EE 382 Introduction to Junior Design

Medusa The Fire-Fighting Robot

by

Ahmed Barradah
Lawrence Landon
Timothy Miller
Robert Rose

May 11, 1999

Table of Contents

List of Figures and Tables:

References

LM7805, Voltage Regulator, National Semiconductor, http://www.national .com/design/index.html.

LM317, Voltage Regulator, National Semiconductor, http://www.national.com/design/index.html.

L298, Dual Full-Bridge Driver, ST Microelectronics, http://www.chipcards.de/

PIC16C77, Microcontroller, Arizona Microchip, http://www.microchip.com

ICL7642, Quad Op-amp, Harris Semiconductor, http://www.harris.com/harris/

GP2D12, IR Proximity Sensor, Sharp, http://www.sharp.com

Abstract

The Annual Trinity College Fire-Fighting Home Robot Contest has gained much popularity recently. Many schools participate in the event, which makes for a very challenging and competitive atmosphere. Although the national competition is not a main focus for the EE 382 Junior Design course at NMT, it is viewed as an incentive for the students to design and build a fire-fighting robot to the best of their ability.

Over the last few years, the fire-fighting robots at NMT have each had their own unique designs and personalities, but they have all had the same brain (microcontroller), the Motorola 68HC11. Our group decided to turn over a new leaf and try something different. We chose use a new family of microntrollers, the MicroCHIP PIC16C77 series. We chose this particular microcontroller because they are very versatile, space saving, and relatively inexpensive. We were drawn to the features of the PIC, mainly the analog-to-digital converter and embedded pulse-width-modulation (PWM) capabilities. Also, we were impressed with the size of the microcontroller itself. One 40-pin package would be sufficient enough control all of the features of our robot.

Since we were the first group to use the PIC series of microcontrollers, we were taking a risk. The following report contains all of the data that we have assessed throughout the spring semester of 1999. Hopefully, we can pave the way for future groups who may choose to use the PIC for the fire-fighting robot design.

Chassis Design

Motor and Drive Hardware

For the design of our robot we decided to use the differential drive configuration with a single caster. This design is called a three point because of the number of point touching the ground at any given time. The benefits of using the differential drive are the ease of implementation, the in-place turning ability, and this is the way the base was delivered to our group. The motors that we used on the robot were manufactured by Pittman and are 24 vdc gearhead motors with optical encoders. The final drive ratio for the motors is 5.9:1, which produces enough torque for only 7 vdc needed to drive the motors. The motors were mounted to the bottom of the lower plate so that the wheels were parallel. The need for the wheels to travel in a parallel direction makes the robot initially travel in a straight line. If the wheels were not parallel then the robot would tend to travel in an arc and make the motor control compensate for the design error. Initially our group was going to use two caster assemblies, but after assembling the robot the simplicity of the one castor assembly was selected. In place of the front castor is where the white line detection sensor is placed. If the robot was to tip forward the plastic guard for the white line sensor would allow the robot to rebound back to the castor, because the guard is beyond the balance point of the fulcrum.

Platform

The amount of physical space that our design needed was minimal, so our robot only needed two layers to mount all of the hardware. On the bottom of the first layer is where the two motors and the single caster assembly are mounted (Fig. 1). Also on the bottom of the first layer is where the white line sensor is mounted. The top of the first

layer is where the proximity sensors are located. In order to establish a clear viewing angle for the proximity sensors, nothing can be placed in front of the sensors. Behind the proximity sensors is where the battery is located (Fig. 2). For our robot only one battery is needed, but the battery that we are using is too large to go under the first layer. On the bottom of the second layer is where the motor control and voltage regulation boards are located (Fig. 3). To reduce the effects of electrical noise on the microcontroller and sensor board, the two main sources (other than the motors) where separated from the control area. Also under the second layer is where the fan that is used to extinguish the fire is located. The top of the second layer is where all of the components that need to be adjusted are located (Fig. 4). The PIC16C77 is located on top to facilitate reprogramming and insertion upon programming changes. The sensor board is also located on the top of the second layer so that the gains for each sensor can be adjusted if needed. Each layer of the robot is 1/8" aluminum plate with 6" long 8-32 all-tread spacer. Four spacer are used to separate the two layers and add rigidity to the design.

Microcontroller and Software

Microcontroller

The microcontroller is responsible for all of the decisions that our robot must make to navigate through the maze. The microcontroller must control the motors from the input it receives from the proximity sensors. If the robot gets close to one wall the microcontroller should slow down the motor on the other side to correct it self. Another responsibility of the microcontroller is to monitor the flame detection sensor to look for the candle. If the candle is found the microcontroller must control the motors to approach the fire, also monitoring the proximity sensors so that the robot maintains a safe distance

from the wall. Once the robot has stopped within 12 inches of the flame, the microcontroller must turn on the flame suppression fan. Finally, after the flame is extinguished the microcontroller must return the robot to home.

Given all of the required information that the microcontroller must monitor, we selected a microcontroller with several different beneficial systems. The first feature that we wanted in our microcontroller was at least 2 channels of pulse width modulation (PWM). PWM is used to control the motors by changing the duty cycle. The Motorola 68HC12 and the MicroCHIP PIC16C77 both have 2 channels of embedded PWM control. Other features that we required for our microcontroller are 8 lines of A/D for poling the sensors. The PIC16C77 has 8 lines of A/D and two pulse accumulators. The reason that the pulse accumulators is an important feature is it can be used to monitor the optical encoder form the motor and eliminate 2 line needed for A/D conversion. The PIC16C77 meets all of the requirements that our group identified and its size is the biggest benefit. Compared to the 68HC1X the PIC16C77 is only a 40-pin DIP chip compared to the evaluation board of the Motorola's.

Software

Programming the PIC16C77 was quite challenging causing our progress to be slower than expected. First, we thought we were going to have access to a C-compiler that would have made the programming task much easier. As it turns out, the C-compiler that was available to us was for different series of PIC microcontrollers. Consequently, we programmed the PIC16C77 using Microchip's assembler. Second, we thought the PIC16C77 was a flash programmable part that would of allowed use to change program much more quickly. However, the PIC16C77 microcontroller is an EPROM type device

that requires 30 to 40 minutes of erase time before reprogramming. And third, debugging and interacting with the PIC16C77 microcontroller requires special features to be added to the software that is being written. In other words, if constants need to be changed, then the software has to be written to allow the operator to enter these constants. There are many creative was of doing this, some of which we employed; but, they add to the complexity of the program and detract from the time that can be devoted to solving the problems at hand. On the other hand, the instruction set for the PIC16C77, which is quite easy to learn, combined with it RISC base design make it a very viable selection in terms of programming.

Enough said about the challenges we faced in programming the PIC16C77, let us elaborate on the programming that was accomplished. Three main sets of code were developed. The first program developed was simply to generate a varying pulse width modulation (PWM) signal for each motor. The second set of code focused on closed loop motor control. It was broken into two programs: one, that used a proportional controller for speed control and the second that used proportional plus integral control. This set of code allowed the operator to change the speed setting and the proportionality constant. The third set of code added left wall following using the left wall sensor.

PWM

The PWM code was the first set of code written for the robot. You can find the source listing in Appendix B. Its main purpose was to demonstrate that the motors could be driven with a PWM signal and to find the correct frequency at which to drive the motors. Figure 11 shows a simple flowchart of this program. The first thing the code does is initialize constants, setup ports, initialize analog to digital (AD) converter, and

setup the PWM function. Then the software enters the main loop. Input voltages are read using the AD to gain the amount of time that the PWM signal is to remain high. The PWM uses an 8-bit integer to determine its on time by comparing a counter with a value stored in a register. This value is update using the voltage read from the AD which is also and 8-bit integer. Thus the PWM duty cycle is directly proportional the input voltage on each channel. There are two separate PWM module in the PIC16C77 so this process is repeated for each channel. Once the PWM duty cycle has been updated, the software wait until the present PWM cycle is complete. Then it repeats the process described above starting with reinitializing the ports and special function modules. It is a good idea to reinitializing the functions at some interval within the software to increase the reliability of the system.

Closed Loop Control

The PWM code was modified to add closed loop control. Two versions of this code were written and can be found in Appendix B. The first accomplishes closed loop control using a proportional control technique; and, the second utilized proportional and integral control. Because the software does not check for overflow errors, both routines perform erratically at the end points of desired speed. Figure 12 shows a basic flow chart of both programs. It is very similar to the PWM code discussed above. It uses the two remaining counters to get the actual speed from the encoders of the motors. In addition, the respective controllers implemented. The ability to adjust the proportionality constant has been added by reading of port B.

Left Wall Following

Here again the left wall following program grew out of the proceeding to programs. It was decided, after some minimal experimentation due to time constraints, to only implement the proportional type control in the wall following program. The wall following code is listed in Appendix B and the flowchart is shown in Figure 13. This code progresses similarly to the two discussed above with the addition of reading the AD to obtain distance information. The value read from the left wall sensor is used to change the speed of each motor. We did not have time to try this code out and it still needs to be debugged using the Microchip simulator before programming a part.

Subsystem Design

Power Supply

We decided to use a 12V battery with two voltage regulators to activate the motor and the rest of the components in the robot. The voltage regulators we used were LM 317 and LM 7805 as shown in the block diagram of figure 9.

LM 317

The LM 317 is a 3-termianl regulator that can regulate a voltage between .25V to 25V by using an adjustable pot to produce the desired output. Since we needed the motor speed to be adjustable between 3V to 11V, we chose the LM317 to regulate the 12V Battery and produce a regulated voltage between 3V to 11V using a 10K pot.

LM 7805

The LM 7805 is a 3-terminal voltage regulator that can regulate a voltage between 7V to 35V and produce a fixed 5V. Most of the components we had in our robot were

running under 5V except the motor; therefore, we chose the 7805 regulator to regulate the 12V battery and outputs a 5V to these components as shown in figure 10.

Sensors

Sensors give the robot valuable information about the environment that it is navigating through. The robot needs this information to decide which direction to go to put out the candle flame. For our project we used three types of sensors: wall sensors, white line sensors, and a flame-detecting sensor.

Wall sensors

The purpose of using wall sensors is to make the robot move in a straight line without colliding with any wall. The robot measures the distance between its chassis and the wall and adjusts its position accordingly.

However, since there were several types of wall sensors that we can use, we needed to choose one that had an efficient range for distance measuring and good accuracy. At the beginning of the semester we decided to design a phototransistor with a 555 timer as an emitter and use the GP1U52X as a receiver. The phototransistor was assembled with 2 IR LED and a 555 timer that works with a frequency of 40 kHz and under 50% duty cycle as shown in figure 5. The circuit is supposed, to send an infrared signal to detect any walls around the robot and, is received by the GP1U52X (photodiode receiver). Then, the circuit will transform the signal to the microprocessor to be analyzed.

Later on the semester we found out that the GP2D12 is more efficient in size and has better accuracy in measurements than the one we were designing.

The GP2D12 is a distance-measuring sensor, which has an IR emitter and a sensitive detector in a single package. It has a very accurate method of measuring the

distance to an object by using the triangulation method. Additionally, it is insensitive to the color and texture of the object it is pointing at and has a range of 3.9in-43.3in, which is sufficient need for the maze. Therefore, we decided to use the GP2D12 as a wall sensor instead of the previous design we had.

In order to give the robot complete coverage of it's surroundings we used three GP2D12 sensors with a 45° angle between each other as shown in figure 6.

Moreover, we took some measurements out of the GP2D12 and plotted them against distance to see how much distance the sensor can detect as shown in table 1, figure 7, and figure 8.

As we can see from the plots, the GP2D12 has high sensitivity, which can detect any object up to the range of 1.1M or 43.3In with good linearity.

White Line Sensor

The schematic of the white line sensor is shown in Appendix A. It utilizes a light feedback technique to eliminate the need for a manually variable IR source. The circuit automatically adjusts IR emission so that the photodetector always receives the same amount of IR light. When the sensor passes over a white line, more light is reflected back to the photodetector causing the circuit to compensate by reducing the drive current to the IR LED. IR emission is directly proportional to the current flowing through the IR LED. Consequently, a measure of how much light is need to keep the light constant at the photodetector is available at the base and emitter of Q5. Since the emitter has the lower output impedance, it is the desired output of the light feedback circuit. The voltages at the emitter ranges from about 1.8 volts when a white line is present to 2.3 volts when it is not hence the signal needs to be amplified. Op-Amp U5D performs two functions. It

amplifies this small signal and removes the DC offset that the signal is riding on. This same amplifier is used by each of the GP2D12 distance sensors.

Flame Sensor

The Flame Sensor Schematic is shown in Appendix A. The flame sensor utilizes a Large Area Photo Detector (LAPD) to detect the flame from a candle. The LAPD is approximately 0.5" square and is sensitive to light. To keep the sensor from detecting ambient light, an optical filter made out of cut up floppy disk media was used. The LAPD is a current device so a current to voltage converter was used to condition its signal. A variable gain amplifier was used to amplify the output of the current to voltage converter thus providing greater functionality. Three gains are used to achieve three different ranges. The ranges are approximately 6 to 3.5 feet, 3.5 to 1.75 feet and 1.75 to 0.5 feet.

Motor Control

In order to establish proper motor control for the robot, the use of an h-bridge circuit is required. The h-bridge circuit consists of a set of four transistors in an IC package that are arranged in an "H" orientation. This layout allows for current to flow bi-directionally through the circuit thus allowing for directional control for our motors. Additionally, logic input signals can be used to determine which direction the motors are spinning. Depending on the paired combination of logic 1's and 0's the motor shaft can turn left, turn right, and brake. Speed control is another feature of the h-bridges, when given a pulse-width-modulated (PWM) input signal, depending on the length of the duty cycle; the speed can be varied accordingly.

