Group 7 Final Report

Ezra Noble

Robert Sanders

Sylas Ashton

Joseph Krause

Jesse Slater

Abstract

For EE 382, an introduction to design, our group was required to design and build a robot capable of navigating a maze and extinguishing a flame from a candle. To accomplish this we used a HC12 microprocessor to control our robot by receiving sensor inputs from motor control, flame and white line detection and proximity sensor subsystems. The final result was a fully functional robot.

Table of Contents

Introduction

The purpose and overall goal of this junior design project was to design and build a functional robot. Our robot, named CB, had to navigate a mockup of the floor of a house, locate and extinguish a candle. The report that follows describe how we broke up the design into different sections. We then show how we combined these different subsystems to produce a working robot.

Wall Following

One of the essential components to our robot was the proximity sensor. Our robot needed these sensors to navigate the maze without contacting into the walls. We considered three different sensors. The first was the GP1U52X. The appeal of this sensor was the immunity to noise interference. However, the sensor was complicated the signals output was digital, because of this we decided not to use this sensor

The next sensor we considered was the GP2D12. This sensor used IR emissions to triangulate a signal reflected from a wall. This sensor was very simplistic. The only concern was to orient the sensor in a vertical position so that it would perceive the wall correctly while navigating the maze. Since this sensor had appeal to the group, we decided to test it to compare the voltage to distance ratio (Figure 1) and see if it compared to the graph on the spec sheet.

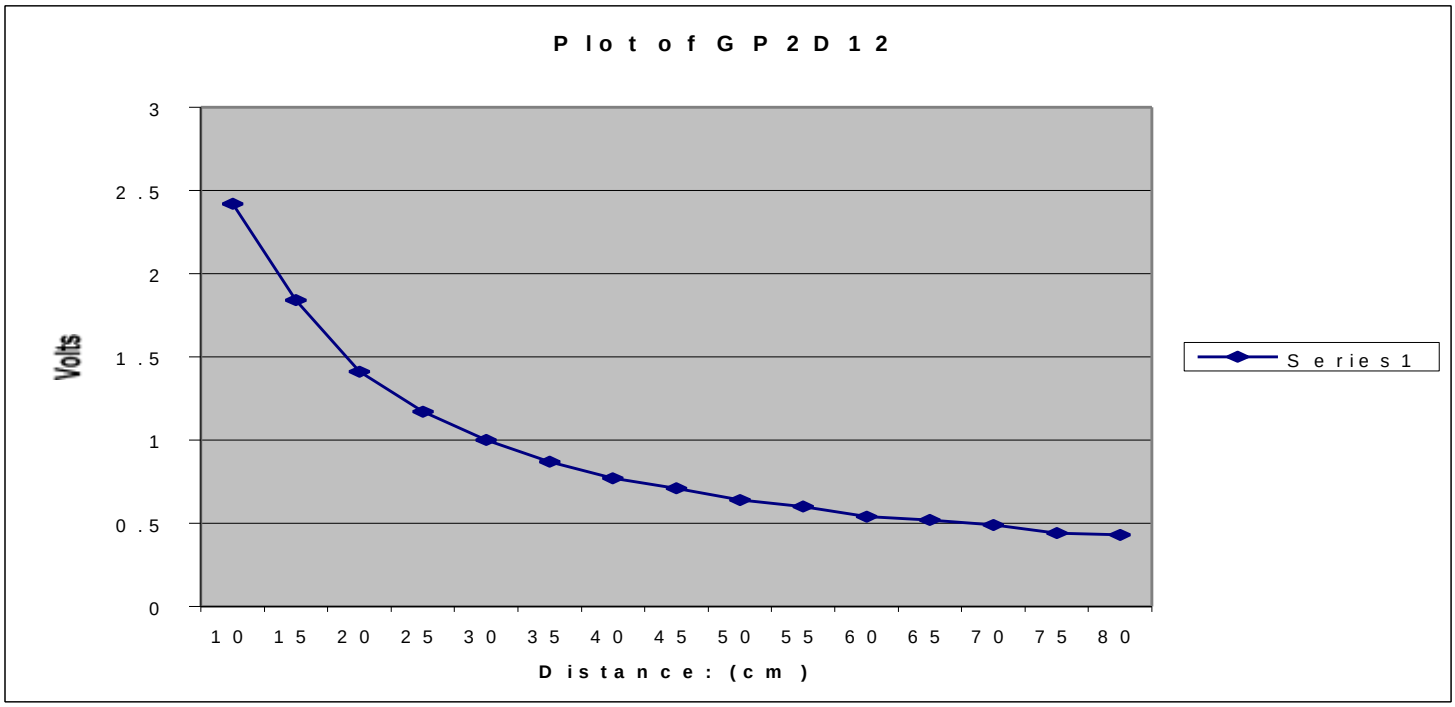
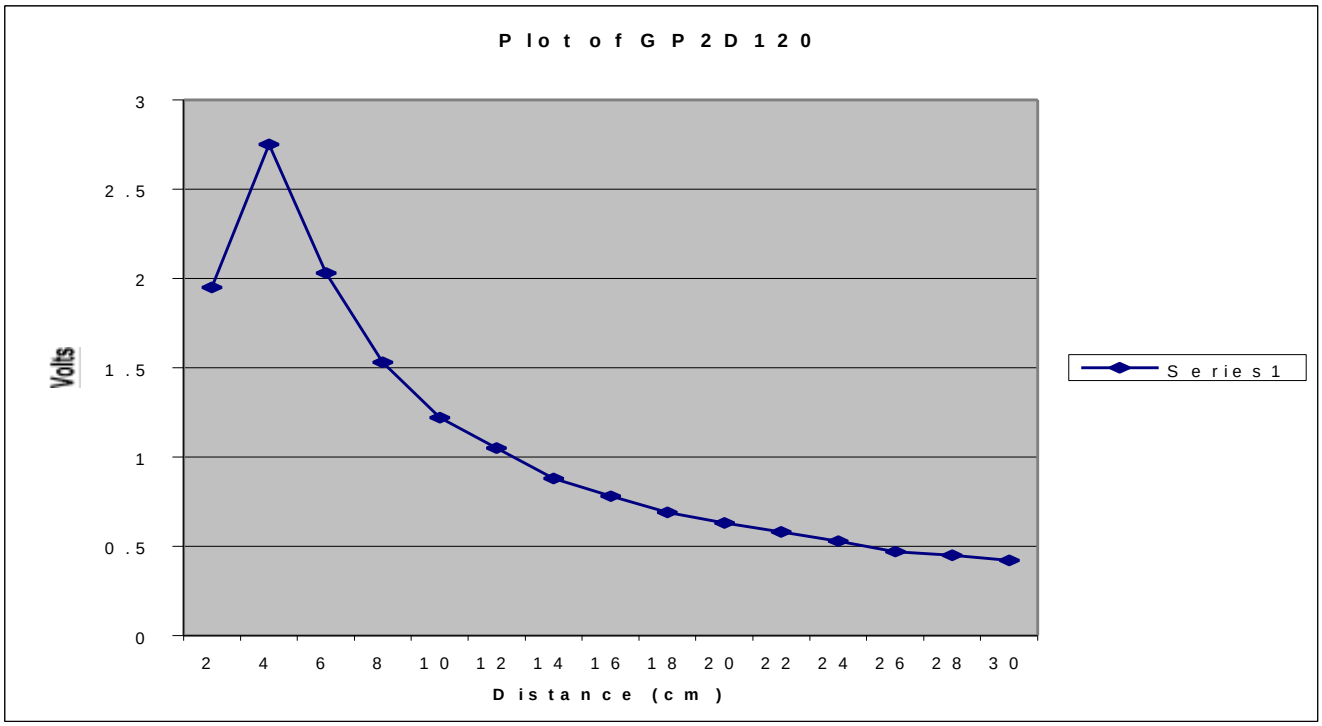


