Microprocessor Systems Final Submission Report

Line Follower Mobile Robot

Name: Mohammed Ihab Elmetwally

ID: 223199

Module Name: Microprocessor Systems

Module Code: 24MTRN15I

Module Leader: Dr. Nourhan Zayed

Teaching Assistance: Eng. Nardin Henawy

Eng. Abdulrahman Said

YR3 Mechatronics and Robotics Engineering Program

Table of Content

Introduction

In the current era of robots and automation, developing autonomous systems that have the ability to carry out pre-programmed tasks on their own without the help of humans has been of significant interest. One of the simplest yet most commonly used autonomous systems is the line follower robot. As suggested by the name, a line follower is a mobile robot that follows a line—a white line on black or vice versa—based either on infrared or optical sensors. It is applied in industrial material handling, warehouse automation and assembly line automation, and even in educational environments as an education platform for studying robotics and embedded systems.

The line follower idea is straightforward but very educational. The robot will be able to drive along the line through the use of sensors and adjust in the moment to stay on the line. The realization of this activity, however, entails an interfacing of many different pieces of hardware and software such as sensors, actuators, power supplies, and a microcontroller that serves as the robotic brain.

For this project, a PIC microcontroller is utilized as the processing unit due to its ability, ruggedness, and cost-effectiveness. PIC microcontrollers have a reputation of being high-performance devices and are well supported for real-time control applications. All of these features of PIC microcontrollers qualify them for the design of control algorithms required for a line following application. The microcontroller converts the sensor reading and regulates the motors appropriately such that the robot follows them to the letter.

The report below will be a step-by-step guide to the design and construction of a PIC microcontroller-based line follower robot. Hardware development, including sensor selection, motor drivers, and chassis, will be described. Issues that were encountered in the development and suggestions for possible improvements in future versions will also be stated.

Problem Statement

Constructing a line follower robot presents several technical issues concerning both hardware and software that must be tackled carefully. The robot must be capable of precisely sensing and following a black line on a differently colored background, including sharp turns, intersections, or slight deviations on the track. This requires a sensitive and accurate sensing mechanism that is typically attained by using infrared (IR) sensors. These sensors need to continuously provide real-time input to the microcontroller, which decodes the data and causes the robot to change its motion accordingly.

Apart from following the line, the robot should also be able to handle dynamic environments. One of the most crucial aspects in this direction is obstacle detection. With the addition of a proximity sensor, the robot is able to sense objects in its path that might distract its main function. The sensor is connected to the microcontroller through an interrupt pin so that the robot can react immediately to unexpected obstacles without any delay, thus enhancing safety and reliability.

Another essential requirement is the delivery of motor speed and direction control in a smooth manner. This is achieved through the use of Pulse Width Modulation (PWM), which allows the microcontroller to supply power to the motors with high precision. With an H-bridge motor driver, the robot is capable of moving forward and backward as well as turning at regulated speed, which is of high necessity in achieving line tracking accuracy.

At the heart of the system is the PIC microcontroller, the central processing unit. It takes inputs from the IR and proximity sensors, executes decision-making logic, and generates the PWM signals to move the motors. All this has to be done in real time and require effective programming and innovative use of microcontroller resources.

The overall goal of the project in this submission is to design a robot that is efficient under common indoor operating conditions. The robot should demonstrate the usage and the process of actuation, control, and sensing through the application of fundamental principles of robotics and embedded systems.

Data

For the purpose of examination of the line follower robot performance, readings will be taken from various sensors utilized as system inputs. The two infrared (IR) sensors at the front are utilized as the primary input to detect the line. These two sensors measure and report the position of the robot on the black line. These readings-analog or digital depending on configuration—are monitored to gauge the precision of robot decision-making and the extent to which it stays on course.

The proximity sensor, connected to an interrupt pin on the microcontroller, is employed for input towards obstacle detection. The sensor gives information to monitor frequency and response rate of the robot to obstacles. Interrupt activation timestamps and motor response times are stored to quantify system responsiveness.

Also, a Sharp IR distance sensor is utilized to provide even more precise distance measurements in front of the robot. The sensor enhances the detection of obstacles and can be used to measure the deceleration or reaction of the robot at various distances. The output voltage of the Sharp sensor is monitored and logged to evaluate the performance of how well the robot evaluates the proximity of obstacles.

These reading inputs—obtained from a sequence of test situations—will be the basis for determining the robot's ability to drive straight in accurate manner, drive around obstacles, and drive reliably in real time.

Evaluation Criteria

Performance of the line follower robot will be evaluated on several important functional and behavioral parameters to determine the effectiveness and dependability of the system. Parameters aims to hardware and software performance under real operating conditions. Line-following accuracy is one of the main parameters that indicate how well the robot can stay along the black line through curves and intersections. Any deviation or overshoot will be detected and analyzed.

Response time matters too—namely, how quickly the robot resonds to changes in sensor input and alters direction. That includes the ADC sampling rate as well as how well the motor control responds to logic change. Obstacle avoidance performance is also a critical requirement, measured by how quickly and repeatedly the robot slows or deviates when detecting an obstacle via the interrupt-based proximity sensor system.

