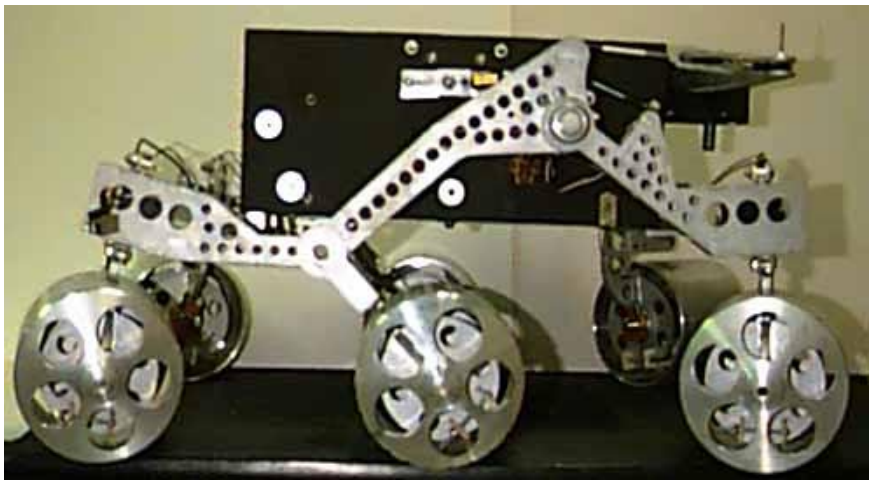
Gross Hund

All-terrain Autonomous Robot



Intelligent Machine Design Laboratory
EEL 5666
Spring 1997

Robert Pitzer
Mechanical Engineering

Table Of Contents

List Of Figures

Abstract

Gross Hund is an all terrain robot developed in the Machine Intelligence Lab at the University of Florida. It consist of a six-wheeled rocker-bogie suspension design initially develop by the Jet Propulsion Laboratory for the Mars expeditions. Each of the wheels has an internal motor to drive it and the four outer wheels are articulated by servos. It will be used in the development of low cost vision navigation systems for autonomous robots.

Executive Summary

The Gross Hund project is a robot that was built to fill the need for an all-terrain robot that is capable of transgressing inhospitable terrain. The Jet Propulsion Laboratory contributed a set of drawings of their Mars Sojourner that are the basis for the Gross Hund design. The body and suspension designs were developed in AutoCad to be prototyped on the Machine Intelligence Lab's T-tech Machine that is a two axis milling machine. The vehicle requires six motors for the drive wheels that were donated MicroMo motors in Tampa, Florida. The four outer wheel are articulated by servos and the suspension is sprung by spring loaded dampers. A ranging system is developed and will soon be integrated into the platform to enable the robot to do long range obstacle detection. The robot is controlled by two Motorola 68HC11 processors that communicate through a serial peripheral interface. The main processor controls all the higher level collision and obstacle avoidance and the slave processor generates the pulse width modulation for motor speed control and servo actuation. Preliminary computer codes for the processor has been developed and current revisions are being worked on to improve the robot's performance. Future work includes the integration of sonar and vision systems.

Introduction

The world is not flat. It is a complex environment of mountains, oceans and flatlands. The surface is covered with rocks, ridges and inclines. To develop a real world robot capable of transgressing the ever-complicated world, one must create an agent capable of moving forward despite the obstacles that may lie in the way. So to transverse this world maybe one should look at how other worlds are being conquered to find a more efficient way. The Jet Propulsion Laboratory (JPL) in California under the contract of the National Aeronautic and Space Administration (NASA) has developed many concept vehicles for the exploration of space and other planets. The past ten years JPL has had a program to develop low cost microrovers for the exploration of other planets. This project has had the title Rocky. The Mars Sojourner is derived out of the Rocky 7 vehicle. Rocky 7 is a highly mobile six-wheeled robot with the JPL patented “rocker-bogie” suspension. I decided to model my robot after this design because of its uniqueness and versatility. The design that I derived based on this model was developed for manufacture in the Machine Intelligence Lab with a few improvements to allow the rover to be more maneuverable in tight spaces. The following paper is a discussion of this robotic project. It gives a description of what I have done so far and what I hope to accomplish in the near future.

Integrated Systems

The robot's physical shape is completed and sensor and software developments are in the process. The system is comprised of a versatile platform that is capable of performing data collection and retrieval. The robot does have a processor that performs the primary mission. This processor does the data collection along with obstacle detection. It makes decisions based on its environment about whether it should progress in its current trajectory or go around an obstacle that may be in it's way. The versatility of the platform enables it to traverse small objects, so a criteria is being developed to determine what it can and cannot transgress. The following block diagram is my initial assessment for the structure of the integration.