Figure 1

This graph was nearly identical to the graph on the spec sheet. A problem with the sensor was that at about 10cm the voltage peaks to approximately 2.2 volts and then sharply declines. We considered placing the sensors 10cm from the edge of the robot to avoid this problem. Unfortunately some of the components got in the way, disallowing us to implement this idea.

The next sensor was the GP2D120. This sensor was ideal for our group. At 4cm the voltage peaks at then sharply declines (Figure 2). This was corrected by setting the sensors in 4cm from the edge. We placed three of these sensors on our robot. One was positioned in the front, the other two sensors were placed behind the wheels at 45 degrees. This guaranteed that the sensors would perceive a corner and react correctly. If the sensors were placed in front of the wheel, we would have to include a delay in the

code to make the robot turn at the correct time. Since the output of the sensors was an analog signal, we connected our three sensors to three PAD pins on the HC12. This made it easy to control the robot in the maze using C code.



]**Figure 2**

Power/Signal Distribution

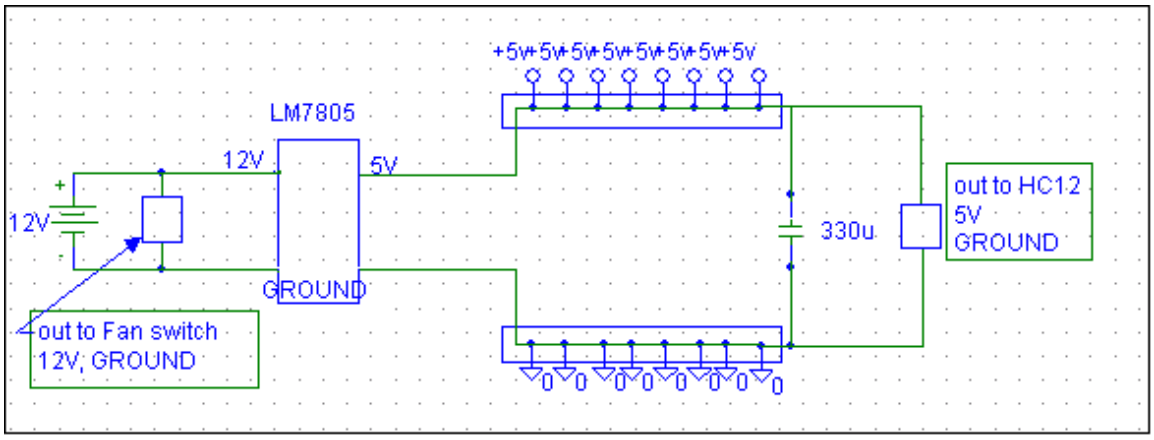


Figure 3

Considering how much emphasis is placed on having good connections for all the various wires running to the sensors and HC12, we decided to build a distribution board (Figure 3). Using locking connectors is much easier if all the wires going to a specific port or circuit are coming from the same place. It was also convenient for using shielded wires, which were available in bundles of four or eight. Incorporating all the signal and low power (5V and ground) connections on one board was fairly simple and convenient. Our final power and signal distribution circuit board prevented us from mixing specific sensor signals with other wires. We placed this board on the top of the bottom plate so as to be close to the fire and proximity sensors, but also shielded from the battery and the HC12.

Fire Sensors

Accomplishing the main objective of the Trinity Fire Fighting Competition involves putting out the fire. This is a simple enough task, once the candle is located.

Finding the candle requires sensors that can isolate a small flame. We considered many possible sensors before concluding that an infrared detecting sensor would work best for the conditions we expected.

The sensors we used were two Honeywell SDP8436's. These silicon phototransistors operate by detecting infrared emissions from the candle flame and sending out an analog signal. The signal out of these sensors varies in voltage depending on how much IR shines on them. After amplification in the circuit, a range of voltage between 0 and 4.8 volts is provided from each sensor. Under test conditions we measured .3 volts for ambient light, and up to 4.8 volts from a candle less than 3 inches away.

The best aspect of the SDP8436's is their concentrated directional acceptance cone. With a total of 18 degrees of acceptance, and a sharp drop off in voltage outside this cone, the individual sensors can precisely locate a specific source. Simple triangulation revealed that using two separate sensors placed two inches apart, the candle can be easily pinpointed with a binocular effect.

These sensors also have a light filter built into their packaging. This protects them against giving false readings based on ambient visible light. To further protect against outside light sources we mounted the fire sensor board underneath the top plate. Overall this worked very well, as false readings were avoided.

Another reason for choosing these specific sensors is that they are relatively inexpensive. At about $1.50 apiece we were able to order a single one of several different types. The idea was to test several sensors before choosing the best one. However, this plan backfired when Honeywell took almost three weeks to deliver the

sensors after we ordered them. Apparently one of the sensors wasn't in stock so they held the entire order. Rather than risk another delay we decided to use the two best sensors we had at the time. These turned out to be the SDP8436-001 and the SDP8436-004. These sensors are basically the same, with a small sensitivity difference. To compensate for the sensitivity discrepancies we used mismatched resistors along with two 10 K pots in the circuit design. (Figure 4)