H-bridges

For our robot we chose to use the L298N series dual full-bridge driver manufactured by STMicroelectronics . We chose this particular h-bridge simply because they were relatively inexpensive (they cost us about $3.00 a piece from Allied), we could control both motors with just one chip thus saving space on the chassis, and they were very robust and not prone to damage from static discharge.

The design for our h-bridge board came directly from the L298 data sheet. It turns out that there is a schematic for bi-directional DC motor control for only one motor, but since the circuit just needed to be mirrored to allow for both motors, the design was easily implemented. The h-bridge circuit did require some minor modifications; the data sheet called for an external bridge of four fast-recovery Shottky diodes to be placed on each of the outputs. The purpose of the diodes is to prevent large spikes of current from entering back into the h-bridge when the motors abruptly stop. The layout for the printed-circuit board was done using the MicroSim evaluation software package provided by the department. Our original plan was to place both the h-bridges and the frequency-to-voltage converters on the same board, but since we decided to utilize the pulse accumulator of the PIC instead, we did not connect the frequency to voltage converters to the rest of the circuit.

PC Board Design and Layout

Some objectives for the project were to be efficient and to complete the task with as few components as possible. Originally, we wanted to put all of the electrical components on a single printed-circuit board. Since the PIC16C77 does not require much space, we felt that this was definitely a possibility. However, we ran into some software

issues when the actual design for the board was being implemented. Since we decided to use MicroSim for the design layout, we were subject to the limitations of the evaluation software. Our biggest problem was simply the fact that we were not able to place all of the components on one board. The evaluation version of MicroSim only allows a certain number of nets to be placed simultaneously in one file, including all of the components on one layout required too many nets and therefore we were forced to build separate boards. Also, use of the autorouter function could not be taken advantage of in the evaluation version either. Unfortunately, bus lines had to be traced and connected by hand, which turned out to be a little tedious at times. Regardless of the software complications, all the boards were successfully designed and etched.

Budget

The budget constraint set for the project was $100. Fortunately, one of our group members was employed by Sandia National Laboratories, which loaned us parts for our robot. We also received free samples of the PIC16C77 directly from Microchip . The majority of our budget was spent on an extra 7.2V battery, the proximity sensors, and various connectors, sockets, and hardware for the chassis. The grand total spent by our group was about $90.82, which could have been much more considering a large portion of parts were donated. Figure 14 is a pie chart representing our allocated budget.

Conclusion

This project presented a problem that required a well thought out solution. Our group decided that a different solution might be better than the common 68HC11/Altera solution. At the beginning of the semester we were going to use the PIC16C77 for motor control and use the 68HC11 for the primary control unit. Upon researching the

PIC16CXX family of microcontrollers, we decided that we would use one PIC microcontroller instead of the 68HC11 with a PIC motor control. The PIC16C77 offered embedded PWM motor control and 8K of onboard memory. The 8 lines of A/D on the PIC were enough for our design specifications. The amount of embedded controls and I/O lines from the PIC16C77 simplified our wiring and overall design. The major problem with the PIC16CXX family of microcontrollers is the C compiler is produced by another company and test results of that compiler is that it does not work. The lack of a working compiler forced us to use Assembly programming language. Assembly is very efficient code but is also difficult for beginners to produce working code. One member of our team is very proficient, so he did all of the programming. In the future, if any group was to use a PIC microcontroller, we suggest that they use a PIC17CXXX family processor because MicroCHIP has a working C compiler for these microcontrollers. The overall status of our robot is that it is not working, but most of the subsystems are working. The PIC has close loop control the motors, can read each of the proximity sensors, and can detect the flame and turn on the fan. All of these subsystems need to be incorporated in the code. Once all of the subsystems have been incorporated in the code then we can calibrate each sensor for exact distances. Our group would like the opportunity to finish our robot because we believe that our design is better and more efficient.

Figures

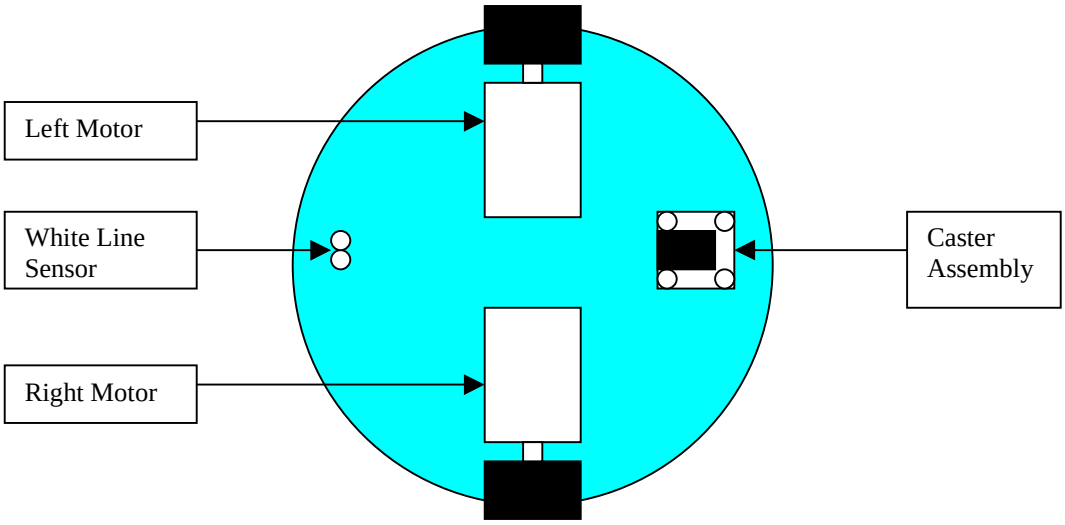


Figure 1: Bottom of First Layer

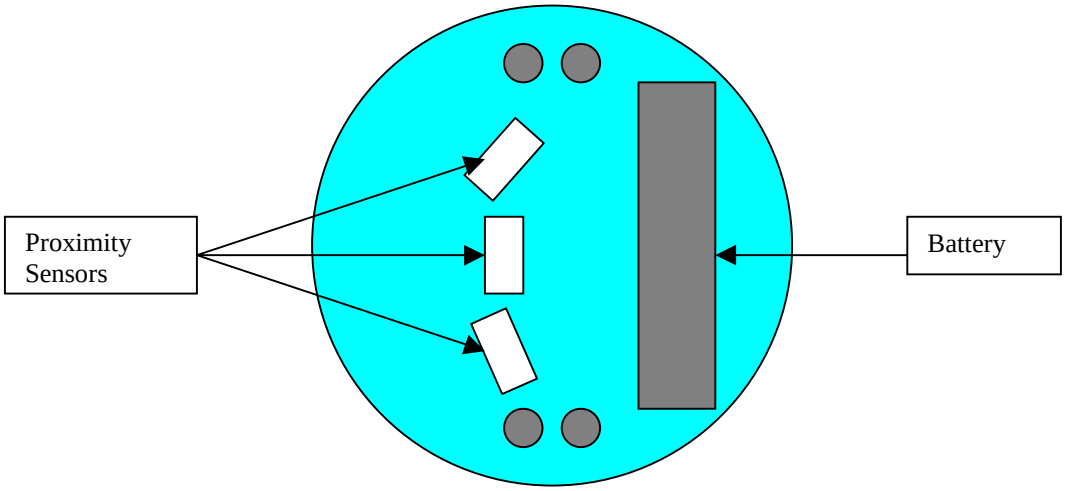


Figure 2: Top of First Layer

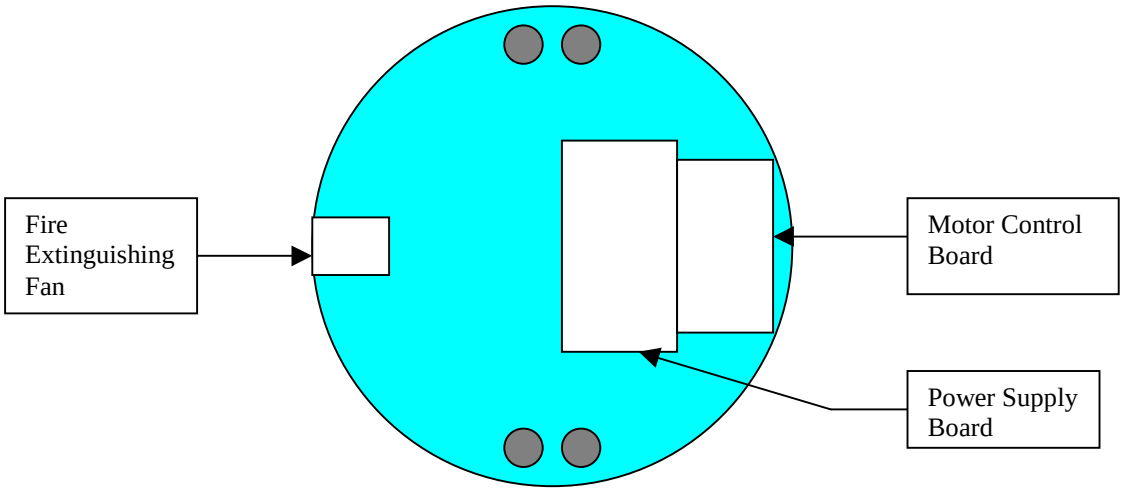


Figure 3: Bottom of Second Layer

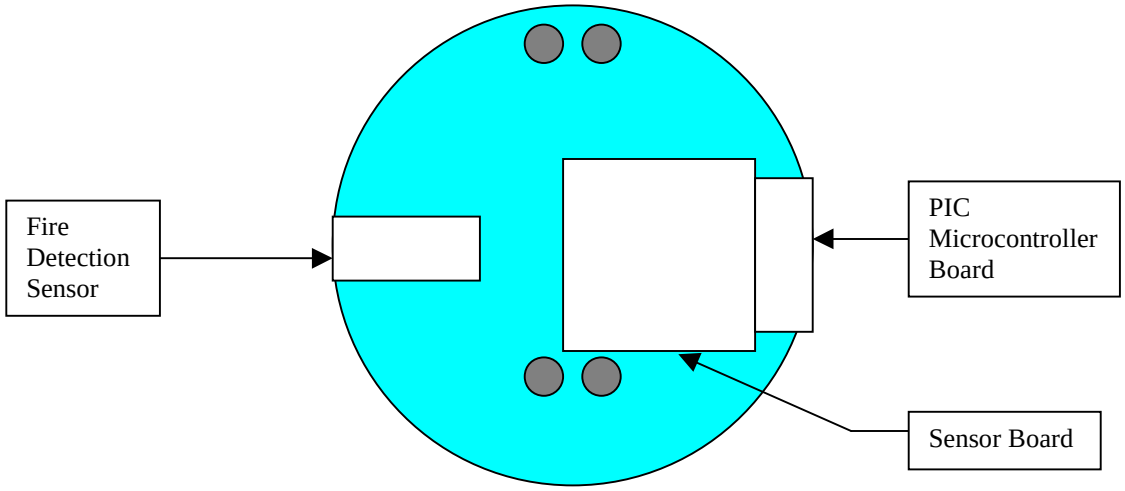


Figure 4: Top of Second Layer

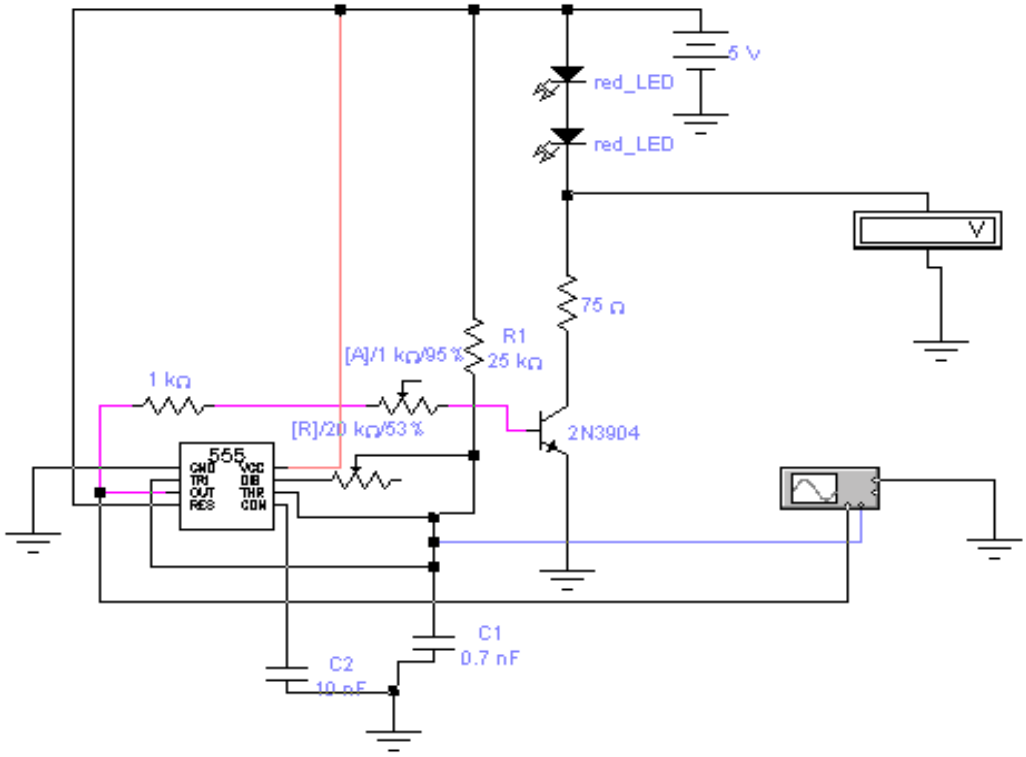


Figure. 5: The circuit design of the phototransistor.