PWM smoothness and stability of motor control will also be checked to ensure the robot is not uneven or jerky when traveling in turns or in corrections. Additionally, long-term test-based analysis of system reliability over time will verify that the robot always responds without reset or failure. All evaluation will be performed on a standardized indoor light and floor environment to ensure fairness.

Approach

To address the issues of autonomous line following and obstacle detection, the approach was systematic with modular hardware and software development. The project was divided into four fundamental functional blocks: line detection, obstacle detection, motor control, and central decision-making logic. The modules were developed, individually tested, and then integrated into a complete system on a PIC16F877A microcontroller.

Line tracking was achieved by using two infrared (IR) sensors at the front of the robot to detect contrast between a dark line and the lighter floor surface. The IR sensors provided analog output voltages proportional to surface reflectivity, which were read by the PIC using its onboard 10-bit Analog-to-Digital Converter (ADC). The ADC configuration was done using the ADCON0 and ADCON1 registers. ADCON0 was used for selecting the input channel and enabling the ADC module, while ADCON1 established the justification and voltage reference configurations. The microcontroller alternately sampled the two channels connected to the IR sensors and compared their values to determine whether the robot was veering off course. Based on this data, it adjusted motor speed or direction accordingly.

Obstacle detection was managed using a digital proximity sensor interfaced to one of the external interrupt pins of the PIC, typically INT0 (RB0/INT). This allowed the robot to react immediately to obstacles at close range by calling an interrupt service routine (ISR) on sensing a falling or rising edge. Upon interrupt, the microcontroller halted the line-following routine temporarily and executed a pre-programmed obstacle avoidance routine, i.e., halting both motors or reversing briefly. Interrupts were enabled by utilizing the INTCON register, where the INTE (INT0 External Interrupt Enable) and GIE (Global Interrupt Enable) bits were activated.

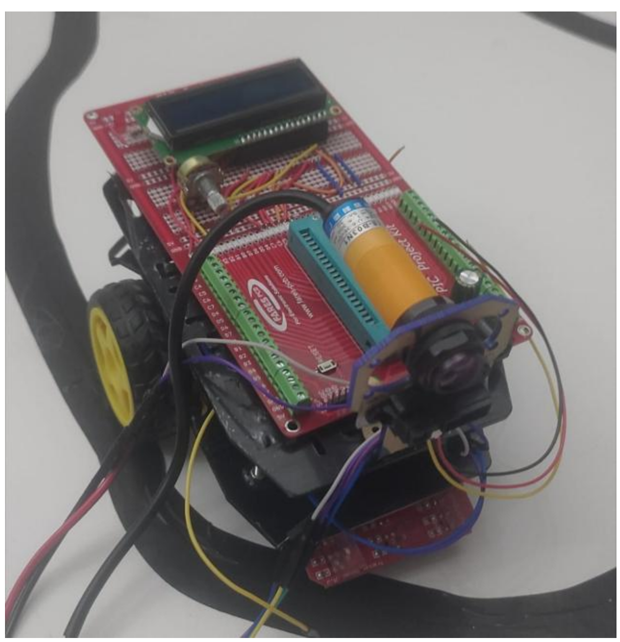
Motor control was done using an H-bridge circuit for bi-directional control of two DC motors. Speed control was achieved by software-generated Pulse Width Modulation (PWM) signals. For duty cycle and timing control, the TMR2 timer module of the microcontroller and the CCP (Capture/Compare/PWM) module were used. The CCP1CON register was configured for PWM mode, and the duty cycle was varied by writing to the CCPR1L register and the two least significant bits of CCP1CON. This provided fine-tuned control of the motor speed, which enabled the robot to adjust its speed while turning or correcting.

The control software was programmed completely in Assembly language for maximum speed and low resource usage. Low-level routines were developed for initializing and managing ADC channels, configuring timers and PWM modules, and handling interrupt vectors. High-level routines coordinated sensor inputs and actuator outputs with conditional logic to maintain the robot on the track and respond to unexpected objects. The robot was incrementally tested at each stage—starting with individual module testing (e.g., sensor calibration, PWM generation) to complete system integration in real environments—both for reliability and responsiveness.

Hardware Implementation

Components

1x PIC16F877A	1x L298N H-Bridge
1x 4MHz Oscillator Crytal	2x DC Motors
2x 27pF Capacitors	2x Infrared Sensor
1x Push Button	1x Proximity Sensor
1x LED	1x Sharp IR Sensor
1x 10K Ω Resistor	1x LCD
1x 220 Ω Resistor	12V Power Supply
1x Buck Converter (12V to 5V)	



Development

While the current application of the line follower robot is effective in carrying out its core functions, there are several places where the system can be extended and developed further in an attempt to maximize performance, flexibility, and intelligence.