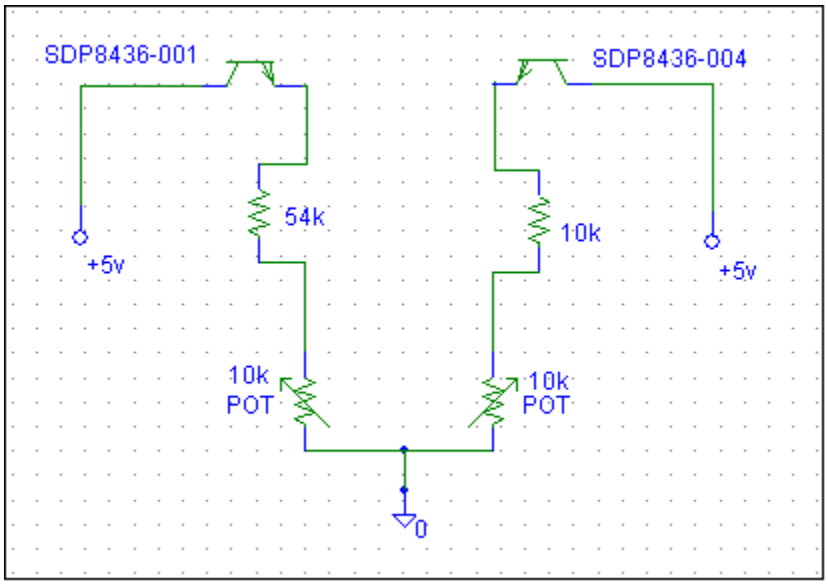


Figure 4

The range on the sensors was initially a concern. However, it was found that the candle will provide enough IR to register a significant (read >.3 volts) change in the output at a distance of 36 inches. This turned out to be a moot point though, as we ended up having to wall follow through each room. In theory we could have done a 360° scan just inside the entrance of each room, but time restraints prevented that aspect of code development.

Fire Sensing Code

The idea of the fire sensing code was to detect the fire and travel forward straight at the candle until the white line around the candle is detected. When the robot was in a room it would look for the candle. If a candle was detected the robot would turn and home in on it. To do this we used the voltage from the two fire sensors. Based on the differences of the voltages the robot would adjust the turning bias accordingly to go straight towards the flame.

Fire Extinguisher

After considering all the possible flame extinguishing techniques it was decided that a simple fan would work best. We narrowed our search to include self contained, inexpensive, 12-volt fans that would provide the most airflow with smallest current draw. We finally decided on the 92mm Panaflo from All Electronics. This fan uses less than 0.3 amps, which comes in handy when overall power is limited.

In order to turn on the fan with a single digital output from the HC12 a simple switching circuit was used. When the fire sensing code determines the robot's proximity to the flame is significant enough to turn on the fan, an active high is sent to this circuit. This activates a MOSFET, which outputs a ground signal and turns on the fan. (Figure 5)

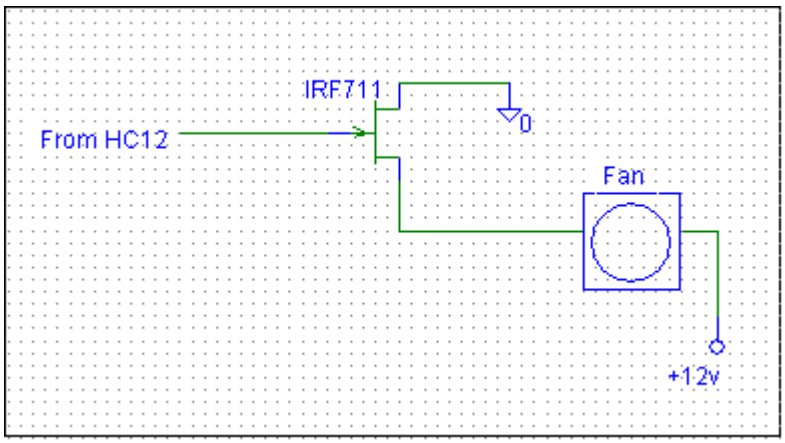


Figure 5

White line

The purpose of the white line sensor is to tell where the robot is located in the maze. For the white line detection circuit an incandescent lamp was used to illuminate the maze floor. A PN168 phototransistor detected the reflection of the lamp when crossing over a white line or the white surface of the home plate. The sensitivity of the phototransistor can easily be adjusted with a bias 10k potentiometer. The signal from the phototransistor was then fed through a 7414 Schmitt trigger to aid in dampening noise from the motors, eliminate false readings, and to provide a digital signal that was easily implemented into the HC12 (Figure 6). This circuit was then etched and attached to the bottom front of the robot. We later shielded it with tin in order to block out ambient light.

Other possible designs considered were a matching IR LED emitter/detector combination or a red LED for the illumination. The IR LED emitter/detector combination produced a very weak signal in the testing phase and would have required additional circuitry for signal amplification. Another downside to the IR LED is that it could receive other IR interference from other sensors as well as ambient lighting. The red LED was too weak in testing to illuminate the floor in order to provide a reflection for the detector.

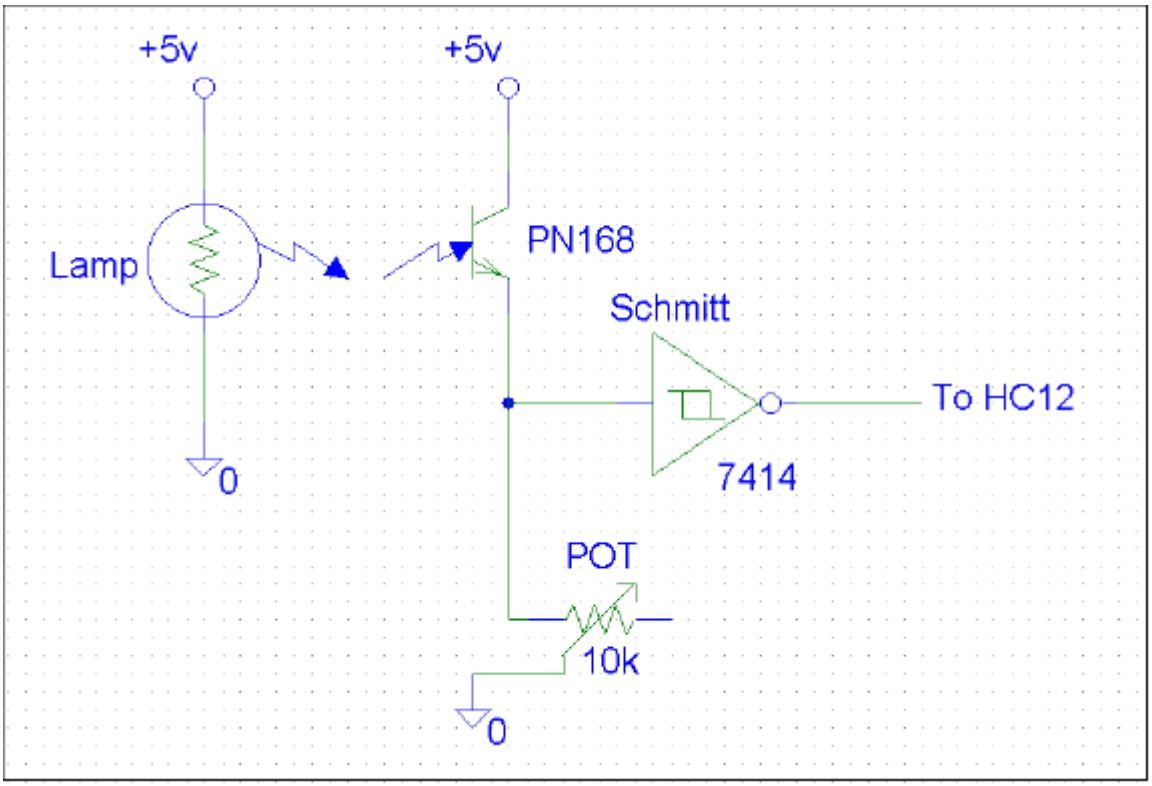


Figure 6

Closed loop motor control

The closed-loop motor control is handled inside the real-time interrupt in the HC12. Inputs to the motor control are two global integers that specify the rate of desired