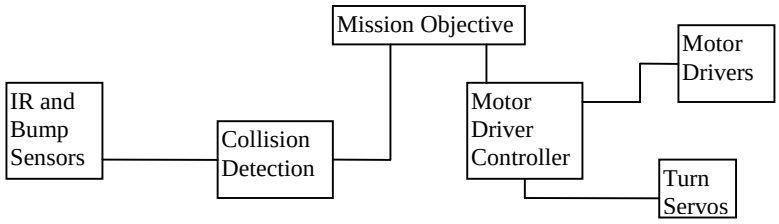


Figure 1

Mobile Platform

The platform that I have built is based on the Mars "Sojourner" rover that was built by the Jet Propulsion Laboratory for the Mars "Pathfinder" mission. JPL sent me a set of drawings that contained the geometry for their rover and I used them to come up with my own set of drawings of components that could be cut out on the T-Tech. The design is called "Rocker-bogie" based on the fact that the two main suspension supports are connected by a rocker arm that causes the two supports that are supported by a common pivot axle to move opposite of one another. This can be seen in figure 2.

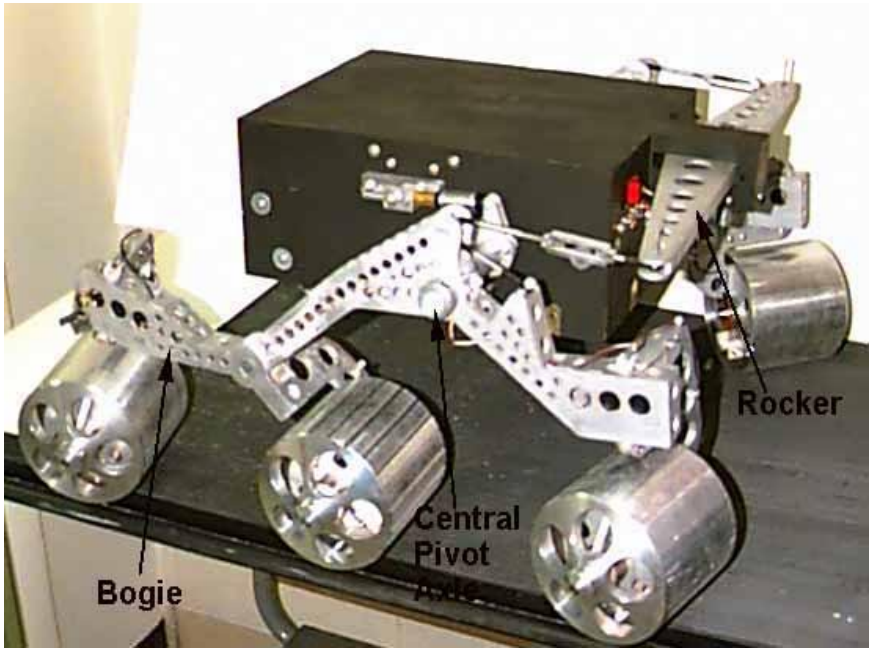
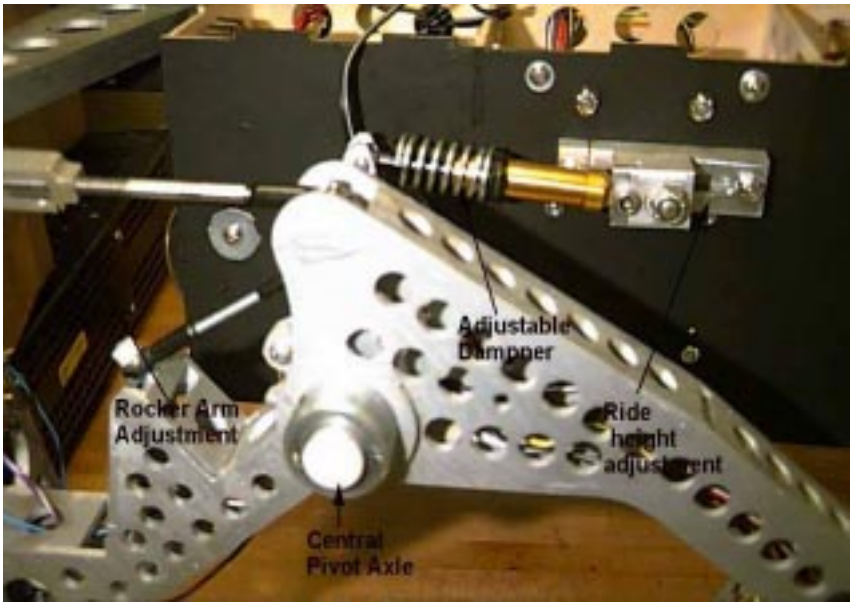


Figure 2

This causes the vehicle to keep a level aptitude as it passes over small objects. The forward bogie support is supported through the rocker arm by the opposite bogie support



and its travel is adjustable through a turnbuckle that connects it to the rocker arm. The rear wheel support is sprung through an adjustable rate dampener which is ultimately the component that determines ride height. These details can be seen in figure 3.