One aspect of improvement is the adding more infrared sensors. In my project, the robot only has two IR sensors, which limits its ability in handling sharp turns or complex paths. So, By increasing the number of sensors to three for example, the system would gain a better sense of the position of the robot relative to the line, resulting in more stable and accurate decisions.

Another major upgrade would be in the obstacle avoidance system. Instead of only braking on sensing an obstacle, the robot could be programmed make choices such as going around the obstacle or taking a temporary time-out before trying again. To accomplish this, additional proximity sensors (e.g., on the sides) can be installed or ultrasonic sensors can be used for increased detection area and improved distance estimates.

Further, the usage of closed-loop feedback by using rotary encoders would allow the robot to monitor its wheel revolutions and control its speed regardless of surface friction. This will enhance the motion and make it more accurate, especially in tight turns or recovery maneuvers.

On the software side, better decision-making can be done using more advanced algorithms for example by adding PID control for line following results in more smoother motion compared to basic conditional control.

Overall, the robot provides a good foundation for experimentation and learning but also lots of potential for expansion in hardware, software, and system capabilities.

Conclusion

The line follower robot built in this project is well able to demonstrate the fundamentals of a PIC microcontroller in embedded robotic systems. Through line detection and obstacle avoidance, the robot is able to move automatically along a pre-programmed path with environmental sensitivity. The application of interrupts and PWM demonstrates elementary microprocessor system concepts.

References

- Microchip Technology Inc. (2003). *PIC16F87XA Data Sheet: 28/40/44-Pin Enhanced Flash Microcontrollers* (Rev. C) [PDF]. Retrieved from https://ww1.microchip.com/downloads/en/DeviceDoc/39582c.pdf