travel and the rate of desired turning. It starts by detecting the actual speeds of the right and left wheels. It does this by reading the counters that run off of the right and left wheel encoders respectively. It then takes the difference between the present value and the previous values of each counter. In order to correct for turning error, the speed control algorithm needs feedback from the "leftspeed" (speed of left motor), "rightspeed" (speed of right motor) and the given input bias. This turning error is the "rightspeed" minus "leftspeed" plus bias (Appendix A). The speed of each wheel is controlled by the pulse width modulation (PWM) going to it. The PWM to the left motor is controlled by the sum of the desired speed minus the actual left wheel speed (as measured by the counters) plus the turning error. The right wheel speed is similar except the turning error is subtracted.

H-Bridge

Our robot used the Allegro 2998 H-bridge. We chose this H-Bridge to power our robot because it needed no external circuitry. It also was a dual H-bridge which meant that we did not need individual H-bridges for each motor. A third selling point for the H-bridge was that it was cheap. The H-bridge did have a drawback, as it could only supply 3 amps. We did not see this as much of a problem because we did not run the pulse width modulation at full duty cycle.

Motors

We used the Pittman Series GM9000 Brush-Commutated Gearmotors. These motors are capable of drawing 6A at 12V. The capability of our H-bridge is 3A per bridge, so we limited the duty cycle of our PWM's to 50%. At 3A, the motors produce 12.9 oz-in of

torque. The output is geared down 5.9:1 through a gearbox that is 85% efficient, giving a torque of 64.8 oz-in. With the 1.5in diameter wheels we used, this gave us 86.4oz of force per wheel. This gave us up to 10.8lb of force to keep us moving through the maze. However, at maximum, we only used 80% of this.

Mechanical

CB's chassis design consisted of a two level design with differential drive and a rear caster wheel for stability. A couple important items to note about the design was that low and high power components were separated as much as possible and the placement of the fire sensors shielded them from ambient light.

Aside from the white line and proximity sensors, which were wired with shielded cable, all low and high power components were physically separated to eliminate as much noise as possible. Also, the fire sensors were moved from the top of the robot to underneath the top level to shield them from ambient IR light. (Appendix B)

Budget

Part Description		Price	Purchased	Donated	Supplyed by group
1 1/2" Tires w/ wheels	$	2.00	X		
Caster wheel assembly	$	1.50	X		
Mechanical misc.	$	4.50	X	X	X
HC12	$	150.00			X
Battery	$	12.00		X	
Charger	$	5.00		X	
DC Power Connectors	$	2.00		X	
Fuse Holder	$	0.50		X	
SPST switch	$	2.00	X		
Fuses	$	6.00	X		
Allegro Dual H-Bridge	$	3.00		X	
Motors (Surplus x2)	$	60.00		X	
7805 Voltage Reg (x3)	$	2.10		X	
Heat Sink (T0-220)	$	0.99	X		
GP2D120 (x3)	$	19.50		X	
12VDC Cooling Fan	$	5.50	X		
SDP8436-004 (x2)	$	3.00	X		
MOSFET	$	1.33		X	
Schmitt Trigger	$	0.50		X	
PN168 Phototransistor	$	0.65		X	
Incadesant Bulb	$	1.00	X		
Copper Etch Boards	$	4.00	X		
Connectors/Headers	$	21.57	X	X	
Resistors/10kPOTs	$	4.00		X	
Total Against Budget			**$ 48.36**		
Grand Total	**$**	**312.64**			

Conclusion

At the beginning, we wanted to keep the robot very simple. To accomplish this we built our subsystems, so that they needed as little external circuitry as possible. This enabled us to have time to refine the code. This idea worked for the most part, we had all the subsystems working individually in a relatively short time. The major problem that slowed us down was the HC12. We had many memory and operational problems. It was difficult to get consistent behavior out of the HC12 from day to day. Looking back at what happened, we should have tested each HC12 board before implementing the subsystems.

This class provided valuable experience to our group. As a group we learned how to delegate individual tasks. As individuals we learned how to design our subsystems and implement the subsystems in to the project.

References:

"Mobile Robots: Inspiration to Implementation" 2nd edition
Joseph Jones, Anita Flynn, Bruce Seiger

Internet:

http://www.ee.nmt.edu/~wedeward/EE382/SP00/gm9434h187_r1.pdf
http://www.ee.nmt.edu/~wedeward/EE382/SP00/allegro_2998.pdf
http://www.ee.nmt.edu/~wedeward/EE382/SP00/gp2d120.pdf
http://www.honeywell.com/
http://www.nationalsemiconductors.com/
http://www.allelectronics.com/
http://www.trincoll.edu/events/robot/images/arena.gif

Advisors:

Dr. Kevin Wedeward

Dr. Stephen Bruder

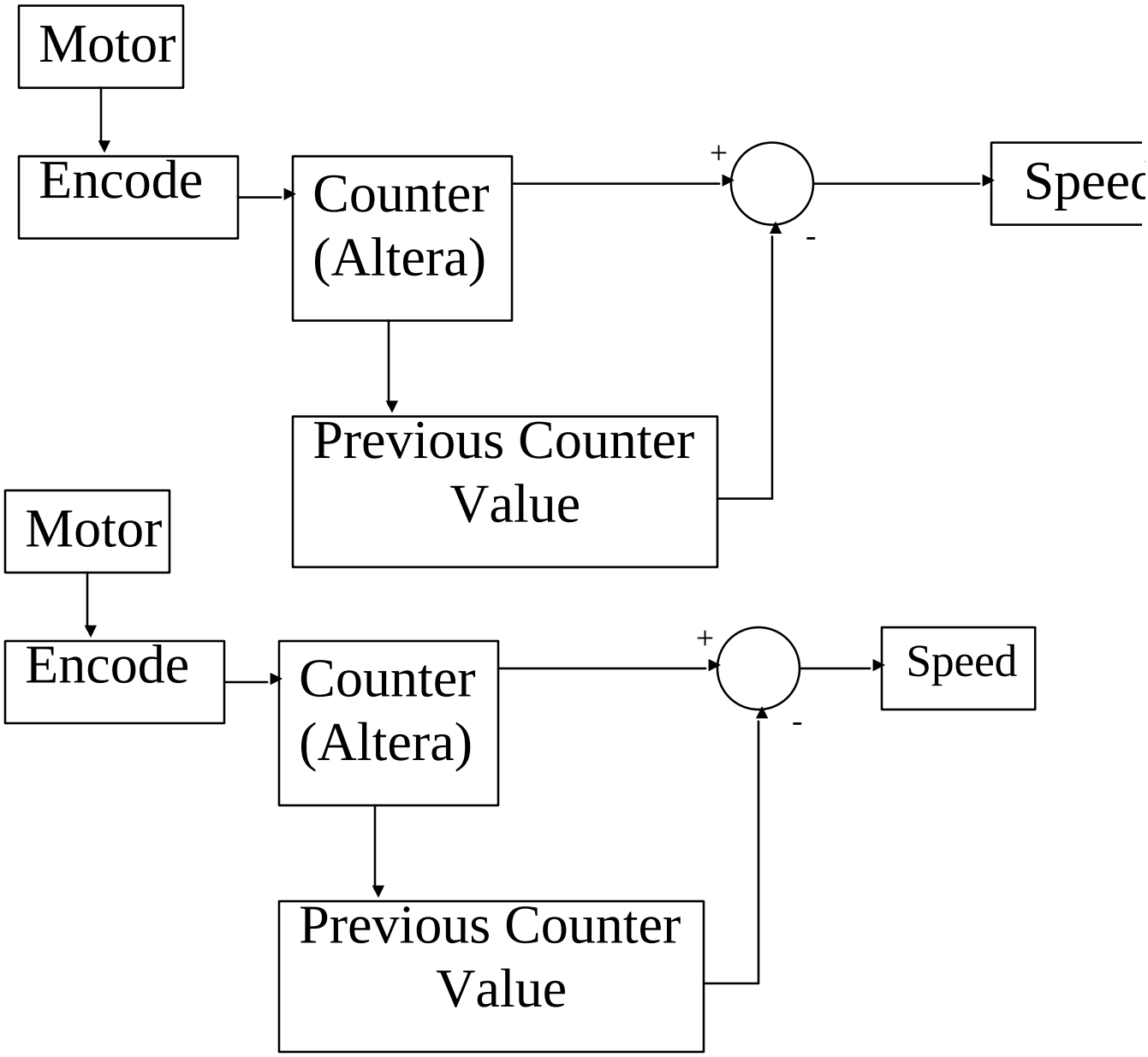
Dr. William Rison

Text:

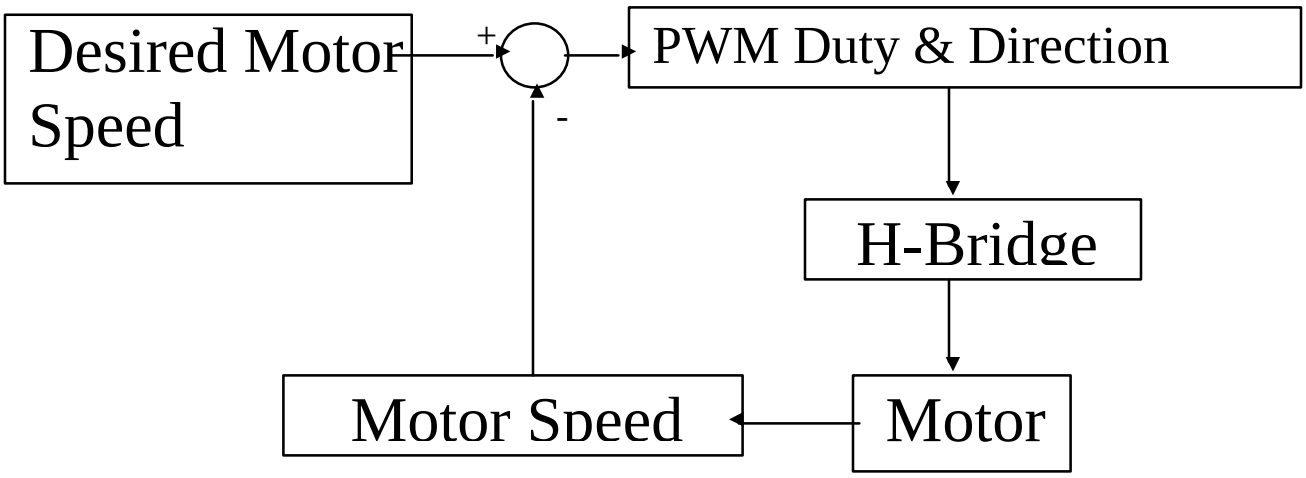
Mobile Robots: Inspiration to Implementation

