outp(0x0F,PORTB);
                IR_count=0;                    // Else set the motors to go forward & zero
the IR count
                PORTA = ( (1<<L_MOTOR_P) | (0<<L_MOTOR_N) | (1<<R_MOTOR_P) |
(0<<R_MOTOR_N) );
        }

        if ( bit_is_clear(PINE,METAL_DETECT) ) { // Check for the metal detector sensing
an object
                metal_count++;

                if (metal_count >= 2) {               // Twice in a row
                        outp(0x01,PORTB);
                        Metal_Detection_Routine();
                }
        }

        else    metal_count=0;

        sei();                                 // Renable interrupts

}

void Metal_Detection_Routine(void)
{

        in_metal_routine = 1;          // Tell the Wait() system that we are already in
metal detection mode
        metal_count = 0;

        PORTA = ( (0<<L_MOTOR_P) | (0<<L_MOTOR_N) | (0<<R_MOTOR_P) | (0<<R_MOTOR_N) |
(0<<RF_RED) | (1<<RF_GREEN) );
        Wait(400);

        outp(MAX_R_TURN,OCR1AL);       // Turn arm to full right
```

```
        Wait(200);

        outp(MAX_L_TURN,OCR1AL);      // Then return the arm back left
        Wait(150);

        outp(0,OCR1AL);

        PORTA = ( (0<<L_MOTOR_P) | (1<<L_MOTOR_N) | (0<<R_MOTOR_P) | (1<<R_MOTOR_N) |
(0<<RF_RED) | (1<<RF_GREEN) );
        Wait(500);                    // Back up

        PORTA = ( (0<<L_MOTOR_P) | (0<<L_MOTOR_N) | (0<<R_MOTOR_P) | (0<<R_MOTOR_N) |
(0<<RF_RED) | (1<<RF_GREEN) );
        Wait(250);                    // Stop

        PORTA = ( (0<<L_MOTOR_P) | (0<<L_MOTOR_N) | (0<<R_MOTOR_P) | (0<<R_MOTOR_N) |
(1<<RF_RED) | (1<<RF_GREEN) );
        while( bit_is_set(PINE,R_BUMP) ) {};  // Wait for the rear bumper to start
execution
        Wait(75);                     // And then send the detonate signal

        RandomDir();

        in_metal_routine = 0;         // Signal we are out of the metal detection mode

}

ubyte ADC_getreading(ubyte channel)
{
        ubyte ADC_Result;
        outp((1<<REFS0)|(0<<REFS1)|(1<<ADLAR),ADMUX);

        ADMUX=ADMUX | channel;

        sbi(ADMUX,ADLAR);
        sbi(ADCSR, ADSC);

        while(!(ADCSR & (1<<ADIF))) {}        // Wait until the A2D signals that the
conversion is complete

        ADC_Result = inp(ADCH);
        sbi(ADCSR, ADIF);
        ADMUX = 0;
        return ADC_Result;            // And then return that hex value
}

void RandomDir(void)
{
                // If TCNT0 is even, turn left, else turn right
        ubyte rand = TCNT1L ^ TCNT0;

        if (rand & 0x01)
                PORTA = ( (0<<L_MOTOR_P) | (1<<L_MOTOR_N) | (1<<R_MOTOR_P) |
(0<<R_MOTOR_N) );
        else
                PORTA = ( (1<<L_MOTOR_P) | (0<<L_MOTOR_N) | (1<<R_MOTOR_P) |
(0<<R_MOTOR_N) );

        if (rand > 50)
                // Then wait for TCNT0 centiseconds (from 0 to 2.55 seconds of turning)
                Wait(rand*2);
        else
                Wait(100);

                // Finally go forward
        PORTA = ( (1<<L_MOTOR_P) | (0<<L_MOTOR_N) | (1<<R_MOTOR_P) | (0<<R_MOTOR_N) );
}

void Wait(int time)    // time is in ms (so 100 = 1 second delay)
{                      // Also checks for bumps & metal during wait cycles at around 100Hz
sample rate
```

```
        volatile int a, b, c;

        ubyte Rear_Eye;

        for (a = 0; a < time; ++a) {

                Rear_Eye = ADC_getreading(REAR_EYE);

                if (Rear_Eye > 0x85)
                        PORTA = ( (0<<L_MOTOR_P) | (0<<L_MOTOR_N) | (0<<R_MOTOR_P) |
(0<<R_MOTOR_N) | (0<<RF_RED) | (0<<RF_GREEN) );

                if ( (bit_is_clear(PINE,R_BUMP)) | (bit_is_clear(PINE,L_BUMP)) )
                        PORTA = ( (0<<L_MOTOR_P) | (0<<L_MOTOR_N) | (0<<R_MOTOR_P) |
(0<<R_MOTOR_N) | (0<<RF_RED) | (0<<RF_GREEN) );

                if (in_metal_routine == 0) {                    // If we are not already in
the metal detection routine
                        if ( bit_is_clear(PINE,METAL_DETECT) ) { // Check for the metal
detector sensing an object
                                metal_count++;

                                if (metal_count == 2) {
                                        outp(0x01,PORTB);
                                        Metal_Detection_Routine();
                                        metal_count = 0;
                                }
                        }

                        else    metal_count=0;
                }

                for (b = 0; b < 100; ++b) {
                        for (c = 0; c < 70; ++c) {}
                }
        }
}

int main( void )
{
        outp(0x00,DDRE);        // Port E is the digital input bank
        outp(0xff,PORTE);       // Enable the Port E pull-up resistors
        outp(0xff,DDRB);        // PWM output port, so all port B is output
        outp(0xff,DDRA);        // Control output port, so all port A is output
        outp(0x00,PORTA);

        InitADC();
        in_metal_routine = 0;
        metal_count = 0;

        outp(0x2F,PORTB);       // Output on the LEDs that the system has started & is
awaiting bump trigger

        InitDeployPWM();                        // Turn on servo control & move the arm into
position

        while( bit_is_set(PINE,R_BUMP) ) {};  // Wait for the rear bumper to start
execution
        Wait(110);

        outp(0,OCR1AL);                 // Turn off the deployment servo

//      InitUART();             // UART is disabled because I do not use it in normal
operation
        InitMotorPWM();
        InitCount3();

        sei();                  // Enable interrupts &, therefore, the sensors & ob avoid
routines
```

```
        outp(0x0F,PORTB);      // Output on the LEDs that we are running

        for(;;) {}
}
```