Appendix

Simulation

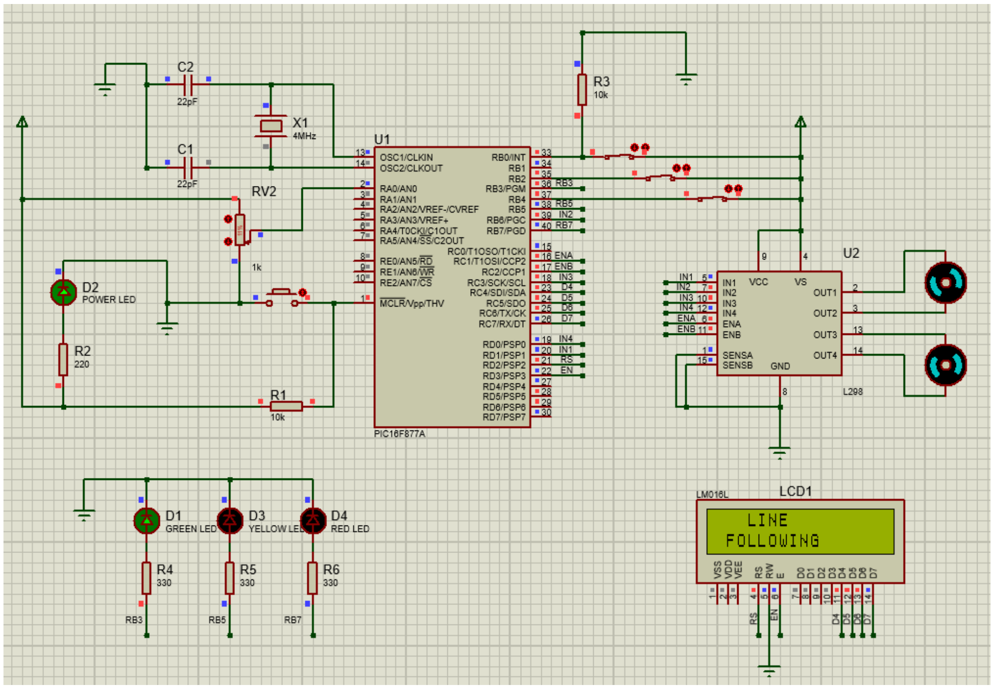


Figure 1: The robot is following the line

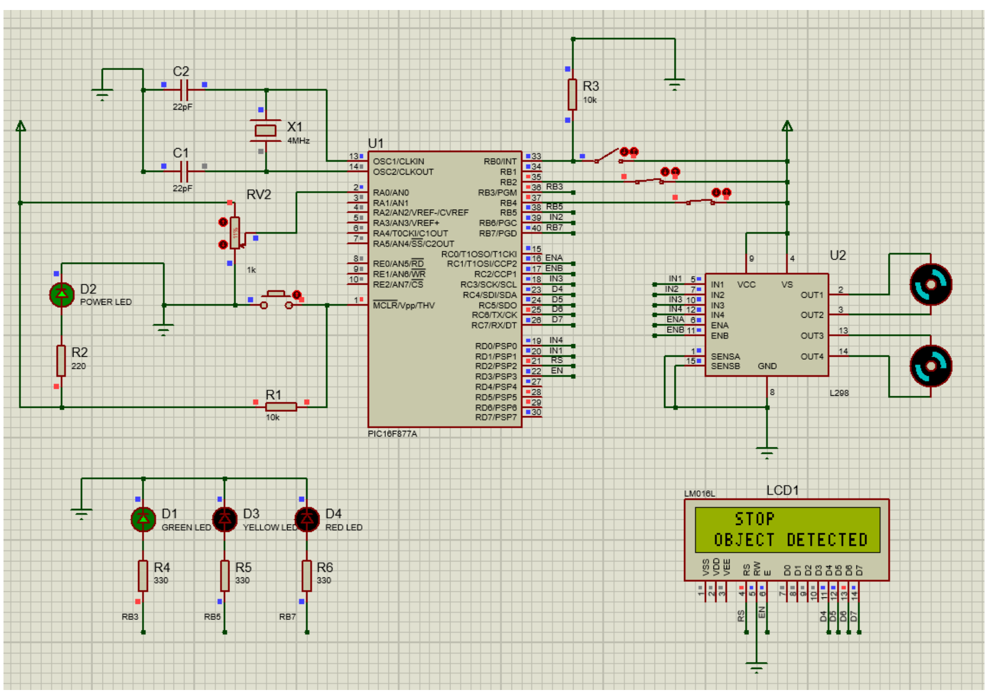


Figure 2: The robot detected obstacle (Interrupt)

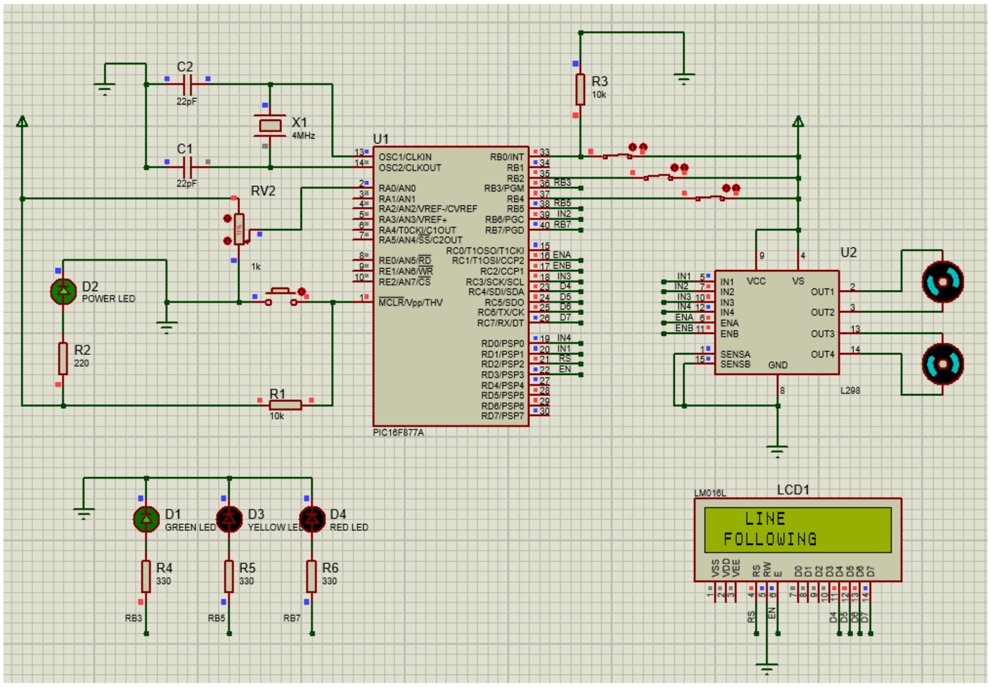


Figure 3: The sharp sensor detects obstacle from long range (GREEN LED ON)