Appendix A

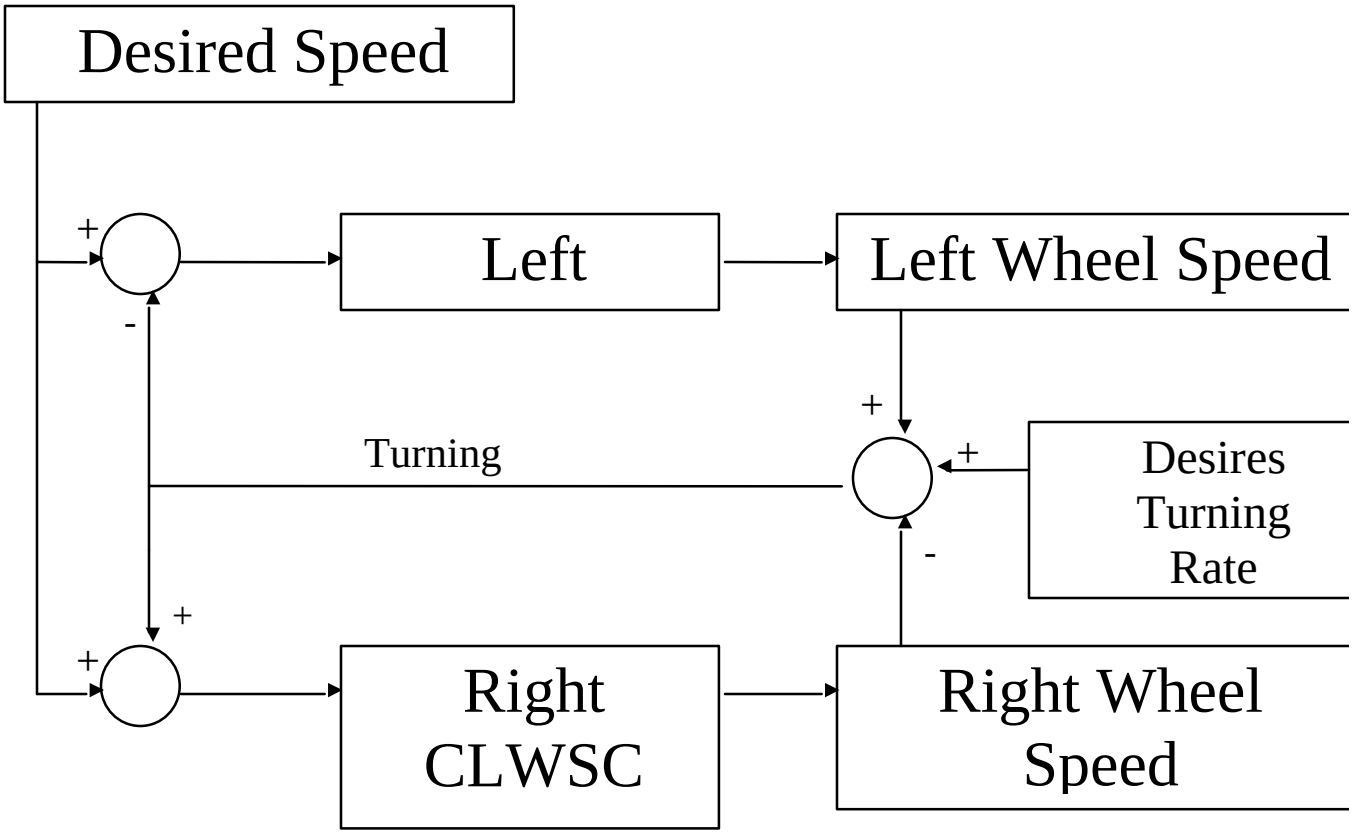
Motor Speed Detection



Closed Loop Wheel Speed Control (CLWSC)