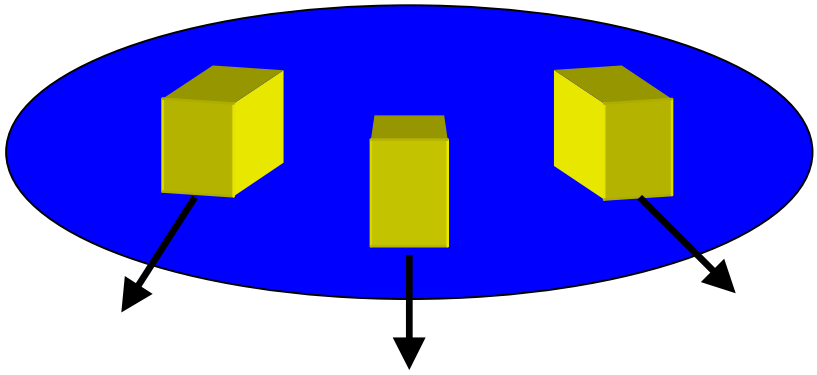


Figure. 6: The layout of the GP2D12 on the robot

Output Voltage(v)	Range(m)	Output Voltage(v) with gain of 2v/v
1.342	0.2	2.684
1.088	0.25	2.176
0.928	0.3	1.856
0.78	0.35	1.56
0.704	0.4	1.408
0.628	0.45	1.256
0.546	0.5	1.092
0.488	0.55	0.976
0.451	0.6	0.902
0.431	0.65	0.862
0.393	0.7	0.786
0.354	0.75	0.708
0.335	0.8	0.67
0.316	0.85	0.632
0.31	0.9	0.62
0.293	0.95	0.586
0.288	1	0.576
0.274	1.05	0.548
0.273	1.1	0.546

Table 1: Measurements for the GP2D12

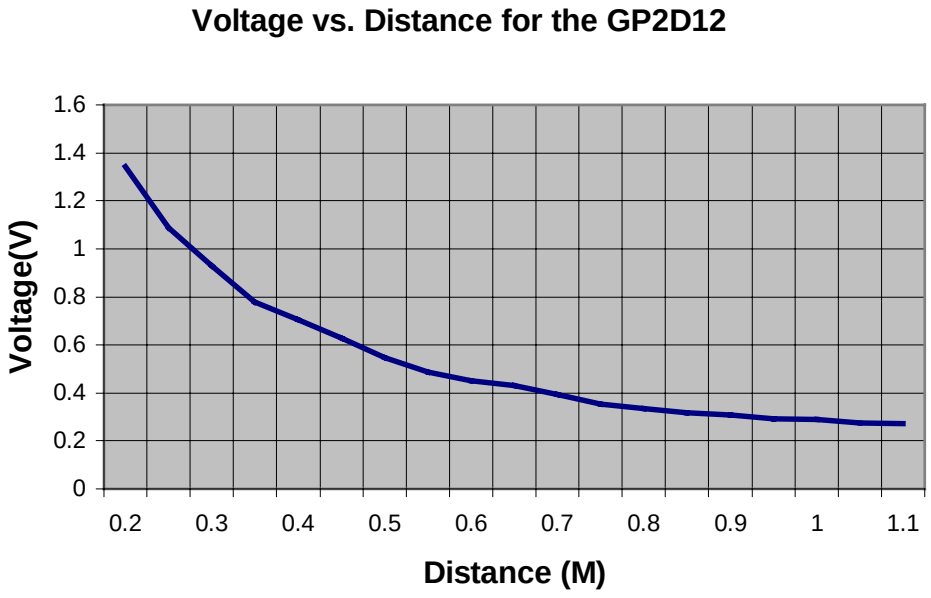


Figure 7: Plot of Voltage vs. Distance for the GP2D12

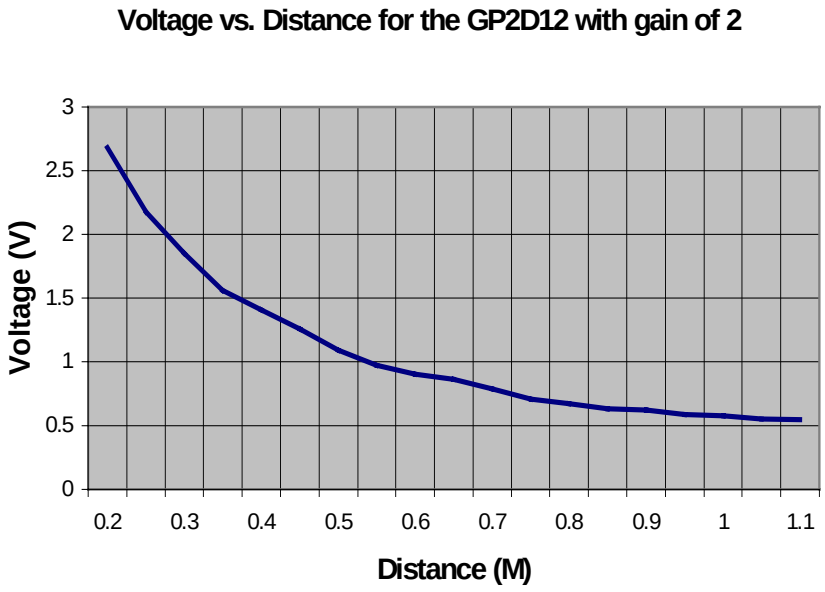


Figure 8: Plot of Voltage vs. Distance with gain of two for the GP2D12.

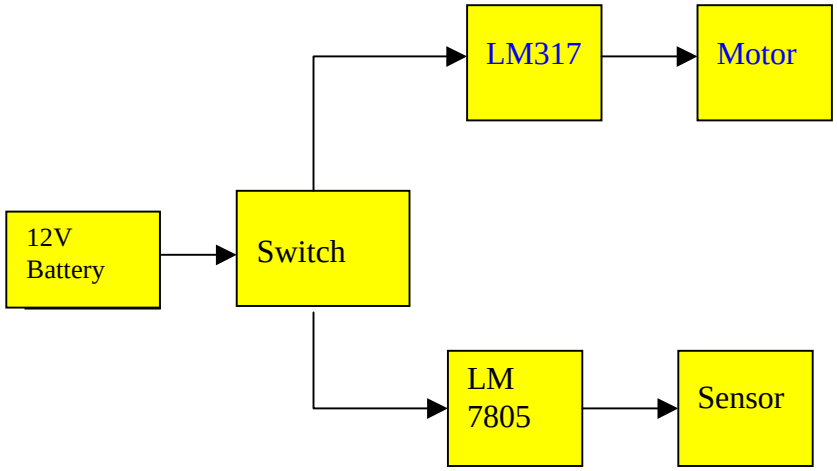


Figure 9: Block Diagram of the power supply.

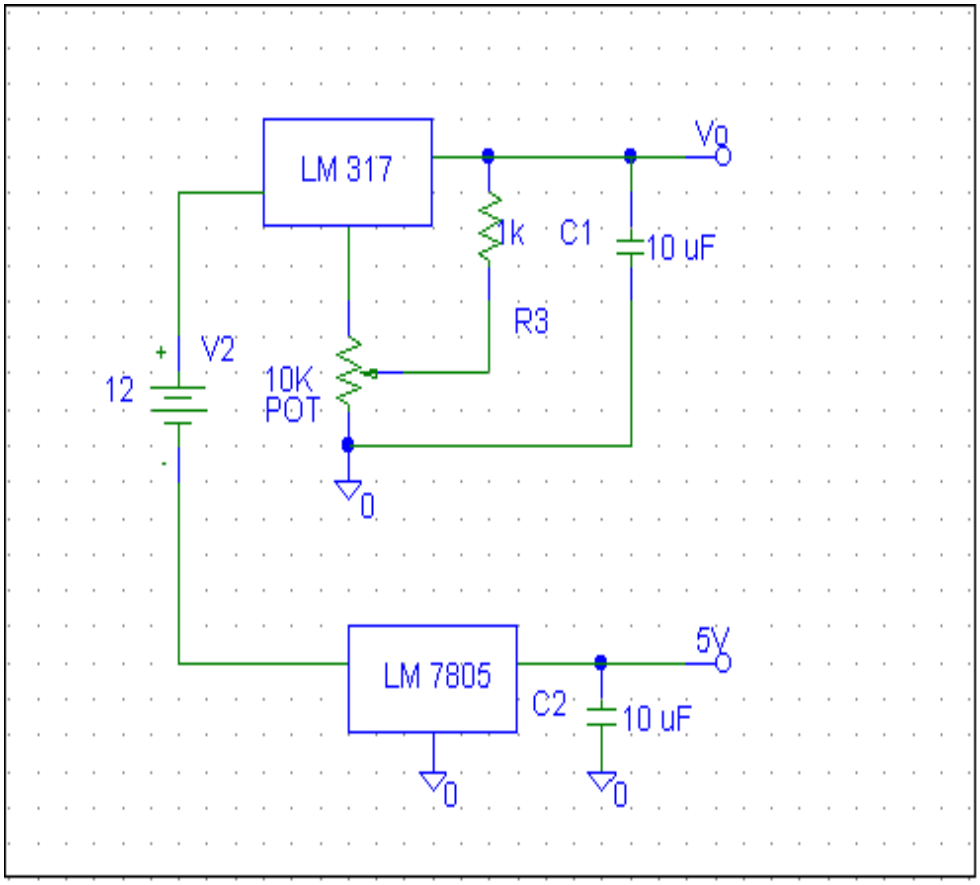


Figure 10: A complete Schematic of the power supply.

Figure 11 PWM Flowchart

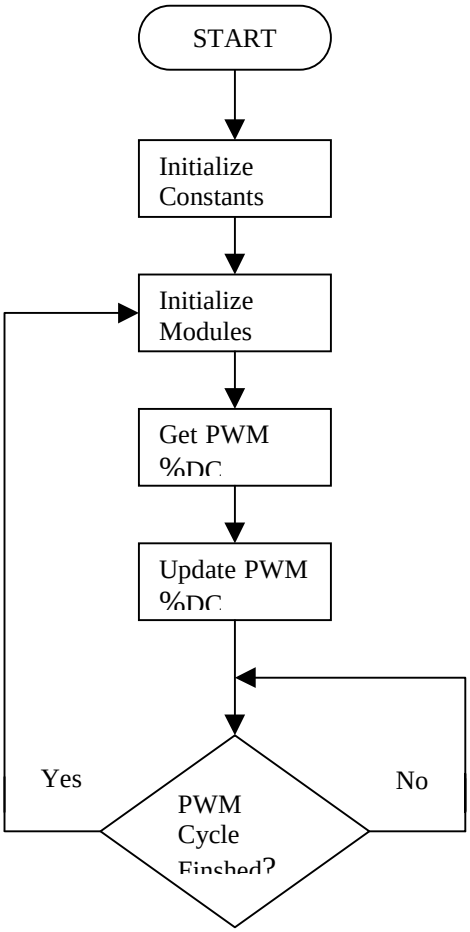


Figure 12 PWM Closed Loop Flowchart

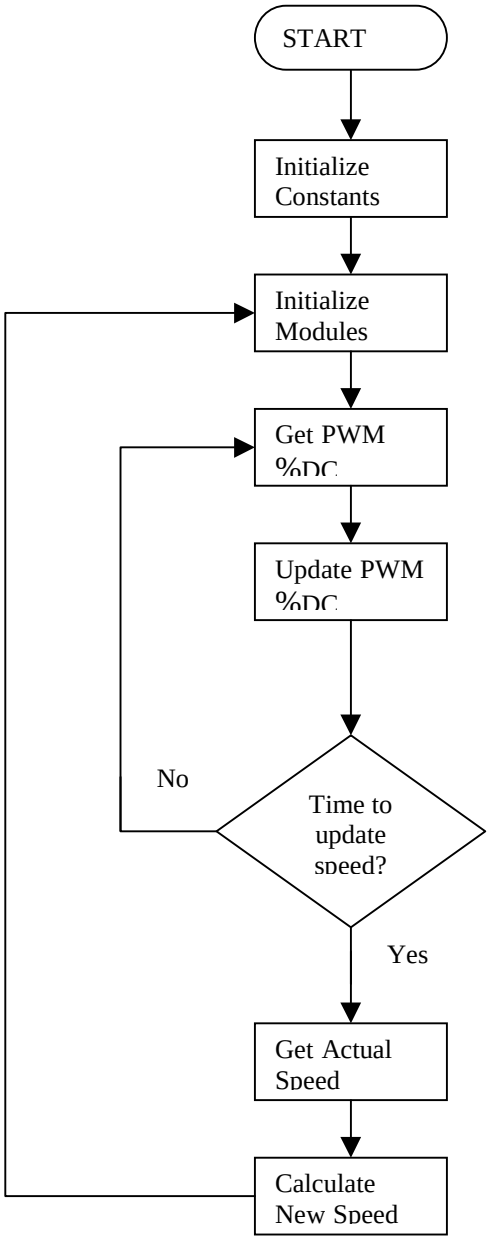
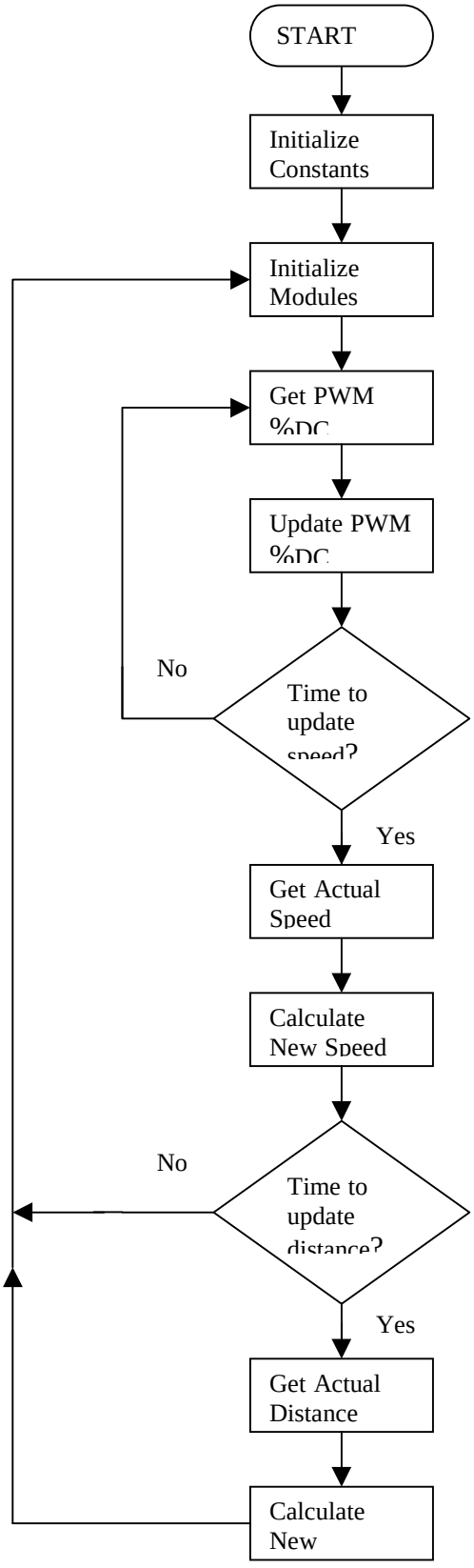