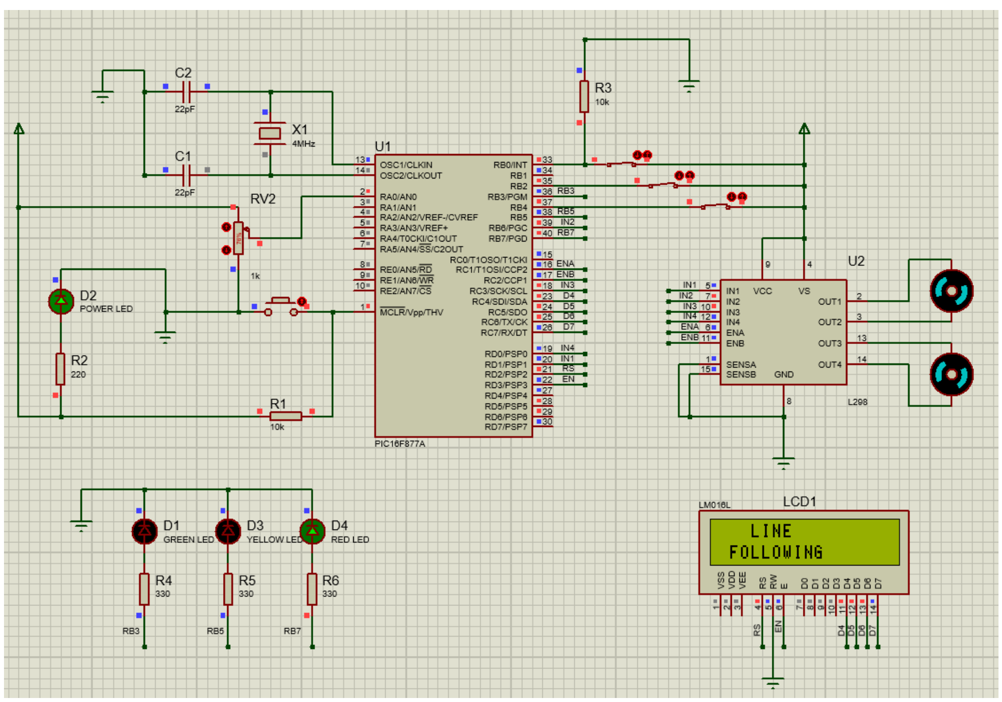


Figure 4: The sharp sensor detects obstacle from close range (RED LED ON)