Figure 3

The drive motors for the robots were supplied by MicroMo motors in Clearwater, Florida and are relatively high torque motors that come in a small package. I needed the high torque to enable the robot to be able to transverse small objects because each motor should be able to theoretically lift the weight on its portion of the body. The wheels for the rover were designed by myself and supplied by the Chemistry Department Machine Shop, along with any other machining that I required. After I have finalized the motor and turning routines I plan to cover the wheels with a rubber compound, but for now I need

the slippage. The wheels were designed to encase the motors internally to allow for added ground clearance. Wheel design can be seen in figure 4.

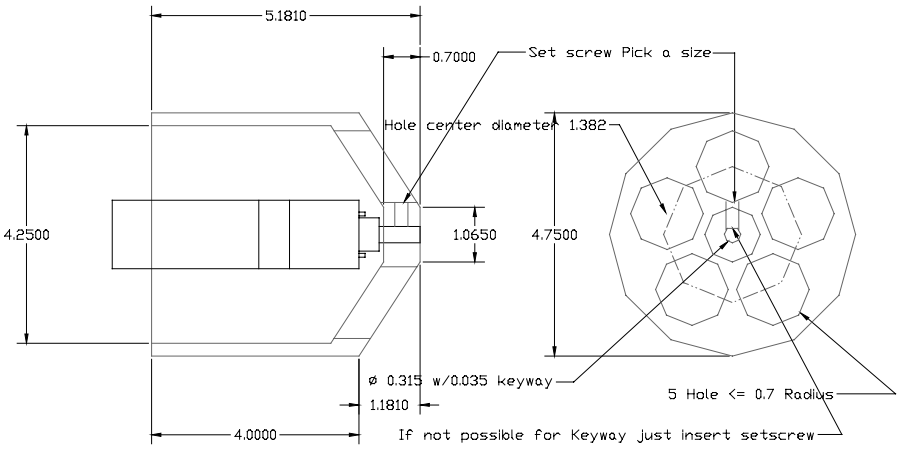


Figure 4

The rover is two feet long and one and a half feet wide. It stands a foot tall. All the parts were conceived in AutoCad and were test fitted in software before being cut out to minimize development time and the wasting of resources. Most of the parts are off the shelf items that were used for speed of final fabrication and cost effectiveness. The initial design has been tested and it has been determined that further development of the current platform is warranted. Figure 5 is one of the overall drawings of the robot that was conceived in AutoCad.

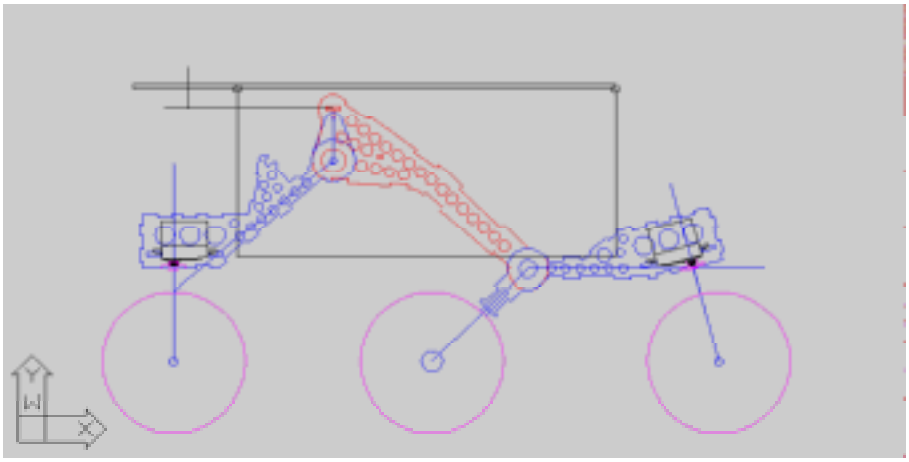


Figure 5

Figure 4 shows the basic layout of the suspension in reference to the body, it can be seen in the picture that the suspension was designed with the outer wheels' servos being integrated in the suspension design to make for a more aesthetically pleasing look. Although the suspension is made of wood it has been greatly strengthened though the use of composite design by making an epoxy sandwich. The front of the rover is to the right in the picture. The way the design is capable of pulling itself on top of objects is by the middle wheel putting enough pressure on the front wheel to enable it to climb up vertical objects and once the front wheel is on top of the object it pulls the middle wheel on top which the happens to the rear wheel as well.