Figure 13 Left Wall Following Flowchart



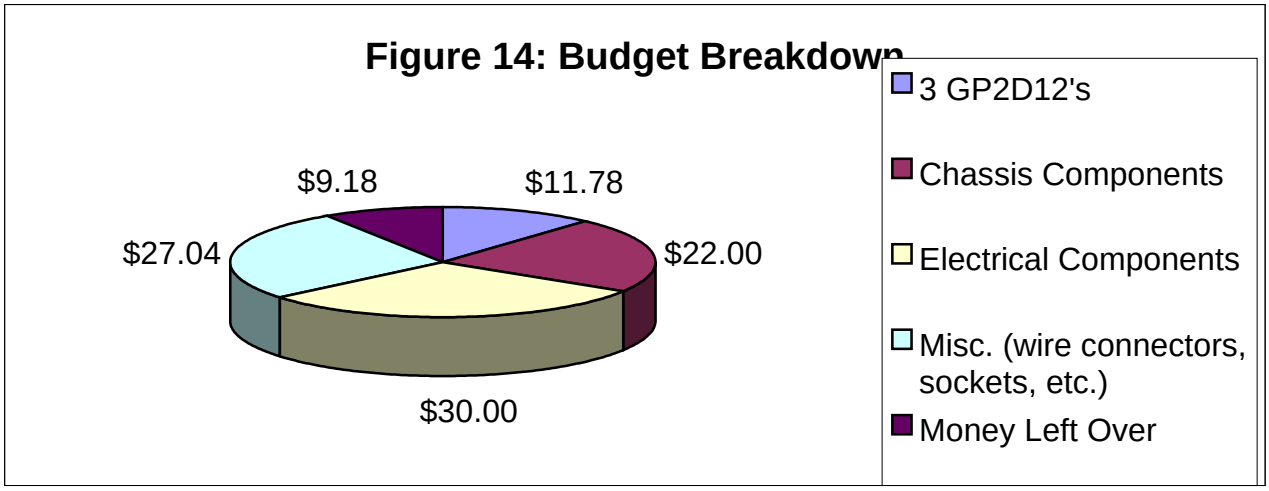
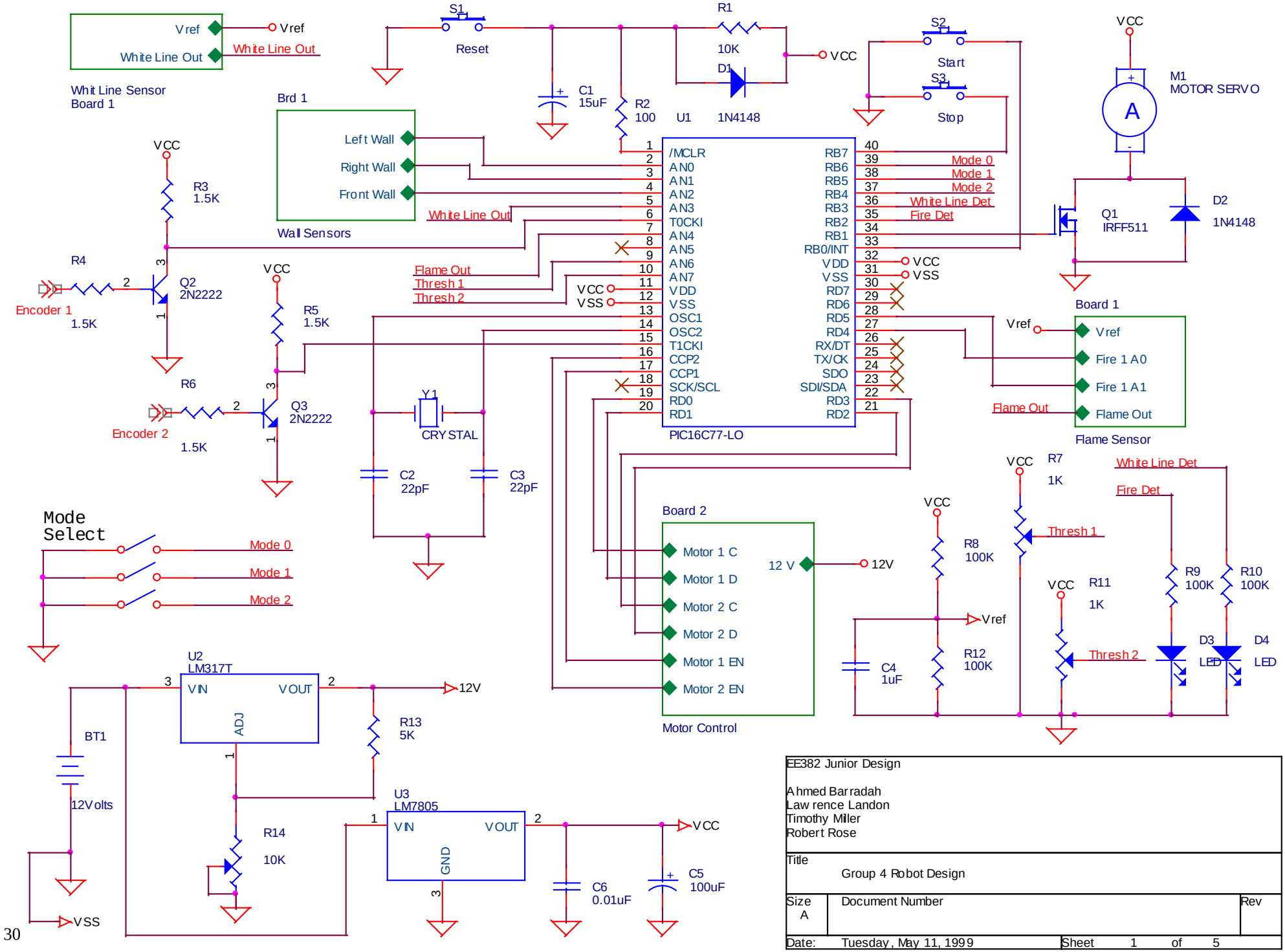


Figure 14: Budget Breakdown
$9.18
$11.78
$27.04
$22.00
$30.00
3 GP2D12's
Chassis Components
Electrical Components
Misc. (wire connectors, sockets, etc.)
Money Left Over