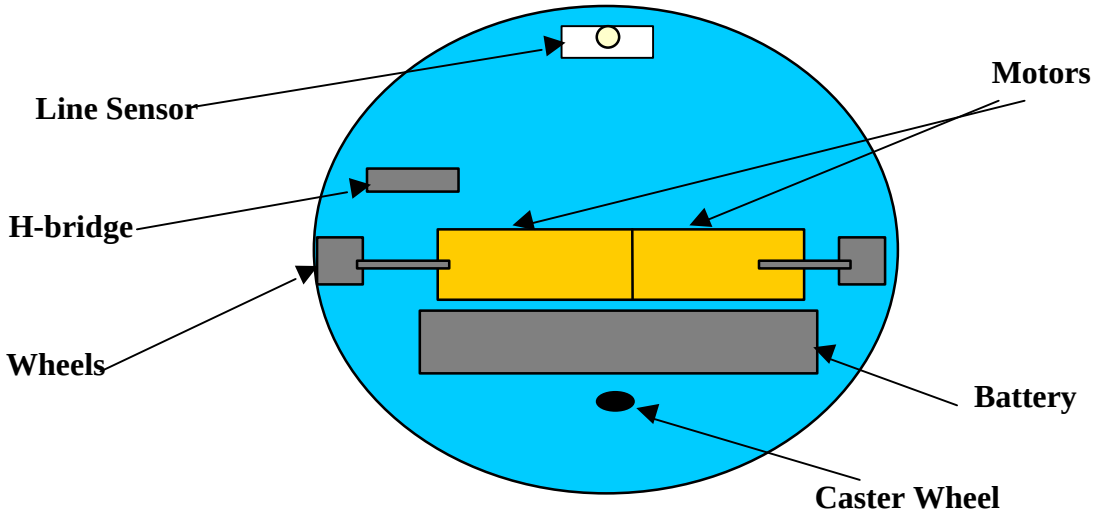


Closed Loop Wheel Control Algorithm

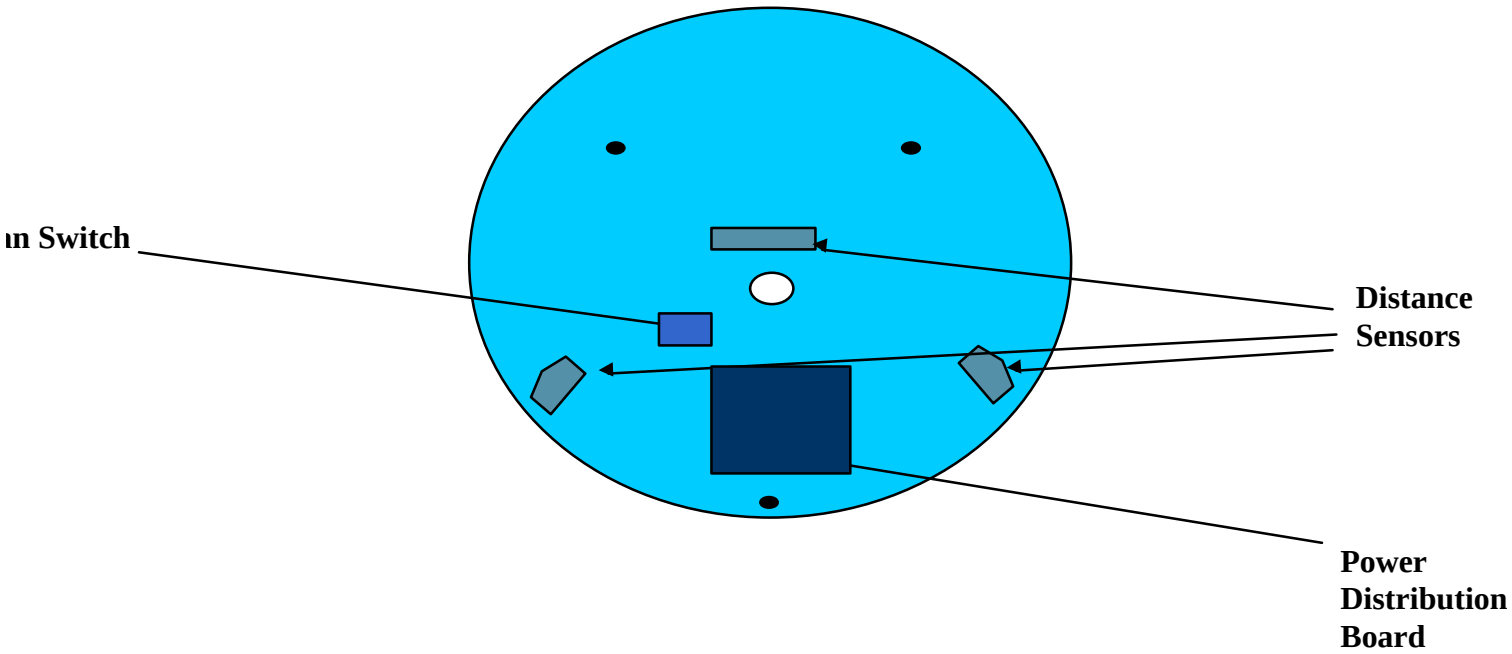


Appendix B

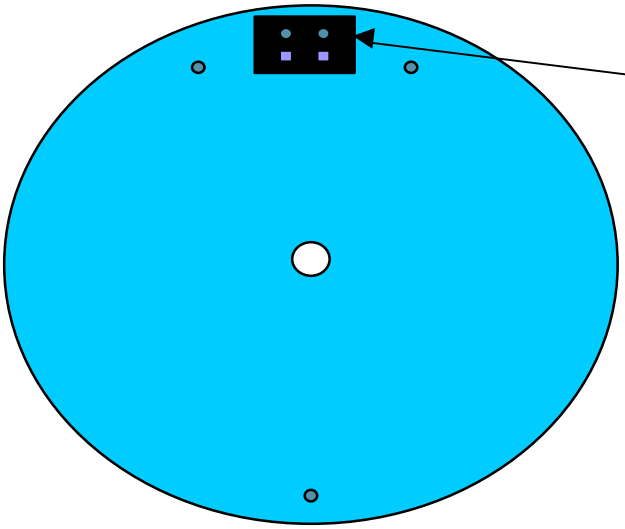
Bottom of Lower Plate



Top of Lower Plate



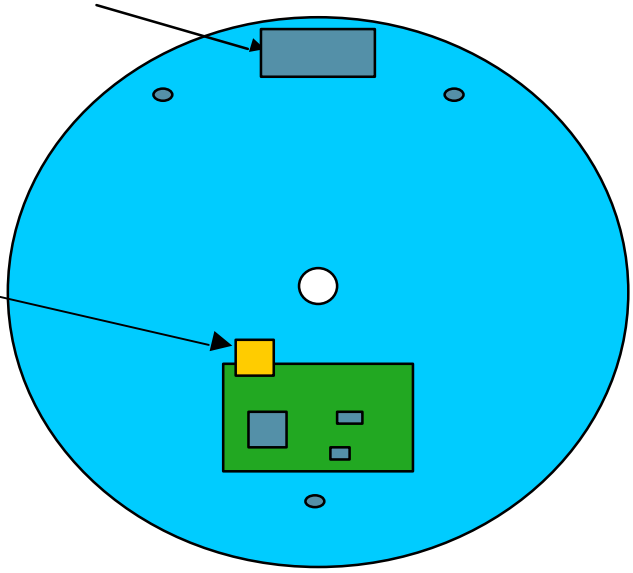
Bottom of Upper Plate



Fire Sensors

Top of Upper Plate

Extinguisher



Push Button Start

HC12

Main Program (rf4.c)

```
#include <1.c>
main()
{
        #include <2.c>
        PORTP&=~0x10;
        while(!startsig);               /* Wait For Pushbutton Start */
        while(whiteline)                /* While on Home Leftwall Follow */
        {leftwall();}
        start=1;                        /* Start Looking for whiteline */
        while(!dead&(count<6))          /* While Fire not out and not in back to home */
        {
                if((count==1)||(count==3)||(count==5))
                        /* Number of whitelines to be in a room */
                {inroom();}
                else                    /* Otherwise in a hall */
                {inhall();}
                /* Print number of whitelines we have seen for whiteline debugging */
                DBug12FNP->printf("%d--%d\r",whiteline,count);
        }
        while(count<6)          /* If the fire but not at home continue leftwall following */
        {       DBug12FNP->printf("%d--%d\r",whiteline,count);
                leftwall();}
        start=0;        /* Disables Whiteline counter so to not double count home */
        while(!whiteline)               /* Leftwall follow until back to Home */
        {
                DBug12FNP->printf("%d--%d\r",whiteline,count);
                leftwall();
        }
        /* Turn around code at whiteline */
        while((front>(da+3)))  /* When at Home turn right until wall close*/
        {
                speed=0;
                bias=130;
                front=255-ADR4H;
        }
        while((front<(da+14)))                  /* Now turn until the wall is far away */
        {
                speed=0;
                bias=130;
                front=255-ADR4H;
        }
        start=1;        /* Re-enable whiteline counter */
        while(count==6)                 /* While no whitelines in the hall */
        {       DBug12FNP->printf("%d--%d\r",whiteline,count);
```

```
            inhall();
        }
        while(count==7)                 /* Leftwall follow into 4th (island) room */
        {
            DBug12FNP->printf("%d--%d\r",whiteline,count);
            inroom();
        }
        while((count==8)&!whiteline)        /* Leftwall follow home */
        {               DBug12FNP->printf("%d--%d\r",whiteline,count);
            leftwall();
        }
        while(1)                            /* Stop at home */
        {
            speed=0;
            bias=0;
        }
}
inhall()                        /* Only Leftwall follow when in the hallway */
{leftwall();}
inroom()                                            /* In Room Code */
{       if((rightfire>30)||(leftfire>30))       /* If see fire call fire kill function */
        {kill();}
        else                    /* If no fire then leftwall follow through room */
        {leftwall();}
}

#include <3.c>
```

Other Programs

1.c

```
#include <DBug12.h>
#include <hc12.h>

#include <wheeldriversetup.c>

#define da 220                      /*general offset distance from walls */
#define kwall 4                     /*left wall follow turning multiplier */

#define whiteline !((PORTP&0x80)==0x80)        /*defines the whiteline pin*/
#define startsig ((PORTP&0x020)==0x20)         /*defines pushbutton start pin */
char front,left,right,white=0,dead=0,count=0,turn=0,start=0,set=0;
                            /*declares misc. variables */

#define rightfire ADR6H     /*defines right fire sensor pin */
#define leftfire ADR5H      /*defines left fire sensor pin */
int I;                      /*for loop variable */
```

2.c

```
#include <wheeldriversetup2.c>

ATDCTL2=0x80;      /*set A/D converters for multiple input continuous scanning */
ATDCTL4=0x01;
ATDCTL5=0x70;

DDRP &=~0x80;               /*sets whiteline pin input*/
DDRP|=0x10;                 /*sets fan pin output*/
```