Code

```
;===================================
;    Author: Mohammed Ihab Elmetwally
;
;    Date: 24/5/2025
;
;    PIC Version: PIC16F877A
;
;    Title: Line Follower mobile robot
;
;    Description: Line follower mobile robot control using two ir
;    sensors, L298N H-Bridge, Proximity sensor, sharp ir, and LCD
;====================================



     LIST P=16F877A
    #include <P16F877A.INC>


    __CONFIG _CP_OFF & _WDT_OFF & _XT_OSC & _PWRTE_ON & _BODEN_OFF
& _LVP_OFF


     ORG 0x00


; Variables
     CBLOCK 0x20
     COUNT1
     COUNT2
```

```
	TEMP
	NIBBLE
		ADC_READING_H
		ADC_READING_L
	ENDC


; LCD Control Pins
LCD_RS      EQU 2    ; PORTD.2
LCD_E       EQU 3    ; PORTD.3
LCD_DATA1    EQU 0x0F0 ; PORTC bits 4-7 (mask)

; LCD Commands
LCD_CLEAR  EQU 0x01
LCD_HOME   EQU 0x02
LCD_4BIT   EQU 0x28
LCD_ON     EQU 0x0C
LCD_ENTRY  EQU 0x06

	GOTO SETUP

	ORG 0x04

	CALL DISPLAY_STOPPED
STOP_INT:
	BCF INTCON, 1
	BCF PORTD, 1      ; IN2 = 0
```

```
    BCF PORTB, 6      ; IN1 = 0

    BCF PORTD, 0      ; IN3 = 0
   BCF PORTC, 3      ; IN4 = 0

   ; Set speed to 0% on both motors
    MOVLW .0
   MOVWF CCPR1L      ; RC2 PWM
   MOVWF CCPR2L      ; RC1 PWM

WAIT_FOR_LOW:
    BTFSS PORTB, 0
    GOTO WAIT_FOR_LOW
    CALL DISPLAY_MOVING
    RETFIE

SETUP:
    ; Setup Phase
    BSF INTCON, 7
    BSF INTCON, 4
    BCF INTCON, 1

    ; Select Bank 1
   BCF STATUS, 6
   BSF STATUS, 5
    BCF OPTION_REG, INTEDG
```

```
  MOVLW   B'11000000'   ; Right-justified, VDD/VSS reference
 MOVWF   ADCON1

 MOVLW b'00010101'   ; Set RB0, RB2, RB4 as INPUTS, others are
OUTPUT
 MOVWF TRISB
  MOVLW b'00000000'   ; Set PORTC as OUTPUT
 MOVWF TRISC
  MOVLW b'00000000'   ; Set PORTD as OUTPUT
 MOVWF TRISD
 BCF STATUS, 5        ; Back to Bank 0

  BCF PORTB, 3
  BCF PORTB, 5
  BCF PORTB, 7

  BCF PORTD, 1      ; IN2 = 0
  BCF PORTB, 6      ; IN1 = 0

  BCF PORTD, 0      ; IN3 = 0
 BCF PORTC, 3      ; IN4 = 0

  MOVLW   B'01000001'   ; ADC ON, Fosc/16, Channel 0 (AN0)
 MOVWF   ADCON0
 CALL    Delay_20us    ; Wait for acquisition time
```

```
 ; Set PWM mode
 MOVLW b'00001100'   ; CCP1 in PWM mode
 MOVWF CCP1CON

 MOVLW b'00001100'   ; CCP2 in PWM mode
 MOVWF CCP2CON

 ; Set PWM period
 MOVLW .249          ; 20kHz PWM
 MOVWF PR2

 ; Set duty cycle
 MOVLW .128
 MOVWF CCPR1L        ; 50% duty on RC2
 MOVLW .128
 MOVWF CCPR2L        ; 50% duty on RC1

 ; Set TMR2
 MOVLW b'00000111'   ; Prescaler = 16
 MOVWF T2CON

 BSF T2CON, 2        ; Turn on TMR2

; Initialize LCD with more robust sequence
CALL LCD_INIT
```

```
	CALL DISPLAY_MOVING

START:
	; Main Loop Logic Behavior
	CALL    READ_ADC      ; Get ADC value
	CALL    CHECK_DISTANCE ; Update LED

	BTFSC PORTB, 2
	GOTO CHECK_LEFT
	GOTO CHECK_RIGHT

STOP:
	BCF PORTD, 1      ; IN2 = 0
	BCF PORTB, 6      ; IN1 = 0

	BCF PORTD, 0      ; IN3 = 0
	BCF PORTC, 3      ; IN4 = 0

	; Set speed to 0%
	MOVLW .0
	MOVWF CCPR1L      ; RC2 PWM
	MOVWF CCPR2L      ; RC1 PWM
	GOTO START

MOVE_FORWARD:
	BCF PORTD, 1      ; IN1 = 0
	BSF PORTB, 6      ; IN2 = 1
```

```
     BCF PORTD, 0      ; IN3 = 0
    BSF PORTC, 3      ; IN4 = 1


    ; Set speed
    MOVLW .70
    MOVWF CCPR1L      ; RC2 PWM
    MOVWF CCPR2L      ; RC1 PWM
    GOTO START

MOVE_LEFT:
     BCF PORTD, 1      ; IN1 = 0
     BSF PORTB, 6      ; IN2 = 1

     BCF PORTC, 3      ; IN4 = 0
     BSF PORTD, 0      ; IN3 = 1

    ; Set speed
    MOVLW .60
    MOVWF CCPR1L      ; RC2 PWM
    MOVWF CCPR2L      ; RC1 PWM
    GOTO START

MOVE_RIGHT:
     BCF PORTB, 6      ; IN2 = 0
    BSF PORTD, 1      ; IN1 = 1
```

```
     BCF PORTD, 0      ; IN3 = 0
     BSF PORTC, 3      ; IN4 = 1

    ; Set speed
    MOVLW .60
    MOVWF CCPR1L      ; RC2 PWM
    MOVWF CCPR2L      ; RC1 PWM
    GOTO START

CHECK_LEFT:
     BTFSC PORTB, 4
     GOTO MOVE_FORWARD
     GOTO MOVE_RIGHT

CHECK_RIGHT:
     BTFSC PORTB, 4
     GOTO MOVE_LEFT
     GOTO STOP

READ_ADC:
    BSF     ADCON0, GO    ; Start conversion
WAIT_READING:
    BTFSC   ADCON0, GO    ; Wait tell reading