Actuation

The only actuated parts on the robot are the motors and the turning servos. The motors that MicroMo are providing are planetary gearbox 20-volt dc motors. They have approximately 450 in-ounces of torque at 35 rpm, which will give me approximately nine inches per second travel rate. It also gives the wheel approximately 12 lbs. of force at the outer diameter. They have an operating temperature of –20 C to 125 C that is fine for any

environment that I will place it in. The motors are controlled by Darlington transistor and are capable of being reversed through a double-pole double-throw relay. Thie ,setup can be seen in figure 6.

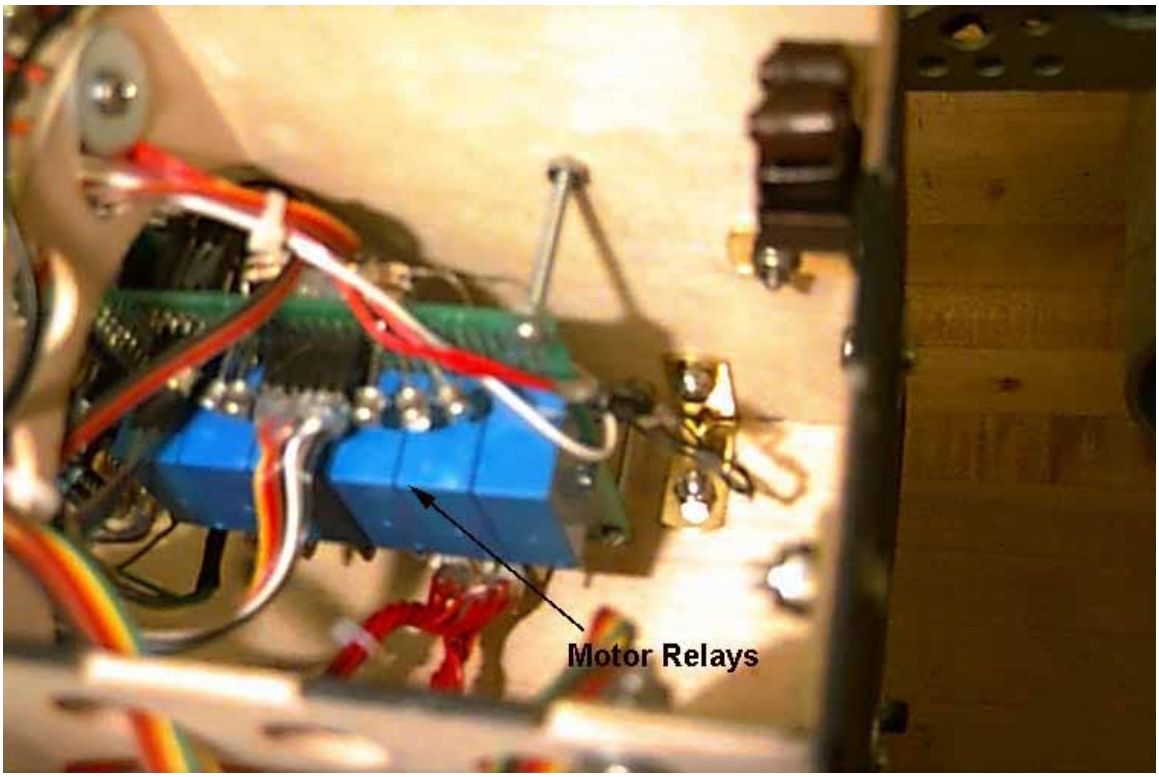


Figure 6

The servos that I am using for steering are standard Futabas with approximately 45 in-ounces of torque which need to be changed for rough terrain navigation but are sufficient for initial development. A 68HC11 single chip computer that communicates through a serial peripheral interface with a MOTOROLA EVBU evaluation board controls the motors and servos. The board layout internal to the rover can be seen in figure 7.