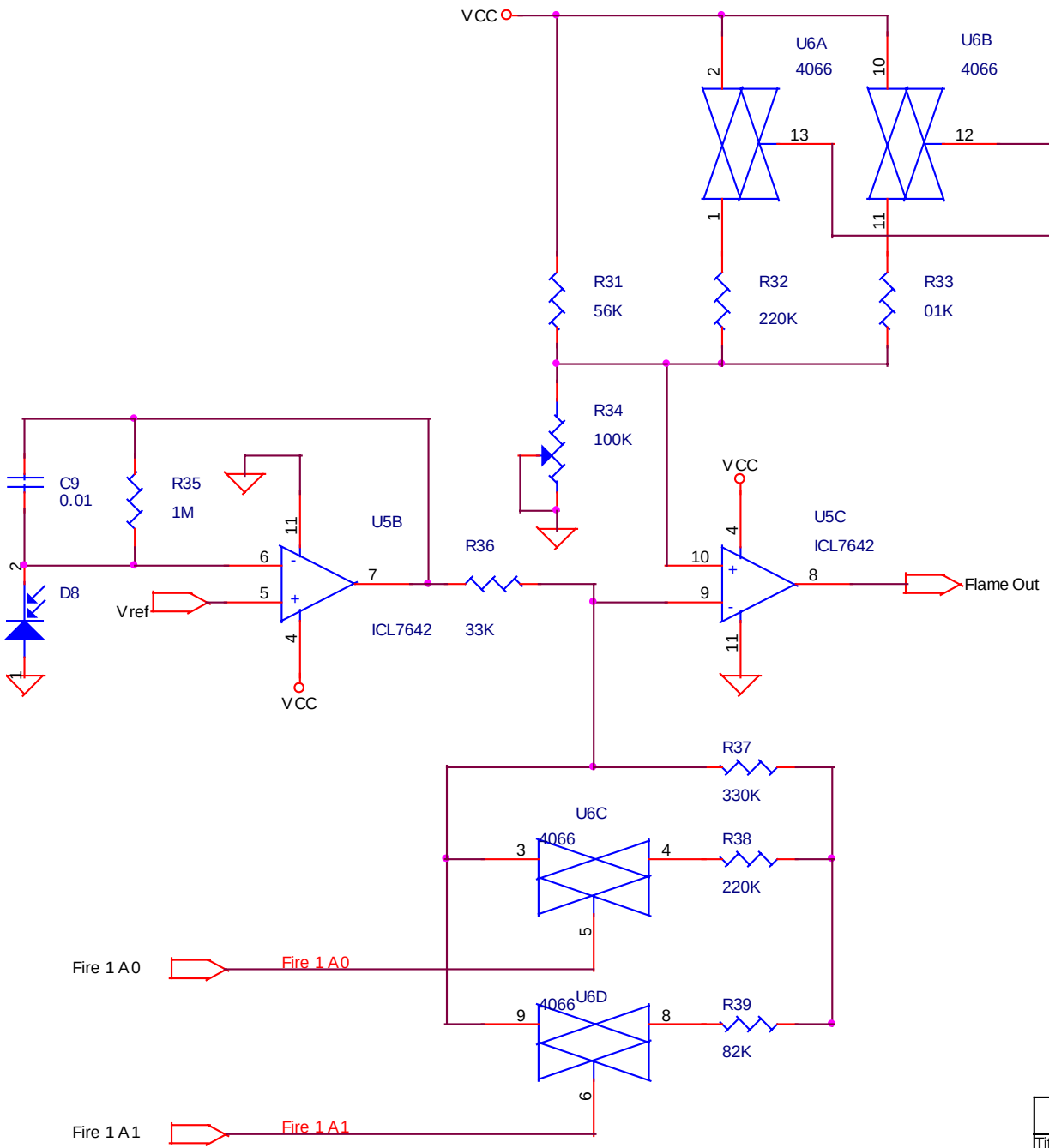
Appendix

A

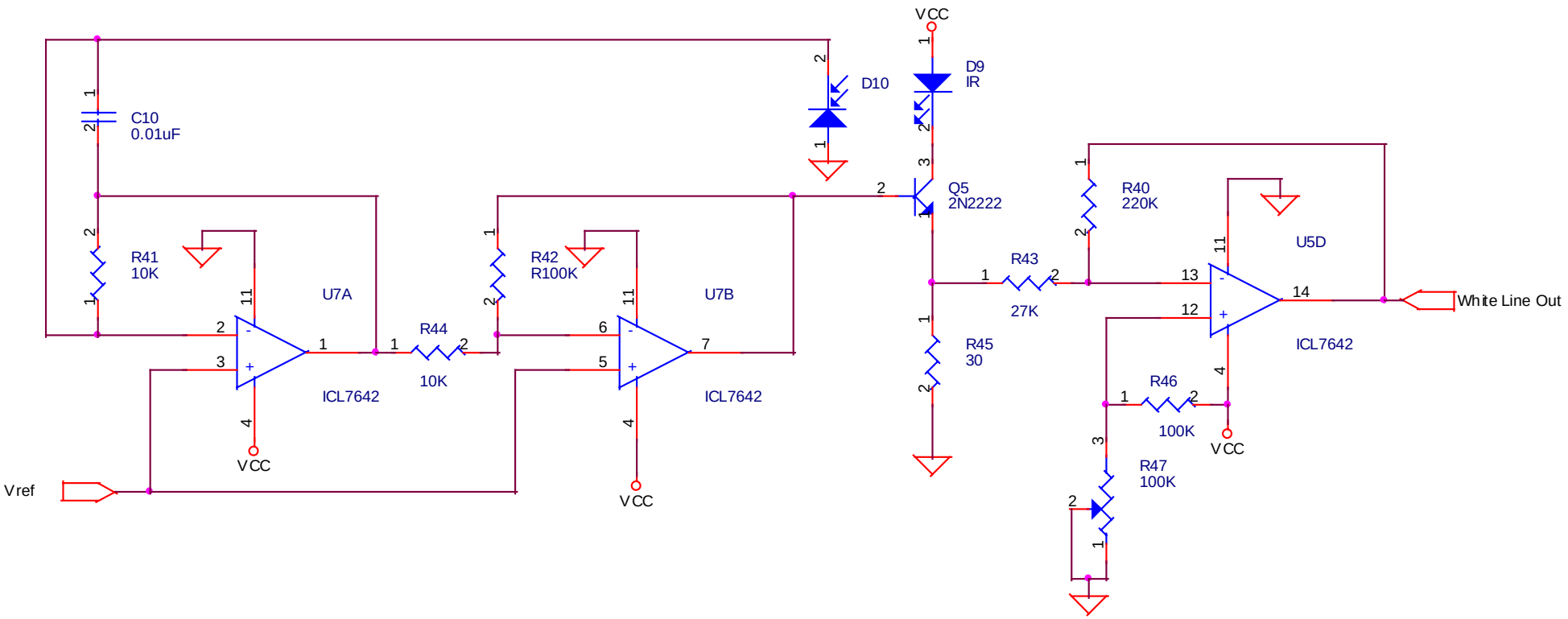
Schematics



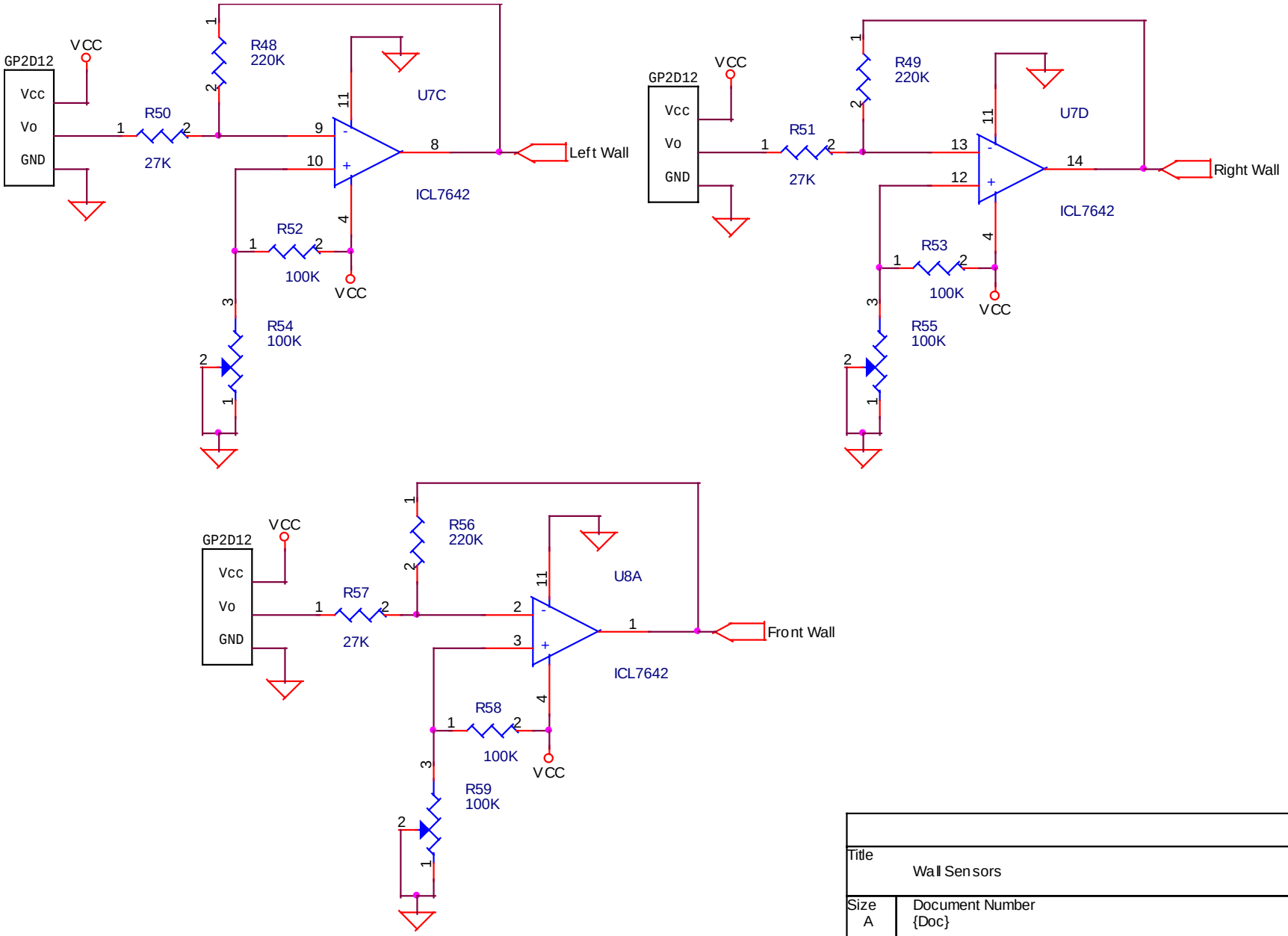
EE382 Junior Design Ahmed Barradah Law rence Landon Timothy Miller Robert Rose		
Title	Group 4 Robot Design	
Size A	Document Number	Rev
Date: Tuesday, May 11, 1999	Sheet 1 of 5	



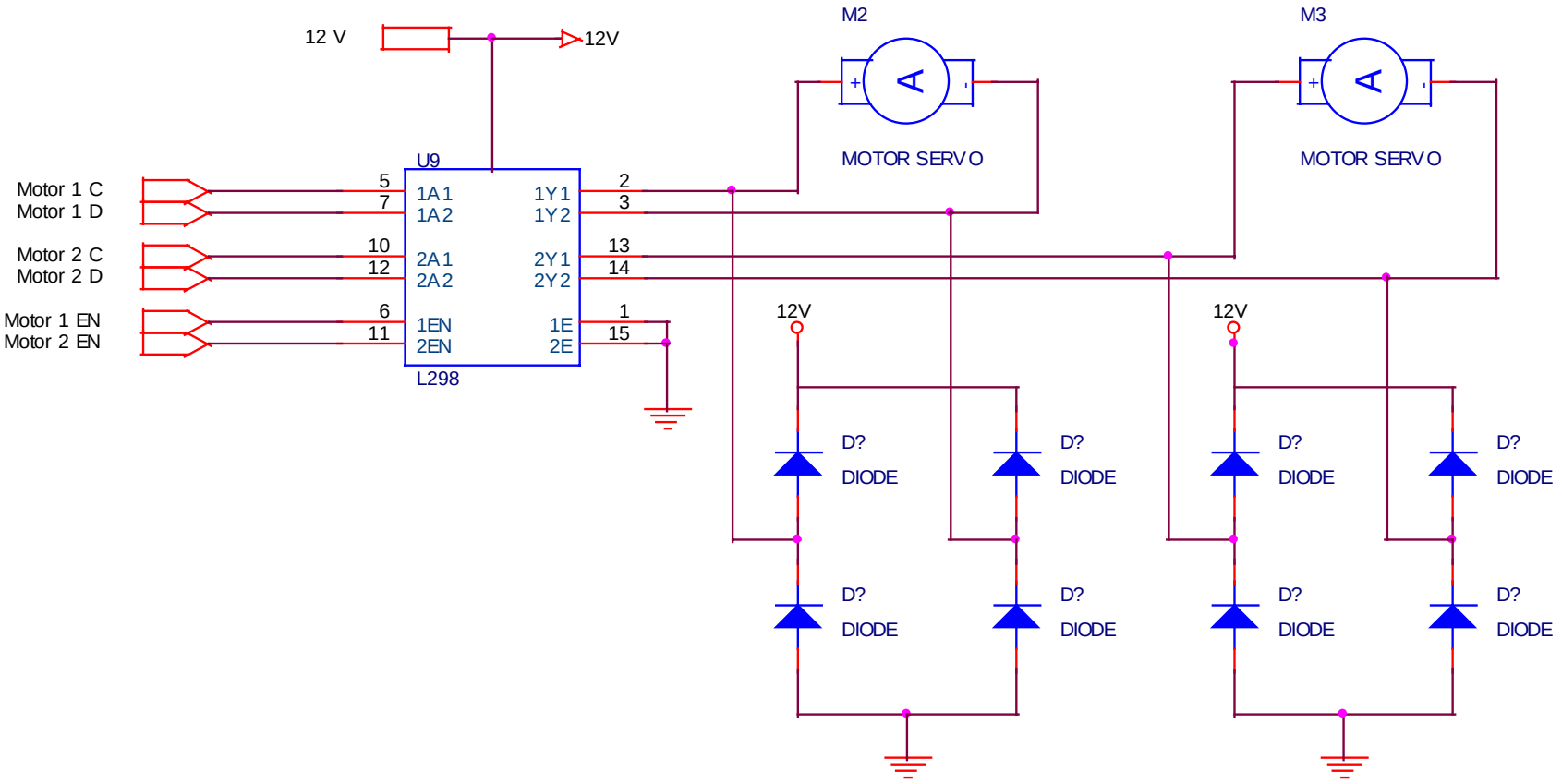
Title: Flame Sensor		
Size: A	Document Number: {Doc}	Rev: {RevCode}
Date: Tuesday, May 11, 1999	Sheet 2 of 5	



Title: White Line Sensor			
Size: A	Document Number: {Doc}		Rev: {RevCode}
Date: Tuesday, May 11, 1999		Sheet 3 of 5	



Title: Wall Sensors		
Size: A	Document Number: {Doc}	Rev: {RevCode}
Date: Tuesday, May 11, 1999	Sheet 4 of 5	



Title: Motor Control		
Size: A	Document Number: {Doc}	Rev: {RevCode}
Date: Tuesday, May 11, 1999	Sheet 5 of 5	