3.c

```
#include <wheeldriver.c>

void leftwall()                         /*leftwall follow function */
{
        front=255-ADR4H;                /*front sensor calibration */
        left=255-ADR3H;                 /*left sensor calibration */

        if(front>(da+12))       /*if front greater than desired distance follow left wall */
        {
                speed=80;
                bias=(int)(kwall*(da-left+3));
        }
        else            /*otherwise turn right until front distance is greater than before*/
        {
                do
                {
                        front=255-ADR4H;
                        speed=30;
                        bias=100;
                }while(front<(da+17));
                /*keep turning until distance is far enough to continue leftwall following*/
        }
}
void kill()                             /*kill the fire function */
{       start=0;                        /*disable whiteline*/
        while((rightfire>30)||(leftfire>30))    /*while either fire sensor see the fire */
        {
                int i,j;
                if(whiteline)           /*whiteline?Y creep in and put out the fire */
                {
                        while((rightfire>30)||(leftfire>30))
                        /*keeps program in loop till the fire is out */
                        {
                                speed=23;
                                bias=(4*rightfire-9*leftfire)/9;
                        /*go at fire unequal due to sensor differences */
                                PORTP|=0x10;            /*turn fan on */
                                for(i=0;i<1000;i++)     /*creep in 10% of the time */
                                {
                                        for(j=0;j<600;j++)
                                        {bias=0;}
                                }
                                speed=0;
                                for(i =0;i<9000;i++)    /*stay still 90% of the time */
```

```
                        {
                                for(j=0;j<600;j++)
                                {bias=0;}
                        }
                        PORTP&=~0x10;                   /*turn off fan */
                        speed=0;
                        bias=0;
                        for(i=0;i<10000;i++)            /*wait and see if fire is out */
                        {for(j=0;j<500;j++){}}
                }
                dead=1;                 /*tells programming fire is out */
        }
        else                                                    /*no fire whiteline*/
        {
                front=255-ADR4H;
                left=255-ADR3H;
                if((left>da)&&(front>da))       /*make sure to not hit a wall*/
                {
                        speed=40;
                        bias=(2*rightfire-9*leftfire)/9;
                                /*got towards the fire*/
                        PORTP&=~0x10;       /*makes sure fan is off*/
                }
                else
                {leftwall();}           /*leftwall follow if to close to a wall*/
        }
}
if(dead)                                /*if fire extinguished*/
{       int fronttemp;
        front=255-ADR4H;
        fronttemp=front;                /*get current front distance*/
        while(front<(fronttemp+8))
                /*turn around by turning until front is greater than distance it was*/
        {       speed=0;
                bias=130;
                front=255-ADR4H;
        }
        while((front>da)&(left>da))
                /*go forward and turn left some until see a wall */
        {
                speed=60;
                bias=-10;
                front=255-ADR4H;
                left=255-ADR3H;
        }
}
```

```
        start=1;                                /*reenable whiteline counter */
}
@interrupt irq()                                /*unused irq interrupt detection */
{       int j;
        if(start)
        {       count++;
                for(i=0;i<60000;i++)
                {
                        for(j=0;j<10000;j++)
                        {       speed=0;
                                bias=0;
                        }
                }
                while(whiteline)
                {leftwall();}
        }
}
```

wheeldriversetup.c

```
#define PORTC *(char *)0x500        /*left wheel counter address*/
#define PORTD *(char *)0x502        /*right wheel counter address*/
#define kpro 1                      /*feedback multiplier*/
#define kintegral 1                 /*feedback multipiter*/
#define kspeed 1                    /*speed multipier*/
int error=0;                        /*feedback variable*/
unsigned int lspeed=0;              /*left wheel speed*/
unsigned int rspeed=0;              /*right wheel speed*/
volatile char Done=0;               /*timing signal*/
int speed=0;                        /*desired speed*/
int bias=0;                         /*desired turning bias*/
int newL, newR,oldL=0;int oldR=0;   /*new counter values,old counter values*/
```

wheeldriver.c

```
wheel_control()                 /*Porportional Control function*/
{
        static int integral=0;          /*initializes turning error value*/
        integral=limit(lspeed-rspeed+bias);             /*sets turning error value*/
        error=kintegral*integral;                       /*turning error multiplier*/
        alterpower(((speed-rspeed-error)),(char) 1);
                                        /*changes pwm and direction of right wheel*/
        alterpower(((speed-lspeed+error)),(char) 0);
                                        /*changes pwm and direction of left wheel*/
}
int limit(int a)                /*limits turning error so no chance of overflow*/
{
        if(a>1000)
        {a=1000;}
        if(a<-1000)
        {a=-1000;}
        return(a);
}
alterpower(int powerin,char motor)  /*changes pwm and direction each wheel*/
{
        char forward;
        powerin=limitpower(powerin);            /*limits magnitude of pwm*/
        if(powerin<0)                                   /*finds desired direction*/
        {
                powerin=-powerin;
                forward=0;
        }
        else
        {forward=1;}
        if(motor==1)                    /*if right motor-change right motor pwm*/
        {
                PWDTY0=(char) powerin;
                if(forward==1)
                {PORTP&=~0x04;}
                else
                {PORTP|=0x04;}
        }
        else                            /*if left motor-change left motor pwm*/
        {
                PWDTY1=(char) powerin;
                if(forward==1)
                {PORTP&=~0x08;}
                else
```

```
                {PORTP|=0x08;}
        }
}
int limitpower(int powerin)
                        /*limits pwm to 50% duty cycle(current restrictions on h-bridge)*/
{
        if(powerin>(PWPER0/2))
        {powerin=(PWPER0/2);}
        if(powerin<(-PWPER0/2))
        {powerin=(-PWPER0/2);}
        return(powerin);
}
int abs(int a)                          /*takes absolute value of a*/
{
        if(a<0)
        {a=-a;}
        return(a);
}
@interrupt rti_isr()                    /*real-time interrupt*/
{
        /*static char i;*/
        newL=PORTC;                     /*reads left encoder counter*/
        newR=PORTD;                     /*reads right encoder counter*/
        lspeed=(kspeed*(newL-oldL));        /*calculates left and right speeds*/
        rspeed=(kspeed*(newR-oldR));
        oldR=newR;                      /*stores counter values for next speed calc*/
        oldL=newL;
        wheel_control();                /*Porportional control function*/
        if(start==1)                    /*White Line Detection Enabled?*/
        {
                if(rightfire<150)       /*Diables Counter in case of fire*/
                {
                        if(whiteline)   /*Avoids Multiple Sampling of Whiteline*/
                        {white=1;}
                        if(white&!whiteline)
                        {
                                if(count<10)
                                {count++;}
                                if(count==9)
                                {count=0;}
                                white=0;
                        }
                }
        }

        RTIFLG=0x80;                    /*clears rti flag*/
```

```
}
```

wheeldriversetup2.c

```
PWCTL=0x00;          /*pwm control register desired set */
PWEN=0x03;           /*pwn enable channel 0 and 1 */
DDRP|=0x0f;          /*sets 2 pins of direction*/
PWPER1=200;          /*sets the pwm period  */
PWPER0=200;          /*set periods same*/
PWCLK=0x30;          /*sets clock speed*/
PWPOL=0x00;          /*active low polarity for active low outputs*/

RTIFLG=0x80;         /*clear rti flag*/
RTICTL=0x84;         /*activates rti interrupt, every 8.2ms*/

INTCR=0xa0;          /*disables IRQ interrupt*/


enable();            /*enables interupts*/
```

rf4.lkf

```
#      link command file for test program
   #
   +seg .text -b 0x1000 -n .text   # program start address
   +seg .const -a .text           # constants follow code
   +seg .data -b 0x7000           # data start address
   crts.o                      # startup routine
   rf4.o                      # application program
   +seg .const -b 0x0b10          # vectors start address
   vector.o                    # interrupt vectors
   +def __stack=0x0a00            # stack pointer initial value
```