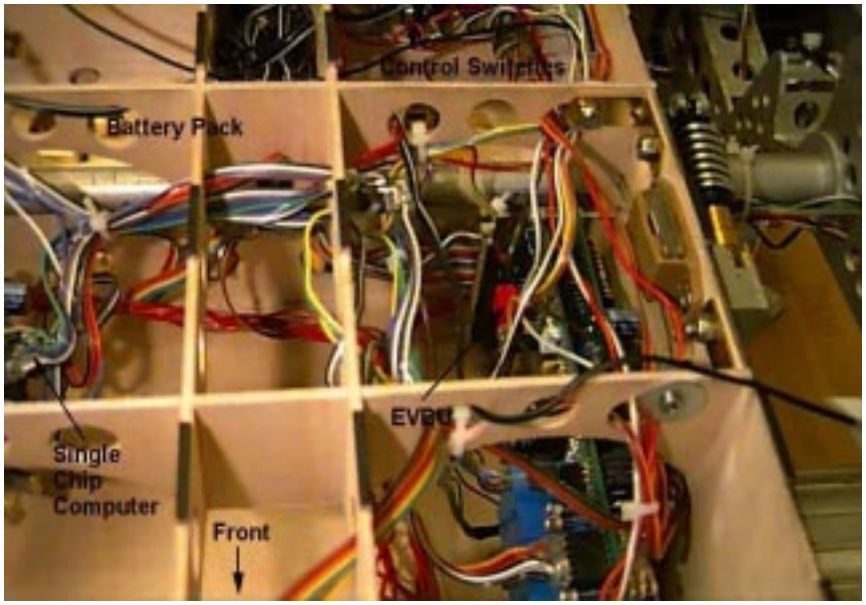


Figure 7

The interface was developed by Pervish Thakker to run in Interactive C. This enables the robot to be controlled through a tether until all the motor and servo routines have been worked out.

Sensors

At the present time the only sensors that the robot possesses are the standard sharp sensors and IR emitters. I have already been writing obstacle detection routines for the scanning IR setup that I devised and have had some luck with ranging. I am using the same basic principal that Kevin Harrelson used with his “Medusa” robot but I am trying to perform the process much faster with multiple custom developed sensors that are based on a design developed by Dan Ekdahl in the Nuclear Engineering robots lab and built by Scott Jantz and myself. Figure 8 shows the detector that was designed for the robot.

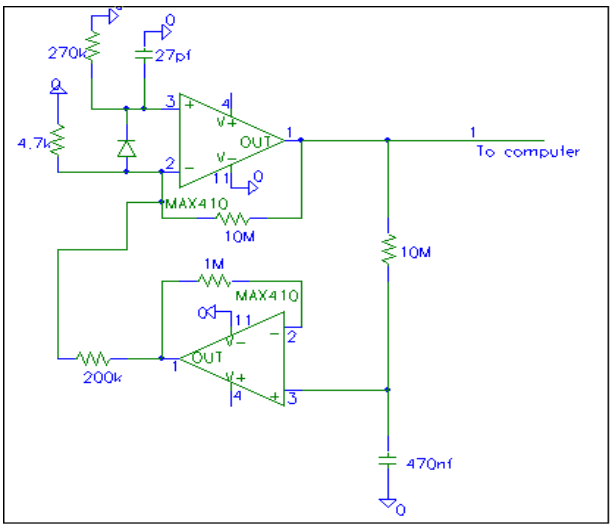


Figure 8

I have a basic design that will act sort of like a laser line scanner that I have built with two motors that spin mirrors to produce a scanning line and initial indications show that it produces a relatively good result. The robot will be able to transgress small objects and I will be able to tell if it is in bad position (one which could potentially flip the robot over) by the resistance that I am reading from the joint potentiometers that I intend to install. Through future experiments I hope to be able to define exactly what I am going to be able to do with my sensors.

Behaviors

As of now my robot is just doing basic collision avoidance, further time will be dedicated in the near future to develop complicated behaviors so that the robot will be able to accomplish its assigned task. I have already run the robot over small objects and have

determined that no special behaviors will have to be developed to allow the robot to perform this type of behavior because it performs these behaviors with no special code. I have a small CCD camera on the robot that sends back images of objects in the immediate vicinity of the robot.

Conclusion

Now that the class is finished I that one semester was not enough time to get as far as I wanted in the development of the robot. The greatest limitation that I found was time. I tried my hardest to get as much done on the robot as I could but things keep getting in the way, like classes. I did manage to put approximately 300 – 400 hrs in the project and I am pleased with the general response I am getting from people who are viewing the fruits of my labor. The robot is more study than I expected at the beginning as I have designed some ways to strengthen the design through the use of composite materials. The natural progression for the development of robots is to build agents that are increasingly more capable and I feel that the product that I produced will push the future students of the lab (Machine Intelligence Lab) to produce more capable robots, increasing the knowledge base of the lab. All other things aside though, the robot is just bad to the bone.