This page intentionally left blank

Appendix

B

Source Code

PWM
Source Code

```
;*****************************************************************
;   This file is a basic code template for assembly code generation   *
;   on the PICmicro PIC16C77. This file contains the basic code       *
;   building blocks to build upon.                                    *
;                                                                     *
;   If interrupts are not used all code presented between the ORG     *
;   0x004 directive and the label main can be removed. In addition    *
;   the variable assignments for 'w_temp' and 'status_temp' can       *
;   be removed.                                                       *
;                                                                     *
;   Refer to the MPASM User's Guide for additional information on     *
;   features of the assembler (Document DS33014).                     *
;                                                                     *
;   Refer to the respective PICmicro data sheet for additional        *
;   information on the instruction set.                               *
;                                                                     *
;   Template file assembled with MPLAB V3.99.18 and MPASM V2.15.06.   *
;                                                                     *
;*****************************************************************
;                                                                     *
;    Filename:       motor.asm                                        *
;    Date:         3-24-99                                          *
;    File Version: V1.0                                             *
;                                                                     *
;    Author:       Tim Miller                                       *
;    Company:      NMT                                              *
;                                                                     *
;                                                                     *
;*****************************************************************
;                                                                     *
;    Files required: p16c77.inc                                       *
;                                                                     *
;                                                                     *
;                                                                     *
;*****************************************************************
;                                                                     *
;    Notes:  This is the first attempt at writing code for the 16C77  *
;    03-27-99     Adder PWM                                             *
;                                                                     *
;                                                                     *
;                                                                     *
;*****************************************************************


      list   p=16c77          ; list directive to define processor
      #include <p16c77.inc>   ; processor specific variable definitions

      __CONFIG   _CP_OFF & _WDT_ON & _BODEN_ON & _PWRTE_ON & _RC_OSC

; '__CONFIG' directive is used to embed configuration data within .asm file.
; The lables following the directive are located in the respective .inc file.
; See respective data sheet for additional information on configuration word.



;***** A to D DEFINITIONS
; These defintions are used to selsect the A to D channel
```

```
; They are set to use the internal RC clock for conversion time
; They are also set to have the A to D turned on when the channel is
; is selected.  So, use the following lines to set the channel and
; turn the A to D on using the internal RC conversion clock.
;     movlw CHx               ; Where CHx is CH1, CH2 ...
;     movwf ADCON0            ;

CH0   EQU   B'11000001' ; Channel 0
CH1   EQU   B'11001001' ; Channel 1
CH2   EQU   B'11010001' ; Channel 2
CH3   EQU   B'11011001' ; Channel 3
CH4   EQU   B'11100001' ; Channel 4
CH5   EQU   B'11101001' ; Channel 5
CH6   EQU   B'11110001' ; Channel 6
CH7   EQU   B'11111001' ; Channel 7


;***** VARIABLE DEFINITIONS
w_temp      EQU     0x70      ; variable used for context saving
status_temp EQU     0x71      ; variable used for context saving


TEMP        EQU     20h       ; temp variable







;**********************************************************************
      ORG   0x000       ; processor reset vector
      clrf  PCLATH      ; ensure page bits are claared
      goto  main        ; go to beginning of program

;*************** Interrupt Service Routine **************************

      ORG   0x004       ; interrupt vector location
      movwf w_temp      ; save off current W register contents
      movf  STATUS,w    ; move status register into W register
      movwf status_temp ; save off contents of STATUS register


; isr code can go here or be located as a call subroutine elsewhere


      movf  status_temp,w     ; retrieve copy of STATUS register
      movwf STATUS            ; restore pre-isr STATUS register contents
      swapf w_temp,f
      swapf w_temp,w          ; restore pre-isr W register contents
      retfie                  ; return from interrupt

;*************** Main Program *************************************

main
      call  InitializePORTS
      call  InitializeAD
      call  InitializePWM
```

```
update
      bcf   PIR1,TMR2IF       ;clear period flag
      movlw CH0               ;setup for Ch 0
      movwf ADCON0            ;sets up  A to D
      call  GetAD             ;starts A to D
      movf  ADRES,W           ;get a/d value
      movwf CCPR1L            ;set dutycycle
      movlw CH1               ;setup for Ch 0
      movwf ADCON0            ;sets up  A to D
      call  GetAD             ;starts A to D
      movf  ADRES,W           ;get a/d value
      movwf CCPR2L            ;set dutycycle
wait
      btfss PIR1,TMR2IF       ;are we done with this cycle?
      goto  wait              ;no we aren't
      goto  update            ;yes we are, do it again sam!

;*************** Subroutines ***************************************

; GetAD starts the A to D and loops until the acquisition is finished.
; The channel should be selected before calling and the A to D should
; be ON.

GetAD
      bcf   PIR1,ADIF   ;clear int flag
      bsf   ADCON0,GO   ;start new conversion
loop
      btfss PIR1,ADIF   ;a/d done?
      goto  loop        ;no then keep checking
      return

;InitializePORT, initializes and sets up the ports.
; Set I/O on ports

InitializePORTS
      bsf   STATUS,RP0  ;Bank 1
      movlw B'111111'   ;Port A 1 = input, 0 = output
      movwf TRISA       ;set port A I/O
      movlw B'11111111' ;Port B 1 = input, 0 = output
      movwf TRISB       ;set port B I/O
      movlw B'11111001' ;Port C 1 = input, 0 = output
      movwf TRISC       ;set port C I/O
      movlw B'11111111' ;Port D 1 = input, 0 = output
      movwf TRISD       ;set port D I/O
      movlw B'111'      ;Port E 1 = input, 0 = output
      movwf TRISE       ;set port E I/O
      bcf   STATUS,RP0  ;Bank 0
      return

;InitializeAD, initializes and sets up the A/D hardware.
;Select ch0 to ch7 as analog inputs.

InitializeAD
      bsf   STATUS,RP0  ;bank 1
      movlw B'00000000' ;select ch0-ch7...
      movwf ADCON1      ;as analog inputs
      bcf   STATUS,RP0  ;bank 0
```

```
      movlw B'11000001' ;select:RC,ch0..
      movwf ADCON0      ;turn on A/D.
      clrf  ADRES       ;clr result reg.
      return

;InitializePWM, initializes and sets up the PWM hardware.

InitializePWM
      movlw B'00000101' ;timer2 ON and 4:1 Prescale
      movwf T2CON       ;setup timer2
      bsf   STATUS,RP0  ;bank 1
      movlw H'FF'       ;value for 2.44Khz w/4:1 presacle
      movwf PR2         ;
      bcf   STATUS,RP0  ;bank 0
      movlw B'00111100' ;set the least sig bits to 11
      iorwf CCP1CON,F   ;on PWM 1 and set to PWM mode
      iorwf CCP2CON,F   ;on PWM 2 and set to PWM mode
      return


;This routine is a software delay of 10uS for the a/d setup.
;At 4Mhz clock, the loop takes 3uS, so initialize TEMp with
;a value of 3 to give 9uS, plus the move etc should result in
;a total time of > 10uS.

SetupDelay
      movlw .3
      movwf TEMP
SD
      decfsz      TEMP
      goto  SD
      return



      END                       ; directive 'end of program'
```

PWM Closed Loop Proportional Control Source Code

```
;*******************************************************************************
;
;      FILENAME:MOTOR1.ASM
;
;  FILE VERSION:V1.0
;
;FILES REQUIRED:p16c77.inc
;
;   DESCRIPTION:This program is setup to control the motors of the robot.
;
;       PURPOSE:Motor control of Robot.
;
;
;          NOTE:Used to develope PWM motor control and closed loop speed
;           control.
;
;
;  CHANGE HISTORY
;  Date      Author     Description
; --------  ---------   ----------------------------------------------------
; 03/24/99  T.Miller    Intial generation
; 03/27/99  T.Miller    Added PWM Code
; 04/06/99  T.Miller    Added code to close the loop useing encoders & timers
; 05/05/99  T.Miller    Added new constants and uses portB for Kp
;
;*******************************************************************************

;===============================================================================
; Instruct Assembler to assemble for a PIC16C77 configured as shown.
;===============================================================================

      list   p=16c77          ; list directive to define processor
      #include <p16c77.inc>   ; processor specific variable definitions

      __CONFIG   _CP_OFF & _WDT_OFF & _BODEN_ON & _PWRTE_ON & _XT_OSC

; '__CONFIG' directive is used to embed configuration data within .asm file.
; The lables following the directive are located in the respective .inc file.
; See respective data sheet for additional information on configuration word.


;===============================================================================
; Initialize constants
;===============================================================================


;***** A to D DEFINITIONS
; These defintions are used to selsect the A to D channel
; They are set to use the internal RC clock for conversion time
; They are also set to have the A to D turned on when the channel is
; is selected.  So, use the following lines to set the channel and
; turn the A to D on using the internal RC conversion clock.
;     movlw CHx               ; Where CHx is CH1, CH2 ...
;     movwf ADCON0            ;

CH0   EQU   B'11000001' ; Channel 0
CH1   EQU   B'11001001' ; Channel 1
CH2   EQU   B'11010001' ; Channel 2
```

```
CH3   EQU   B'11011001' ; Channel 3
CH4   EQU   B'11100001' ; Channel 4
CH5   EQU   B'11101001' ; Channel 5
CH6   EQU   B'11110001' ; Channel 6
CH7   EQU   B'11111001' ; Channel 7


;***** CONSTANT DEFINITIONS
;SPD        EQU   D'36'       ; desired speed -> Wd
;KSPD       EQU   D'156'      ; desired speed * constant -> K*Wd
CYCLE       EQU   D'20'       ; cytle time for feedback

;***** VARIABLE DEFINITIONS
w_temp      EQU     0x70            ; variable used for context saving
status_temp EQU     0x71      ; variable used for context saving


TEMP        EQU   20h   ; temp variable
WD0         EQU   21h   ; desired speed channel 0
WD1         EQU   22h   ; desired speed channel 1
WA0         EQU   23h   ; actual speed 0
WA1         EQU   24h   ; actual speed 1
WD          EQU   25h   ; desired speed for subroutine
WA          EQU   26h   ; actual speed for subroutine
KSPD        EQU   27h   ; desired speed * constant -> K*Wd
SPD0        EQU   28h   ; desired speed * constant -> K*Wd
SPD1        EQU   29h   ; desired speed * constant -> K*Wd
CYC_CNT     EQU   30h   ; cycle counter for feedback




;=============================================================================
; Set starting point in program ROM to zero.
;=============================================================================
      ORG   0x000             ; processor reset vector
      clrf  PCLATH            ; ensure page bits are claared
      goto  initial           ; go to beginning of program

;=============================================================================
; Interrupt Service Routine
;=============================================================================

      ORG   0x004             ; interrupt vector location
      movwf w_temp            ; save off current W register contents
      movf  STATUS,w          ; move status register into W register
      movwf status_temp       ; save off contents of STATUS register


; isr code can go here or be located as a call subroutine elsewhere


      movf  status_temp,w     ; retrieve copy of STATUS register
      movwf STATUS            ; restore pre-isr STATUS register contents
      swapf w_temp,f
      swapf w_temp,w    ; restore pre-isr W register contents
      retfie            ; return from interrupt
```

```
;=============================================================================
; Intail setup.
;=============================================================================

initial
      movlw D'144'      ;initial speed of motor
      movwf WD0         ;set inital speed
      movwf WD1         ;set inital speed

;=============================================================================
; Begin Main Body of Code
;=============================================================================
; The main loop will reinitialize all the ports at this time.
; If timing or other factors become an issue, the main loop can be
; modified accordingly.  It is good pratice to perodicaly reinitialize
; the configuration registers incase a glitch causes them to become upset.
;=============================================================================

main
      call  InitializePORTS
      call  InitializeAD
      call  InitializePWM

      movlw CYCLE       ;get number of cycles for feedback
      movwf CYC_CNT     ;load counter
      clrf  TMR0        ;clear timers
      clrf  TMR1L
      clrf  TMR1H

update
      bcf   PIR1,TMR2IF ;clear period flag

      movf  WD0,W       ;get speed seeting
      movwf CCPR1L      ;set dutycycle

      movf  WD1,W       ;get speed seeting
      movwf CCPR2L      ;set dutycycle

wait
      btfss PIR1,TMR2IF       ;are we done with this cycle?
      goto  wait              ;no we aren't
      decfsz      CYC_CNT,F   ;is it time to update speed?
      goto  update            ;no, use the same speed setting

      ;update speed variables WD0 and WD1
brk3  nop

      call  GetSpeed    ;get an update of desired
                        ;and actual speed

      movf  WA0,W       ;get actual speed 0
      movwf WA          ;and save it as actual speed
      movf  SPD0,W      ;get desired speed
      movwf KSPD        ;and save it as desired speed
      call  UpdateSpeed ;update the speed variable
      movf  WD,W        ;get the resultant WD
      movwf WD0         ;save the result
```

```
      movf  WA1,W       ;get actual speed 1
      movwf WA          ;and save it as actual speed
      movf  SPD1,W      ;get desired speed
      movwf KSPD        ;and save it as desired speed
      call  UpdateSpeed ;update the speed variable
      movf  WD,W        ;get the resultant WD
      movwf WD1         ;save the result

      goto  main        ;do it again sam!

;=============================================================================
; Subroutines
;=============================================================================

;***** UpdateSpeed
; UpdateSpeed, solves the following equation:
; %DC = K*Wd + Kp * (Wd - W)  where,
; %DC is the duty cycle which will equall the final WD in this routine
; K is 4 and Wd is 39 for our program.  KSPD is 4*39=156 and SPD = 39.
; Kp is 3 for our program.  W is the actual speed from the counters.
;
; I know this is a poor desription of this routine but in the interest
; of time I will leave this for a later description.

UpdateSpeed

      movf  KSPD,W      ;get desired motor speed w/ contant
      movwf WD          ;for calculation (K=2)
      bcf   STATUS,C    ;clear carry bit before rotate
      rrf   WD,F        ;rotate to divide by 2
      movf  WA,W        ;get actual speed
      subwf WD,F        ; WD = WD - WA
      movf  PORTB,W     ;Get Kp from port B
      movwf TEMP        ;TEMP is multiply counter
      movf  WD,W        ;get intial WD in W register

Again                   ; calculate Kp*(WD-WA)
      Decfsz      TEMP  ;are we done adding?
      goto  Add         ;no, so go Add
      goto  NewPWM      ;yes we are, now WD = Kp*(WD-WA)
Add
      addwf WD,F        ;now WD = WD + intial WD
      goto  Again       ; go see if we are done

NewPWM                  ;calculate WD = KSPD + WD
      movf  KSPD,W      ;get desired motor speed w/ contant
      addwf WD,F        ; now WD has the update value

      return

;***** GetSpeed
;GetSpeed, Update speed and get actual speed.
GetSpeed
      movlw CH0         ;setup for Ch 0
      movwf ADCON0      ;sets up  A to D
      call  GetAD       ;starts A to D
      movf  ADRES,W     ;get a/d value
```

```
      movwf SPD0        ;set speed 0
      movlw CH1         ;setup for Ch 0
      movwf ADCON0      ;sets up  A to D
      call  GetAD       ;starts A to D
      movf  ADRES,W     ;get a/d value
      movwf SPD1        ;set speed 1

      movf  TMR0,W      ;get timer 0 value
      movwf WA0         ;and save it as actual speed
      movf  TMR1L,W     ;get timer 1 value
      movwf WA1         ;and save it as actual speed

      return

;***** GetAD
; GetAD starts the A to D and loops until the acquisition is finished.
; The channel should be selected before calling and the A to D should
; be ON.

GetAD
      bcf   PIR1,ADIF   ;clear int flag
      bsf   ADCON0,GO   ;start new conversion
loop
      btfss PIR1,ADIF   ;a/d done?
      goto  loop        ;no, then keep checking
      return

;***** InitializePORT
;InitializePORT, initializes and sets up the ports.
; Set I/O on ports

InitializePORTS
      bsf   STATUS,RP0  ;Bank 1
      movlw B'111111'   ;Port A 1 = input, 0 = output
      movwf TRISA       ;set port A I/O
      movlw B'11111111' ;Port B 1 = input, 0 = output
      movwf TRISB       ;set port B I/O
      movlw B'11111001' ;Port C 1 = input, 0 = output
      movwf TRISC       ;set port C I/O
      movlw B'11111111' ;Port D 1 = input, 0 = output
      movwf TRISD       ;set port D I/O
      movlw B'111'      ;Port E 1 = input, 0 = output
      movwf TRISE       ;set port E I/O
      bcf   STATUS,RP0  ;Bank 0
      return

;***** InitializeAD
;InitializeAD, initializes and sets up the A/D hardware.
;Select ch0 to ch7 as analog inputs.

InitializeAD
      bsf   STATUS,RP0  ;bank 1
      movlw B'00000000' ;select ch0-ch7...
      movwf ADCON1      ;as analog inputs
      bcf   STATUS,RP0  ;bank 0
      movlw B'11000001' ;select:RC,ch0..
      movwf ADCON0      ;turn on A/D.
      clrf  ADRES       ;clr result reg.
```

```
      return

;***** InitializePWM
;InitializePWM, initializes and sets up the PWM and TMR hardware.

InitializePWM
      movlw B'00000101' ;timer2 ON and 4:1 Prescale
      movwf T2CON       ;setup timer2
      bsf   STATUS,RP0  ;bank 1
      movlw H'FF'       ;value for 2.44Khz w/4:1 presacle
      movwf PR2         ;
      movlw B'00101000' ;TMRO source to external rising edge,
      movwf OPTION_REG  ; Prescaler assigned to WDT
      bcf   STATUS,RP0  ;bank 0
      movlw B'00111100' ;set the least sig bits to 11
      iorwf CCP1CON,F   ;on PWM 1 and set to PWM mode
      iorwf CCP2CON,F   ;on PWM 2 and set to PWM mode
      movlw B'00000111' ;TMR1 to external
      movwf T1CON       ;
      return


;***** SetupDelay
;This routine is a software delay of 10uS for the a/d setup.
;At 4Mhz clock, the loop takes 3uS, so initialize TEMp with
;a value of 3 to give 9uS, plus the move etc should result in
;a total time of > 10uS.

SetupDelay
      movlw .3
      movwf TEMP
SD
      decfsz      TEMP,F
      goto  SD
      return

;==========================================================================
; End of Program
;==========================================================================


      END               ; directive 'end of program'
```