    GOTO    WAIT_READING
    MOVF    ADRESH, W     ; Store high byte
    MOVWF   ADC_READING_H
```

```
     BANKSEL ADRESL
    MOVF    ADRESL, W     ; Store low byte
    MOVWF   ADC_READING_L
    BCF STATUS, 5
     RETURN


CHECK_DISTANCE:
    ;----------------------------
    ; Step 1: If ADC > 478 ? RED
    ;----------------------------
    MOVF    ADC_READING_H, W
    SUBLW   0x01
    BTFSS   STATUS, C         ; if ADC_H < 0x01 ? cannot be >478
    GOTO    CHECK_GREEN
    BTFSS   STATUS, Z         ; if ADC_H > 0x01 ? definitely >478
    GOTO    GREEN_LED_ON

    ; If ADC_H == 0x01, check low byte
    MOVF    ADC_READING_L, W
    SUBLW   0xDE              ; check if L > 0xDE
    BTFSS   STATUS, C         ; if ADC_L > 0xDE ? ADC > 478
    GOTO    GREEN_LED_ON

CHECK_GREEN:
    ;----------------------------
    ; Step 2: If ADC < 342 ? GREEN
```

```
    ;---------------------------
    MOVF    ADC_READING_H, W
    SUBLW   0x01
    BTFSC   STATUS, C         ; if ADC_H < 0x01 ? definitely <342
    GOTO    YELLOW_LED_ON
    BTFSS   STATUS, Z         ; if ADC_H > 0x01 ? definitely >342
    GOTO    RED_LED_ON

    ; ADC_H == 0x01 ? check low byte
    MOVF    ADC_READING_L, W
    SUBLW   0x56              ; check if L < 0x56
    BTFSC   STATUS, C         ; if ADC_L < 0x56 ? ADC < 342
    GOTO    YELLOW_LED_ON

    ;---------------------------
    ; Step 3: Else ? YELLOW
    ;---------------------------
    GOTO    RED_LED_ON

ADC_FINISH:
    RETURN

GREEN_LED_ON:
    BCF PORTB, 5
    BCF PORTB, 7
    BSF PORTB, 3
    GOTO ADC_FINISH
```

```
YELLOW_LED_ON:
    BCF PORTB, 3
    BCF PORTB, 7
    BSF PORTB, 5
    GOTO ADC_FINISH

RED_LED_ON:
    BCF PORTB, 3
    BCF PORTB, 5
    BSF PORTB, 7
    GOTO ADC_FINISH

; Modified LCD Initialization
LCD_INIT:
    ; Extended power-up delay (minimum 100ms to be safe)
    CALL DELAY_50MS
    CALL DELAY_50MS
    CALL DELAY_50MS

    ; First initialization sequence (2-bit mode)
    BCF STATUS, 6    ; Bank 0
    BCF STATUS, 5

    ; Send 0x03 three times with longer delays
    MOVLW 0x30
    CALL LCD_SEND_INIT
```

```
CALL DELAY_10MS

MOVLW 0x30
CALL LCD_SEND_INIT
CALL DELAY_10MS

MOVLW 0x30
CALL LCD_SEND_INIT
CALL DELAY_10MS

; Switch to 4-bit mode
MOVLW 0x20
CALL LCD_SEND_INIT
CALL DELAY_10MS

; Now in 4-bit mode - send function set
MOVLW LCD_4BIT     ; 4-bit, 2-line, 5x8
CALL LCD_CMD
CALL DELAY_5MS

MOVLW LCD_ON       ; Display on, cursor off, blink off
CALL LCD_CMD
CALL DELAY_5MS

MOVLW LCD_CLEAR    ; Clear display
CALL LCD_CMD
CALL DELAY_5MS
```

```
    MOVLW LCD_ENTRY    ; Entry mode: increment, no shift
    CALL LCD_CMD
    CALL DELAY_5MS

    RETURN

; Special initialization send
LCD_SEND_INIT:
    BCF PORTD, LCD_E    ; Ensure E is low
    BCF PORTD, LCD_RS   ; Command mode

    ; Send upper nibble to all 8 data lines (for initialization)
    MOVWF TEMP
    MOVF TEMP,W
    ANDLW 0xF0
    MOVWF PORTC         ; Send to all data lines

    ; Pulse Enable
    BSF PORTD, LCD_E
    NOP
    NOP
    BCF PORTD, LCD_E

    ; Wait for command to complete
    CALL DELAY_1MS
    RETURN
```

```
; Send command to LCD (4-bit mode)
LCD_CMD:
    BCF PORTD, LCD_RS   ; Command mode
    GOTO LCD_SEND_4BIT

; Send data to LCD (4-bit mode)
LCD_SEND_DATA:
    BSF PORTD, LCD_RS   ; Data mode

LCD_SEND_4BIT:
    MOVWF TEMP
    ; Send upper nibble first
    SWAPF TEMP,W
    CALL LCD_SEND_NIBBLE
    ; Send lower nibble
    MOVF TEMP,W
    CALL LCD_SEND_NIBBLE
    ; Short delay for command execution
    CALL DELAY_100US
    RETURN

; Send a nibble to LCD
LCD_SEND_NIBBLE:
    ANDLW 0x0F          ; Only keep lower 4 bits
    MOVWF NIBBLE
```

```
    ; Bank selection for PORTC
    BCF STATUS, RP0     ; Bank 0
    BCF STATUS, RP1

    ; Clear data bits first
    MOVLW 0x0F
    ANDWF PORTC,F

    ; Set data bits according to nibble
    BTFSC NIBBLE, 0
    BSF PORTC, 4
    BTFSC NIBBLE, 1
    BSF PORTC, 5
    BTFSC NIBBLE, 2
    BSF PORTC, 6
    BTFSC NIBBLE, 3