Appendix

Attachment 1	First Version Collision Avoidance Code
Attachment 2	Basic SPI Interface and Servo Routine Code

Attachment 1

```
/*globals*/

int milli_seconds;
int minir = 2;
int left_ir;
int mleft_ir;
int mright_ir;
int right_ir;
int left;
int mleft;
int mright;
int right;
float mults, mult = 8.;
float servo1=75.0;
float servo2=105.0;
float servo3=55.0;
float servo4=90.0;
float servo1_max=115.0;
float servo2_max=150.0;
float servo3_max=95.0;
float servo4_max=140.0;
float servo1_min=20.0;
float servo2_min=65.0;
float servo3_min=10.0;
float servo4_min=45.0;
int time;

float minf(float x, float y)
{
      if(x > y) return y;
      else return x;
}

float maxf(float x, float y)
{
      if(x > y) return x;
      else return y;
}

/*wait routine*/

void wait(int milli_seconds)
{
      long timer_a;
      timer_a = mseconds() + (long) milli_seconds;
      while (timer_a > mseconds())
      {
            defer();
      }
}


void zero_servo()
{
      set_servo(1,servo1);
```

```
      set_servo(2,servo2);
      set_servo(3,servo3);
      set_servo(4,servo4);

}

void set_servo(int x, float z)
{
      int y;

      hog_processor();
      y=degree_to_pulse(z);
      out_spi(x);
      out_spi( y>>8 );
      out_spi( y&0xff );
      out_spi(x+(y>>8)+(y&0xff));
}

void set_motor(int num, int speed, int dir)
{
      int temp;
      int temp2;

      hog_processor();
      temp=num+40;
      if (dir==1) temp=temp+8;
      out_spi(temp);
      out_spi(256 - speed);
      if (num == 1) temp2=1;
      if (num == 2) temp2=2;
      if (num == 3) temp2=4;
      if (num == 4) temp2=8;
      if (num == 5) temp2=16;
      if (num == 6) temp2=32;
      if (num == 7) temp2=64;
      if (num == 8) temp2=128;
      out_spi(temp2);
      out_spi(temp+256-speed+temp2);
}

void reset_coms()
{
      out_spi(0);
      out_spi(0);
      out_spi(0);
}

void forward(int speed)
{
      zero_servo();
      set_motor(1,speed,1);
      set_motor(2,speed,1);
      set_motor(3,speed,1);
      set_motor(4,speed,1);
      set_motor(5,speed,1);
      set_motor(6,speed,1);

}

void stop_motor()
{
```

```
	set_motor(1,0,0);
	set_motor(2,0,0);
	set_motor(3,0,0);
	set_motor(4,0,0);
	set_motor(5,0,0);
	set_motor(6,0,0);

}

void reverse(int speed)
{
	zero_servo();
	set_motor(1,speed,0);
	set_motor(2,speed,0);
	set_motor(3,speed,0);
	set_motor(4,speed,0);
	set_motor(5,speed,0);
	set_motor(6,speed,0);

}

void turn0(int speed, float servo, int direct)
{
	if(direct == 0)
	{
		set_servo(1, minf((servo1 + mults),servo1_max));
		set_servo(2, maxf((servo2 - mults),servo2_min));
		set_servo(3, maxf((servo3 - mults),servo3_min));
		set_servo(4, minf((servo4 + mults),servo4_max));
		set_motor(1,(int)((float)speed*.5),1);
		set_motor(2,(int)((float)speed*.6),1);
		set_motor(3,(int)((float)speed*.5),1);
		set_motor(4,(int)((float)speed*1.1),1);
		set_motor(5,(int)((float)speed*1.),1);
		set_motor(6,(int)((float)speed*1.1),1);

	 }
	 else
	 {
		set_servo(1, maxf((servo1 - mults),servo1_min));
		set_servo(2, minf((servo2 + mults),servo2_max));
		set_servo(3, minf((servo3 + mults),servo3_max));
		set_servo(4, maxf((servo4 - mults),servo4_min));
		set_motor(1,(int)((float)speed*1.1),1);
		set_motor(2,(int)((float)speed*1.),1);
		set_motor(3,(int)((float)speed*1.1),1);
		set_motor(4,(int)((float)speed*.5),1);
		set_motor(5,(int)((float)speed*.6),1);
		set_motor(6,(int)((float)speed*.5),1);

	  }


}
void spin(int speed, int direction)
{
	int oppo;
	set_servo(1,20.);
	set_servo(2,150.);
	set_servo(3,10.);
	set_servo(4,140.);
```

```
      set_motor(1,(int)((float)speed*1.1),direction);
      set_motor(2,(int)((float)speed*.9),direction);
      set_motor(3,(int)((float)speed*1.1),direction);
      if (direction == 1) oppo = 0;
      else oppo = 1;
      set_motor(4,(int)((float)speed*1.1),oppo);
      set_motor(5,(int)((float)speed*.9),oppo);
      set_motor(6,(int)((float)speed*1.1),oppo);
}

void main()
{
      poke(0x7000,0x0f);
      init_spi();
      start_process (avoid());
/*      while(1)
      {
            write ("right_ir = ");
            put_int(right_ir);
            write ("    mright_ir = ");
            put_int(mright_ir);
            write ("    mleft_ir = ");
            put_int(mleft_ir);
            write ("    left_ir = ");
            put_int(left_ir);
            write ("    ");
            put_char(13);

            wait (1000);
      }             */
}

void avoid()
{       /*initiate drive systems*/

      int speed = 50;
      int turn;
      float direct, time;
      forward(50);
      while(1)
      {
            right_ir = analog(0);
            mright_ir = analog(1);
            mleft_ir = analog(2);
            left_ir = analog(3);


      /*Normalize IR*/

            left = left_ir - 88;
            mleft = mleft_ir - 87;
            mright = mright_ir - 88;
            right = right_ir - 86;

            if (left < minir) left = 0;
            if (mleft < minir) mleft = 0;
            if (mright < minir) mright = 0;
            if (right < minir) right = 0;

            if (left > right)
            {
```

```
                mults = (float)left * 4.;
        }
        else
                mults = (float)right * 4.;

        time = mults ;


        if ((mleft > 18) && (mright > 18))
        {
                if ((left+mleft) < (right+mright))
                {
                        stop_motor();
                        reverse(speed);
                        wait((int)(mults * mult));
                        spin(speed,0);
                        wait((int)(mults * mult*2.));
                        forward(speed);

                }
                else
                {
                        stop_motor();
                        reverse(speed);
                        wait((int)(mults * mult));
                        spin(speed,1);
                        wait((int)(mults * mult*2.));
                        forward(speed);

                }
        }
        else if ((left+mleft) > (int)((float)(right+mright)*2.5))
        {
                turn0(speed,(float)(mults * mult * 10.), 1);
        }
        else if ((right+mright) > (int)((float)(mleft+left)*2.5))
        {
                turn0(speed,(float)(mults * mult * 10.), 0);
        }
        else
        {
                forward(speed);
        }
    }
}
```