PWM Closed Loop
Proportional + Integral Control
Source Code

```
;*******************************************************************************
;
;       FILENAME: MOTOR1_1.ASM
;
;   FILE VERSION: V1.1
;
; FILES REQUIRED: p16c77.inc
;
;    DESCRIPTION: This program is setup to control the motors of the robot.
;
;        PURPOSE: Motor control of Robot.
;
;
;           NOTE: Used to develope PWM motor control and closed loop speed
;                 control.
;
;
;  CHANGE HISTORY
;  Date      Author     Description
; --------  ---------   -----------------------------------------------------
; 03/24/99  T.Miller    Intial generation
; 03/27/99  T.Miller    Added PWM Code
; 04/06/99  T.Miller    Added code to close the loop useing encoders & timers
; 05/05/99  T.Miller    Added new constants and uses portB for Kp
; 05/05/99  T.Miller    Implmentation of PI controller
;
;*******************************************************************************

;===============================================================================
; Instruct Assembler to assemble for a PIC16C77 configured as shown.
;===============================================================================

      list   p=16c77          ; list directive to define processor
      #include <p16c77.inc>   ; processor specific variable definitions

      __CONFIG   _CP_OFF & _WDT_OFF & _BODEN_ON & _PWRTE_ON & _XT_OSC

; '__CONFIG' directive is used to embed configuration data within .asm file.
; The lables following the directive are located in the respective .inc file.
; See respective data sheet for additional information on configuration word.


;===============================================================================
; Initialize constants
;===============================================================================


;***** A to D DEFINITIONS
; These defintions are used to selsect the A to D channel
; They are set to use the internal RC clock for conversion time
; They are also set to have the A to D turned on when the channel is
; is selected.  So, use the following lines to set the channel and
; turn the A to D on using the internal RC conversion clock.
;     movlw CHx               ; Where CHx is CH1, CH2 ...
;     movwf ADCON0            ;

CH0   EQU   B'11000001' ; Channel 0
CH1   EQU   B'11001001' ; Channel 1
```

```
CH2   EQU   B'11010001' ; Channel 2
CH3   EQU   B'11011001' ; Channel 3
CH4   EQU   B'11100001' ; Channel 4
CH5   EQU   B'11101001' ; Channel 5
CH6   EQU   B'11110001' ; Channel 6
CH7   EQU   B'11111001' ; Channel 7


;***** CONSTANT DEFINITIONS
;SPD        EQU   D'36'       ; desired speed -> Wd
;KSPD       EQU   D'156'      ; desired speed * constant -> K*Wd
CYCLE       EQU   D'20'       ; cytle time for feedback

;***** VARIABLE DEFINITIONS
w_temp      EQU     0x70      ; variable used for context saving
status_temp EQU     0x71      ; variable used for context saving


TEMP        EQU   20h   ; temp variable
WD0         EQU   21h   ; desired speed channel 0
WD1         EQU   22h   ; desired speed channel 1
WA0         EQU   23h   ; actual speed 0
WA1         EQU   24h   ; actual speed 1
ERR0        EQU   25h   ; errorsum 0
ERR1        EQU   26h   ; errorsum 1
WD          EQU   27h   ; desired speed for subroutine
WA          EQU   28h   ; actual speed for subroutine
ERR         EQU   29h   ; errorsum for subroutine
KSPD        EQU   30h   ; desired speed * constant -> K*Wd
SPD0        EQU   31h   ; desired speed * constant -> K*Wd
SPD1        EQU   32h   ; desired speed * constant -> K*Wd
CYC_CNT     EQU   33h   ; cycle counter for feedback




;=========================================================================
; Set starting point in program ROM to zero.
;=========================================================================
      ORG   0x000             ; processor reset vector
      clrf  PCLATH            ; ensure page bits are claared
      goto  initial           ; go to beginning of program

;=========================================================================
; Interrupt Service Routine
;=========================================================================

      ORG   0x004             ; interrupt vector location
      movwf w_temp            ; save off current W register contents
      movf  STATUS,w          ; move status register into W register
      movwf status_temp       ; save off contents of STATUS register


; isr code can go here or be located as a call subroutine elsewhere


      movf  status_temp,w     ; retrieve copy of STATUS register
```

```
      movwf STATUS            ; restore pre-isr STATUS register contents
      swapf w_temp,f
      swapf w_temp,w          ; restore pre-isr W register contents
      retfie                  ; return from interrupt

;=========================================================================
; Intail setup.
;=========================================================================

initial
      movlw D'144'            ;initial speed of motor
      movwf WD0               ;set inital speed
      movwf WD1               ;set inital speed
      clrf  ERR0              ;clear errorsum
      clrf  ERR1              ;clear errorsum
;=========================================================================
; Begin Main Body of Code
;=========================================================================
; The main loop will reinitialize all the ports at this time.
; If timing or other factors become an issue, the main loop can be
; modified accordingly.  It is good pratice to perodicaly reinitialize
; the configuration registers incase a glitch causes them to become upset.
;=========================================================================

main
      call  InitializePORTS
      call  InitializeAD
      call  InitializePWM

      movlw CYCLE             ;get number of cycles for feedback
      movwf CYC_CNT           ;load counter
      clrf  TMR0              ;clear timers
      clrf  TMR1L
      clrf  TMR1H

update
      bcf   PIR1,TMR2IF       ;clear period flag

      movf  WD0,W             ;get speed seeting
      movwf CCPR1L            ;set dutycycle

      movf  WD1,W             ;get speed seeting
      movwf CCPR2L            ;set dutycycle

wait
      btfss PIR1,TMR2IF       ;are we done with this cycle?
      goto  wait              ;no we aren't
      decfsz      CYC_CNT,F   ;is it time to update speed?
      goto  update            ;no, use the same speed setting

      ;update speed variables WD0 and WD1
brk3  nop

      call  GetSpeed          ;get an update of desired
                              ;and actual speed

      movf  WA0,W             ;get actual speed 0
      movwf WA                ;and save it as actual speed
```

```
      movf  SPD0,W            ;get desired speed
      movwf KSPD              ;and save it as desired speed
      movf  ERR0,W            ;get errorsum 0
      movwf ERR               ;and save it as errorsum
      call  UpdateSpeed       ;update the speed variable
      movf  WD,W              ;get the resultant WD
      movwf WD0               ;save the result
      movf  ERR,W             ;get updated errorsum
      movwf ERR0              ;and save it as errorsum 0

      movf  WA1,W             ;get actual speed 1
      movwf WA                ;and save it as actual speed
      movf  SPD1,W            ;get desired speed
      movwf KSPD              ;and save it as desired speed
      movf  ERR1,W            ;get errorsum 1
      movwf ERR               ;and save it as errorsum
      call  UpdateSpeed       ;update the speed variable
      movf  WD,W              ;get the resultant WD
      movwf WD1               ;save the result
      movf  ERR,W             ;get updated errorsum
      movwf ERR1              ;and save it as errorsum 1

      goto  main              ;do it again sam!

;=============================================================================
; Subroutines
;=============================================================================

;***** UpdateSpeed
; UpdateSpeed, solves the following equation:
; %DC = K*Wd + Kp * (Wd - W)  where,
; %DC is the duty cycle which will equall the final WD in this routine
; K is 4 and Wd is 39 for our program.  KSPD is 4*39=156 and SPD = 39.
; Kp is 3 for our program.  W is the actual speed from the counters.
;
; I know this is a poor desription of this routine but in the interest
; of time I will leave this for a later description.

UpdateSpeed

      movf  KSPD,W            ;get desired motor speed w/ contant
      movwf WD                ;for calculation (K=2)
      bcf   STATUS,C          ;clear carry bit before rotate
      rrf   WD,F              ;rotate to divide by 2
      movf  WA,W              ;get actual speed
      subwf WD,F              ; WD = WD - WA
      movf  ERR,W             ;get errorsum
      addwf WD,F              ;now WD = ERR+(WD-WA)
      movf  WD,W              ;get ready to update errorsum
      movwf ERR               ;and update it
      movf  PORTB,W           ;get Kp from port B
      movwf TEMP              ;TEMP is multiply counter
      movf  WD,W              ;get intial WD in W register

Again                         ; calculate Kp*(WD-WA)
      decfsz      TEMP        ;are we done adding?
      goto  Add               ;no, so go Add
      goto  NewPWM            ;yes we are, now WD = Kp*(WD-WA)
```

```
Add
      addwf WD,F              ;now WD = WD + intial WD
      goto  Again             ; go see if we are done

NewPWM                        ;calculate WD = KSPD + WD
      movf  KSPD,W            ;get desired motor speed w/ contant
      addwf WD,F              ; now WD has the update value

      return

;***** GetSpeed
;GetSpeed, Update speed and get actual speed.
GetSpeed
      movlw CH0               ;setup for Ch 0
      movwf ADCON0            ;sets up  A to D
      call  GetAD             ;starts A to D
      movf  ADRES,W           ;get a/d value
      movwf SPD0              ;set speed 0
      movlw CH1               ;setup for Ch 0
      movwf ADCON0            ;sets up  A to D
      call  GetAD             ;starts A to D
      movf  ADRES,W           ;get a/d value
      movwf SPD1              ;set speed 1

      movf  TMR0,W            ;get timer 0 value
      movwf WA0               ;and save it as actual speed
      movf  TMR1L,W           ;get timer 1 value
      movwf WA1               ;and save it as actual speed

      return

;***** GetAD
; GetAD starts the A to D and loops until the acquisition is finished.
; The channel should be selected before calling and the A to D should
; be ON.

GetAD
      bcf   PIR1,ADIF         ;clear int flag
      bsf   ADCON0,GO         ;start new conversion
loop
      btfss PIR1,ADIF         ;a/d done?
      goto  loop              ;no, then keep checking
      return

;***** InitializePORT
;InitializePORT, initializes and sets up the ports.
; Set I/O on ports

InitializePORTS
      bsf   STATUS,RP0        ;Bank 1
      movlw B'111111'         ;Port A 1 = input, 0 = output
      movwf TRISA             ;set port A I/O
      movlw B'11111111'       ;Port B 1 = input, 0 = output
      movwf TRISB             ;set port B I/O
      movlw B'11111001'       ;Port C 1 = input, 0 = output
      movwf TRISC             ;set port C I/O
      movlw B'11111111'       ;Port D 1 = input, 0 = output
      movwf TRISD             ;set port D I/O
```

```
      movlw B'111'            ;Port E 1 = input, 0 = output
      movwf TRISE             ;set port E I/O
      bcf   STATUS,RP0        ;Bank 0
      return

;***** InitializeAD
;InitializeAD, initializes and sets up the A/D hardware.
;Select ch0 to ch7 as analog inputs.

InitializeAD
      bsf   STATUS,RP0        ;bank 1
      movlw B'00000000'       ;select ch0-ch7...
      movwf ADCON1            ;as analog inputs
      bcf   STATUS,RP0        ;bank 0
      movlw B'11000001'       ;select:RC,ch0..
      movwf ADCON0            ;turn on A/D.
      clrf  ADRES             ;clr result reg.
      return

;***** InitializePWM
;InitializePWM, initializes and sets up the PWM and TMR hardware.

InitializePWM
      movlw B'00000101'       ;timer2 ON and 4:1 Prescale
      movwf T2CON             ;setup timer2
      bsf   STATUS,RP0        ;bank 1
      movlw H'FF'             ;value for 2.44Khz w/4:1 presacle
      movwf PR2               ;
      movlw B'00101000'       ;TMRO source to external rising edge,
      movwf OPTION_REG        ; Prescaler assigned to WDT
      bcf   STATUS,RP0        ;bank 0
      movlw B'00111100'       ;set the least sig bits to 11
      iorwf CCP1CON,F         ;on PWM 1 and set to PWM mode
      iorwf CCP2CON,F         ;on PWM 2 and set to PWM mode
      movlw B'00000111'       ;TMR1 to external
      movwf T1CON             ;
      return


;***** SetupDelay
;This routine is a software delay of 10uS for the a/d setup.
;At 4Mhz clock, the loop takes 3uS, so initialize TEMp with
;a value of 3 to give 9uS, plus the move etc should result in
;a total time of > 10uS.

SetupDelay
      movlw .3
      movwf TEMP
SD
      decfsz      TEMP,F
      goto  SD
      return

;=============================================================================
; End of Program
;=============================================================================
```

```
END                ; directive 'end of program'
```