    BSF PORTC, 7

    ; Pulse Enable
    BSF PORTD, LCD_E
    NOP                 ; 1µs delay
    NOP                 ; 1µs delay
    BCF PORTD, LCD_E
    RETURN


DISPLAY_MOVING:
```

```
 MOVLW   0x01         ; First line address (0x00)
CALL    LCD_CMD

 MOVLW   0x80         ; First line address (0x00)
CALL    LCD_CMD

MOVLW   ' '
CALL    LCD_SEND_DATA
 MOVLW   ' '
CALL    LCD_SEND_DATA
 MOVLW   ' '
CALL    LCD_SEND_DATA
 MOVLW   'L'
CALL    LCD_SEND_DATA
MOVLW   'I'
CALL    LCD_SEND_DATA
 MOVLW   'N'
CALL    LCD_SEND_DATA
 MOVLW   'E'
CALL    LCD_SEND_DATA

; Display second line
MOVLW   0xC0         ; Second line address (0x40)
CALL    LCD_CMD

 MOVLW   ' '
CALL    LCD_SEND_DATA
```

```
    MOVLW   'F'
    CALL    LCD_SEND_DATA
    MOVLW   'O'
    CALL    LCD_SEND_DATA
    MOVLW   'L'
    CALL    LCD_SEND_DATA
     MOVLW   'L'
    CALL    LCD_SEND_DATA
     MOVLW   'O'
    CALL    LCD_SEND_DATA
     MOVLW   'W'
    CALL    LCD_SEND_DATA
     MOVLW   'I'
    CALL    LCD_SEND_DATA
     MOVLW   'N'
    CALL    LCD_SEND_DATA
     MOVLW   'G'
    CALL    LCD_SEND_DATA

     RETURN

DISPLAY_STOPPED:
     MOVLW   0x01         ; First line address (0x00)
    CALL    LCD_CMD

     MOVLW   0x80         ; First line address (0x00)
    CALL    LCD_CMD
```

```
MOVLW   ' '
CALL    LCD_SEND_DATA
 MOVLW   ' '
CALL    LCD_SEND_DATA
 MOVLW   ' '
CALL    LCD_SEND_DATA
MOVLW   'S'
CALL    LCD_SEND_DATA
 MOVLW   'T'
CALL    LCD_SEND_DATA
MOVLW   'O'
CALL    LCD_SEND_DATA
 MOVLW   'P'
CALL    LCD_SEND_DATA

; Display second line
MOVLW   0xC0          ; Second line address (0x40)
CALL    LCD_CMD

 MOVLW   ' '
CALL    LCD_SEND_DATA
MOVLW   'O'
CALL    LCD_SEND_DATA
MOVLW   'B'
CALL    LCD_SEND_DATA
MOVLW   'J'
```

```
CALL    LCD_SEND_DATA
 MOVLW   'E'
CALL    LCD_SEND_DATA
 MOVLW   'C'
CALL    LCD_SEND_DATA
 MOVLW   'T'
CALL    LCD_SEND_DATA
 MOVLW   ' '
CALL    LCD_SEND_DATA
 MOVLW   'D'
CALL    LCD_SEND_DATA
 MOVLW   'E'
CALL    LCD_SEND_DATA
 MOVLW   'T'
CALL    LCD_SEND_DATA
 MOVLW   'E'
CALL    LCD_SEND_DATA
 MOVLW   'C'
CALL    LCD_SEND_DATA
 MOVLW   'T'
CALL    LCD_SEND_DATA
 MOVLW   'E'
CALL    LCD_SEND_DATA
 MOVLW   'D'
CALL    LCD_SEND_DATA

 RETURN
```

```
; DELAY FUNCTIONS


; --- 50ms Delay ---
DELAY_50MS:
    MOVLW   D'200'       ; 200 loops
    MOVWF   COUNT1
DELAY_50MS_1:
    MOVLW   D'250'       ; 250 inner loops
    MOVWF   COUNT2
DELAY_50MS_2:
    NOP                  ; 1µs
    DECFSZ  COUNT2,F     ; 1µs (2µs when last)
    GOTO    DELAY_50MS_2 ; 2µs
    DECFSZ  COUNT1,F     ; 1µs
    GOTO    DELAY_50MS_1 ; 2µs
    RETURN               ; Total: 200*(250*3 + 3) + 2 ˜ 50ms

; --- 10ms Delay ---
DELAY_10MS:
    MOVLW   D'40'        ; 40 loops
    MOVWF   COUNT1
DELAY_10MS_1:
    MOVLW   D'250'       ; 250 inner loops
    MOVWF   COUNT2
DELAY_10MS_2:
```

```
    NOP
    DECFSZ  COUNT2,F
    GOTO    DELAY_10MS_2
    DECFSZ  COUNT1,F
    GOTO    DELAY_10MS_1
    RETURN

; --- 5ms Delay ---
DELAY_5MS:
    MOVLW   D'20'
    MOVWF   COUNT1
DELAY_5MS_1:
    MOVLW   D'250'
    MOVWF   COUNT2
DELAY_5MS_2:
    NOP
    DECFSZ  COUNT2,F
    GOTO    DELAY_5MS_2
    DECFSZ  COUNT1,F
    GOTO    DELAY_5MS_1
    RETURN

; --- 2ms Delay ---
DELAY_2MS:
    MOVLW   D'8'
    MOVWF   COUNT1
DELAY_2MS_1:
```

```
    MOVLW   D'250'
    MOVWF   COUNT2
DELAY_2MS_2:
    NOP
    DECFSZ  COUNT2,F
    GOTO    DELAY_2MS_2
    DECFSZ  COUNT1,F
    GOTO    DELAY_2MS_1
    RETURN

; --- 1ms Delay ---
DELAY_1MS:
    MOVLW   D'4'
    MOVWF   COUNT1
DELAY_1MS_1:
    MOVLW   D'250'
    MOVWF   COUNT2
DELAY_1MS_2:
    NOP
    DECFSZ  COUNT2,F
    GOTO    DELAY_1MS_2
    DECFSZ  COUNT1,F
    GOTO    DELAY_1MS_1
    RETURN

; --- 100us Delay ---
DELAY_100US:
```

```
    MOVLW   D'100'
    MOVWF   COUNT1
DELAY_100US_LOOP:
    NOP
    DECFSZ  COUNT1,F
    GOTO    DELAY_100US_LOOP
    RETURN

; --- 20us Delay ---
Delay_20us:
    MOVLW   D'20'
    MOVWF   TEMP
Delay_Loop:
    DECFSZ  TEMP, F
    GOTO    Delay_Loop
     RETURN



     END
```