Attachment 2

```
/*********************************************************************
*/
/*                                                                   */
/* "servo.c" for the 6.270 'bot                                      */
/*    servo support code                                             */
/* by Anne Wright                                                    */
/* Sat Jan 11                                                        */
/*                                                                   */
/*********************************************************************
*/
/* these were chosen experimentally...*/
int MIN_SERVO_WAVETIME = 650;
int MAX_SERVO_WAVETIME = 4500;
int SERVO_RANGE =(MAX_SERVO_WAVETIME-MIN_SERVO_WAVETIME);

float rexcursion = 3.14159;
float dexcursion = 180.;

void servo_on()
{
  asm_servo_on(0);
}
void servo_off()
{
  asm_servo_off(0);
}
/*********************************************************************
*/
/* Servo movement commands
*/
int servo(int period) /* argument in clock cycles of pulse, moves servo
*/
{
  if(period>MAX_SERVO_WAVETIME)
    return(servo_pulse_wavetime=MAX_SERVO_WAVETIME);
  else if(period<MIN_SERVO_WAVETIME)
    return(servo_pulse_wavetime=MIN_SERVO_WAVETIME);
  else
    return(servo_pulse_wavetime=period);

}
int servo_rad(float angle) /* argument in radians, moves servo */
{
  return servo(radian_to_pulse(angle));
}
int servo_deg(float angle) /* argument in degrees, moves servo */
{
  return servo(degree_to_pulse(angle));
}
/*********************************************************************
*/
/* Pulse width calculations
*/
int radian_to_pulse(float angle) /* argument in radians, returns pulse
width */
{
  return
((int)(angle*((float)SERVO_RANGE)/rexcursion)+MIN_SERVO_WAVETIME);
}
```

```
int degree_to_pulse(float angle) /* argument in degrees, returns pulse
width */
{
  return
((int)((angle*((float)SERVO_RANGE))/dexcursion)+MIN_SERVO_WAVETIME);
}



S12380200BB8000000B6103C84402605CEFF002003CEBF00ECE2FD8094CC8069EDE2CE10
CA
S1238040001C26801C0C80CC0BB8FD8020CE1000CC0000FD80221D0D8039CE1000CC0001
AF
S1238060FD80228614B7802439FC80222725B6802427078BFFB780242019CE1000EC0EF3
D4
S11980808020FD10161C0D801C0B801D0D808614B780247E0000B6
S9030000FC
S123872B0BB8000000B6103C84402605CEFF002003CEBF00ECE2FD879FCC8774EDE2CE10
94
S123874B001C26801C0C80CC0BB8FD872BCE1000CC0000FD872D1D0D8039CE1000CC0001
79
S123876BFD872D8614B7872F39FC872D2725B6872F27078BFFB7872F2019CE1000EC0EF3
68
S119878B872BFD10161C0D801C0B801D0D808614B7872F7E000080
S9030000FC
6811 assembler version 2.1  10-Aug-91
  please send bugs to Randy Sargent (rsargent@athena.mit.edu)
  original program by Motorola.
"servo.asm"(86): Warning --- Value Truncated
"servo.asm"(86): Warning --- Value Truncated
ADCTL      1030 *0049
ADR1       1031 *0050
ADR2       1032 *0051