Wall Following
Source Code

```
;*******************************************************************************
;
;      FILENAME:MOTOR2.ASM
;
;  FILE VERSION:V2.0
;
;FILES REQUIRED:p16c77.inc
;
;   DESCRIPTION:This program is setup to control the motors of the robot.
;
;       PURPOSE:Motor control of Robot.
;
;
;          NOTE:Used to develope PWM motor control and closed loop speed
;           control.
;
;
;  CHANGE HISTORY
;  Date      Author     Description
; --------  ---------   ----------------------------------------------------
; 03/24/99  T.Miller    Intial generation
; 03/27/99  T.Miller    Added PWM Code
; 04/06/99  T.Miller    Added code to close the loop useing encoders & timers
; 05/05/99  T.Miller    Added new constants and uses portB for Kp
; 05/06/99  T.Miller    Added Wall following anf interface control.  Still
;                       need to check for overflow problems on PWM settings.
;
;*******************************************************************************

;===============================================================================
; Instruct Assembler to assemble for a PIC16C77 configured as shown.
;===============================================================================

      list   p=16c77          ; list directive to define processor
      #include <p16c77.inc>   ; processor specific variable definitions

      __CONFIG   _CP_OFF & _WDT_OFF & _BODEN_ON & _PWRTE_ON & _XT_OSC

; '__CONFIG' directive is used to embed configuration data within .asm file.
; The lables following the directive are located in the respective .inc file.
; See respective data sheet for additional information on configuration word.


;===============================================================================
; Initialize constants
;===============================================================================


;***** A to D DEFINITIONS
; These defintions are used to selsect the A to D channel
; They are set to use the internal RC clock for conversion time
; They are also set to have the A to D turned on when the channel is
; is selected.  So, use the following lines to set the channel and
; turn the A to D on using the internal RC conversion clock.
;     movlw CHx         ; Where CHx is CH1, CH2 ...
;     movwf ADCON0            ;

CH0   EQU   B'11000001' ; Channel 0
```

```
CH1   EQU   B'11001001' ; Channel 1
CH2   EQU   B'11010001' ; Channel 2
CH3   EQU   B'11011001' ; Channel 3
CH4   EQU   B'11100001' ; Channel 4
CH5   EQU   B'11101001' ; Channel 5
CH6   EQU   B'11110001' ; Channel 6
CH7   EQU   B'11111001' ; Channel 7


;***** CONSTANT DEFINITIONS
LWD         EQU   D'114'      ; left wall distance
KSPDI       EQU   D'156'      ; desired speed * constant -> K*Wd
PWMCYC      EQU   D'20'       ; cycle time for feedback
WALCYC      EQU   D'6'        ; cycle time for Wall feedback

;----- bits
STOP        EQU   H'0007'     ;Stop bit
START       EQU   H'0000'     ;Start bit
FIRED       EQU   H'0002'     ;Fire Detect Bit
FIREE       EQU   H'0001'     ;Fire Extinquish
WLINE       EQU   H'0003'     ;White line detect

;----- Motor, Amp Gain, and others

FA0         EQU   B'00001001' ;Foward, Gain = High
FA1         EQU   B'00011001' ;Foward, Gain = Medium
FA2         EQU   B'00111001' ;Foward, Gain = Low

BA0         EQU   B'00000110' ;Bacward, Gain = High
BA1         EQU   B'00010110' ;Bacward, Gain = Medium
BA2         EQU   B'00110110' ;Bacward, Gain = Low

MSTP        EQU   B'00000000' ;Stop Motors, Gain = High
ALLSTP      EQU   B'00000000' ;Turn indecators and extinquisher off


;***** VARIABLE DEFINITIONS
w_temp      EQU     0x70            ; variable used for context saving
status_temp EQU     0x71      ; variable used for context saving


TEMP        EQU   20h   ; temp variable
WD0         EQU   21h   ; desired speed channel 0
WD1         EQU   22h   ; desired speed channel 1
WA0         EQU   23h   ; actual speed 0
WA1         EQU   24h   ; actual speed 1
ERR0        EQU   25h   ; errorsum 0
ERR1        EQU   26h   ; errorsum 1
WD          EQU   27h   ; desired speed for subroutine
WA          EQU   28h   ; actual speed for subroutine
ERR         EQU   29h   ; errorsum for subroutine
KSPD        EQU   30h   ; desired speed * constant -> K*Wd
SPD0        EQU   31h   ; desired speed * constant -> K*Wd
SPD1        EQU   32h   ; desired speed * constant -> K*Wd
PWM_CNT     EQU   33h   ; cycle counter for PWM feedback
WAL_CNT     EQU   33h   ; cycle counter for Wall feedback
LWERR       EQU   35h   ; Left Wall error
```

```
;=========================================================================
; Set starting point in program ROM to zero.
;=========================================================================
      ORG   0x000                   ; processor reset vector
      clrf  PCLATH                  ; ensure page bits are claared
      goto  initial                 ; go to beginning of program

;=========================================================================
; Interrupt Service Routine
;=========================================================================

      ORG   0x004             ; interrupt vector location
      movwf w_temp            ; save off current W register contents
      movf  STATUS,w          ; move status register into W register
      movwf status_temp       ; save off contents of STATUS register


; isr code can go here or be located as a call subroutine elsewhere


      movf  status_temp,w     ; retrieve copy of STATUS register
      movwf STATUS            ; restore pre-isr STATUS register contents
      swapf w_temp,f
      swapf w_temp,w          ; restore pre-isr W register contents
      retfie                  ; return from interrupt

;=========================================================================
; Intail setup.
;=========================================================================

initial
      movlw KSPDI       ;initial speed of motor
      movwf WD0         ;set inital speed
      movwf WD1         ;set inital speed
      movlw WAL_CYC     ;setup Wall interrupt cycle
      movwf WAL_CNT
      clrf  ERR0        ;clear errorsum
      clrf  ERR1        ;clear errorsum

      call  InitializePORTS
      call  InitializeAD
      call  InitializePWM
      call  Start       ;are we to start yet?

;=========================================================================
; Begin Main Body of Code
;=========================================================================
; The main loop will reinitialize all the ports at this time.
; If timing or other factors become an issue, the main loop can be
; modified accordingly.  It is good pratice to perodicaly reinitialize
; the configuration registers incase a glitch causes them to become upset.
;=========================================================================

main
      call  InitializePORTS
      call  InitializeAD
      call  InitializePWM
```

```
      movlw PWMCYC            ;get number of cycles for feedback
      movwf PWM_CNT           ;load counter
      clrf  TMR0              ;clear timers
      clrf  TMR1L
      clrf  TMR1H

update
      bcf   PIR1,TMR2IF       ;clear period flag

      movf  WD0,W             ;get speed seeting
      movwf CCPR1L            ;set dutycycle

      movf  WD1,W             ;get speed seeting
      movwf CCPR2L            ;set dutycycle

wait
      btfss PORTB,STOP        ;check if we are to stop
      call  Stop              ;goto stop routine
      btfss PIR1,TMR2IF       ;are we done with this cycle?
      goto  wait              ;no we aren't
      decfsz      PWM_CNT,F   ;is it time to update speed?
      goto  update            ;no, use the same speed setting

      ;update speed variables WD0 and WD1
brk3  nop

      decfsz      WAL_CNT     ;update wall distance?
      goto  UpdatePWM         ;just update PWM w/o wall
      call  GetSpeed          ;get an update of desired speed

      movf  TMR0,W            ;get timer 0 value
      movwf WA                ;and save it as actual speed
      movf  SPD0,W            ;get desired speed
      movwf KSPD              ;and save it as desired speed
      call  UpdateSpeed       ;update the speed variable
      movf  WD,W              ;get the resultant WD
      movwf WD0               ;save the result

      movf  TMR1L,W           ;get timer 1 value
      movwf WA                ;and save it as actual speed
      movf  SPD1,W            ;get desired speed
      movwf KSPD              ;and save it as desired speed
      call  UpdateSpeed       ;update the speed variable
      movf  WD,W              ;get the resultant WD
      movwf WD1               ;save the result

      goto  main              ;do it again sam!

;*****************************************************************************
; Subroutines
;*****************************************************************************

;***** Start
; Start, poles the Start bit to see if it has gone low.

Start
      btfsc PORTB,START ;test the start bit
```

```
      goto  Start       ;not cleared, keep checking
      return            ;ok, you can start now!

;***** Stop
; Stop, turns motors off and spins here

Stop
      movlw MSTP        ;get motor stop settings
      movwf PORTD       ;
      movlw ALLSTP      ;get all stop settings
      movwf PORTB       ;
      goto  Stop        ;just something todo
      return            ;to bad, I am never executed!

;***** UpdateSpeed
; UpdateSpeed, solves the following equation:
; %DC = K*Wd + Kp * (Wd - Wa)  where,
; %DC is the duty cycle which will equall the final WD in this routine.
; K is 2 and Kp = 1.  Wd is the desired speed which is = KSPD/2.
;
; I know this is a poor desription of this routine but in the interest
; of time I will leave this for a later description.

UpdateSpeed

      movf  KSPD,W      ;get desired motor speed w/ contant
      movwf WD          ;for calculation (K=2)
      bcf   STATUS,C    ;clear carry bit before rotate
      rrf   WD,F        ;rotate to divide by 2
      movf  WA,W        ;get actual speed
      subwf WD,F        ; WD = WD - WA
      movf  KSPD,W      ;get desired motor speed w/ contant
      addwf WD,F        ; now WD has the update value

      return

;***** GetSpeed
;GetSpeed, Update speed and get actual speed.
GetSpeed
      ;update speed registers
      movlw KSPD        ;get overall desired speed
      movwf SPD0        ;set speed 0
      movwf SPD1        ;set speed 1

      ;get wall distance and calculate error
      movlw LWD         ;get desired distance
      movwf LWERR       ;and set up for calculation

      movlw CH0         ;setup for Ch 0, Left Wall
      movwf ADCON0      ;sets up  A to D
      call  GetAD       ;starts A to D
      movf  ADRES,W     ;get a/d value
      subwf LWERR       ;calculate distance error

      ;update speeds
      movf  LWERR,F     ;get distance error
      subwf SPD0,F      ; SPDO = SPD0 - LWERR
      addwf SPD1,F      ; SPDO = SPD0 + LWERR
```

```
      movlw WAL_CYC     ;setup Wall interrupt cycle
      movwf WAL_CNT
      return

;***** GetAD
; GetAD starts the A to D and loops until the acquisition is finished.
; The channel should be selected before calling and the A to D should
; be ON.

GetAD
      bcf   PIR1,ADIF   ;clear int flag
      bsf   ADCON0,GO   ;start new conversion
loop
      btfss PIR1,ADIF   ;a/d done?
      goto  loop        ;no, then keep checking
      return

;***** InitializePORT
;InitializePORT, initializes and sets up the ports.
; Set I/O on ports

InitializePORTS
      bsf   STATUS,RP0  ;Bank 1
      movlw B'111111'   ;Port A 1 = input, 0 = output
      movwf TRISA       ;set port A I/O
      movlw B'11110001' ;Port B 1 = input, 0 = output
      movwf TRISB       ;set port B I/O
      movlw B'11111001' ;Port C 1 = input, 0 = output
      movwf TRISC       ;set port C I/O
      movlw B'11000000' ;Port D 1 = input, 0 = output
      movwf TRISD       ;set port D I/O
      movlw B'111'      ;Port E 1 = input, 0 = output
      movwf TRISE       ;set port E I/O
      bcf   STATUS,RP0  ;Bank 0
      return

;***** InitializeAD
;InitializeAD, initializes and sets up the A/D hardware.
;Select ch0 to ch7 as analog inputs.

InitializeAD
      bsf   STATUS,RP0  ;bank 1
      movlw B'00000000' ;select ch0-ch7...
      movwf ADCON1      ;as analog inputs
      bcf   STATUS,RP0  ;bank 0
      movlw B'11000001' ;select:RC,ch0..
      movwf ADCON0      ;turn on A/D.
      clrf  ADRES       ;clr result reg.
      return

;***** InitializePWM
;InitializePWM, initializes and sets up the PWM and TMR hardware.

InitializePWM
      movlw B'00000101' ;timer2 ON and 4:1 Prescale
      movwf T2CON       ;setup timer2
      bsf   STATUS,RP0  ;bank 1
```

```
      movlw H'FF'       ;value for 2.44Khz w/4:1 presacle
      movwf PR2         ;
      movlw B'00101000' ;TMRO source to external rising edge,
      movwf OPTION_REG  ; Prescaler assigned to WDT
      bcf   STATUS,RP0  ;bank 0
      movlw B'00111100' ;set the least sig bits to 11
      iorwf CCP1CON,F   ;on PWM 1 and set to PWM mode
      iorwf CCP2CON,F   ;on PWM 2 and set to PWM mode
      movlw B'00000111' ;TMR1 to external
      movwf T1CON       ;
      return


;***** SetupDelay
;This routine is a software delay of 10uS for the a/d setup.
;At 4Mhz clock, the loop takes 3uS, so initialize TEMp with
;a value of 3 to give 9uS, plus the move etc should result in
;a total time of > 10uS.

SetupDelay
      movlw .3
      movwf TEMP
SD
      decfsz      TEMP,F
      goto  SD
      return

;=============================================================================
; End of Program
;=============================================================================


      END               ; directive 'end of program'
```