ADR3       1033 *0052
ADR4       1034 *0053
BADOPINT   00f8 *0086
BASE       1000 *0006 0036 0049 0061 0084
BAUD       102b *0044
BPROT      1035 *0054
CFORC      100b *0018 0091
CMEINT     00fc *0088
CONFIG     103f *0064
COPRST     103a *0059
DDRC       1007 *0014
DDRD       1009 *0016
HPRIO      103c *0061 0015
INIT       103d *0062
IRQINT     00f2 *0083
NOCOPINT   00fa *0087
OC1D       100d *0020 0053 0090 0093
OC1M       100c *0019 0039
OPTION     1039 *0058
PACNT      1027 *0040
PACTL      1026 *0039 0038
PAIINT     00da *0071
PAOVINT    00dc *0072
PIOC       1002 *0010
PORTA      1000 *0008 0009
PORTB      1004 *0012
PORTC      1003 *0011
```

```
PORTCL     1005 *0013
PORTD      1008 *0015
PORTE      100a *0017
PPROG      103b *0060
REPEAT_PERIOD 0014 *0011 0065 0095
RESETINT   00fe *0089
RESV1      1001 *0009
RESV2      1036 *0055
RESV3      1037 *0056
RESV4      1038 *0057
RTIINT     00f0 *0082
SCCR1      102c *0045
SCCR2      102d *0046
SCDR       102f *0048
SCIINT     00d6 *0069
SCSR       102e *0047
SERVO_ADDRESS 1000 *0009
SERVO_MASK 0080 *0010
SPCR       1028 *0041
SPDR       102a *0043
SPIINT     00d8 *0070
SPSR       1029 *0042
SWIINT     00f6 *0085
SetPulse   8785 *0082 0075
TCNT       100e *0023 0086
TCTL1      1020 *0033
TCTL2      1021 *0034
TEST1      103e *0063
TFLG1      1023 *0036
TFLG2      1025 *0038
TI4O5      101e *0031
TIC1       1010 *0024
TIC1INT    00ee *0081
TIC2       1012 *0025
TIC2INT    00ec *0080
TIC3       1014 *0026
TIC3INT    00ea *0079
TMSK1      1022 *0035
TMSK2      1024 *0037
TOC1       1016 *0027 0088
TOC1INT    00e8 *0078
TOC2       1018 *0028
TOC2INT    00e6 *0077
TOC3       101a *0029
TOC3INT    00e4 *0076
TOC4       101c *0030
TOC4INT    00e2 *0075 0029 0034
TOC5INT    00e0 *0074
TOINT      00de *0073
XIRQINT    00f4 *0084
interrupt_code_exit 879e *0098 0030 0072 0080
interrupt_code_start 8774 *0070 0033
servo_count 872f *0022 0066 0074 0078 0096
subroutine_asm_servo_off 8758 *0048
subroutine_asm_servo_on 8765 *0060
subroutine_initialize_module 8730 *0024
variable_servo_enable 872d *0019 0052 0063 0071
variable_servo_pulse_wavetime 872b *0018 0043 0087

/*
  init_spi
```

```
  Initializes the SPI port on the 68HC11 to operate at 1Mhz.  This
  function must be called at the beginning of your program if you wish
  to use any of the functions in this library.

  Example:
    init_spi();
*/
void init_spi()
{
  bit_set(0x1008, 0x08);
  bit_clear(0x1008, 0x10);

  bit_set(0x1009, 0x18);
  bit_clear(0x1009, 0x04);

  poke(0x1028, 0x50);
}

/*
  out_spi

  Writes an ASCII character to the spi port.

  Examples:
    out_spi(65);
    out_spi('A');
*/
void out_spi(int outchar)
{
  int test=0;

/*  hog_processor(); */
  poke(0x102A, outchar);
  while (test == 0)
  {
    defer();
    test = peek(0x1029);
    test = test & 0x80;
  }
  test = peek(0x102a);
}
```