

Princess Sumaya University for Technology

King Abdullah II Faculty of Engineering

Electrical Engineering Department



Embedded Systems (22442)
Avoido – Obstacle Avoidance Robot

Author (1): Qais Jildeh [20210155]

Author (2): Malak Alshawish [20210944]

Author (3): Abdallah Al-Omari [20191041]

Supervisor: Prof. Belal Sababha

January 14, 2025

Abstract

This project aims to design and implement an autonomous obstacle avoidance robot using the PIC16F877A. Our project utilizes three ultrasonic sensors providing real time obstacle avoidance capabilities. The system integrates DC motors for movement, and it integrates analog input devices such as LDR for detecting light intensity, digital input/output such as LEDs and buzzer. In addition, the project integrates interrupt-based tasks using IR sensor and timer-based task such as CCP and delays to improve its' functionality. Avoido offers an efficient and streamlined solution to enhance robotic mobility using a combination of ultrasonic and IR sensors.

Table of Contents

Introduction.....	2
Objectives	2
Hardware Components.....	2
Design	4
Pin Connection.....	4
Software Design.....	5
Hardware Design.....	6
Procedure and Methods.....	7
Functions.....	7
Problems and Recommendations	12
Conclusion	12
References.....	12

Introduction

The Avido project involves developing an autonomous obstacle-avoiding robot using ultrasonic sensors for obstacle detection and an IR sensor for interrupt-based tasks. The robot will move around using DC motors, and the state is indicated by LEDs and buzzers. This system integrates interrupt-based and timer-based tasks to ensure real-time obstacle detection and response. This simplified solution provides an effective method for autonomous navigation that can be applied to warehouse automation and search and rescue scenarios.

Objectives

- Analog Input Utilization: Use at least one analog input device, such as an ultrasonic sensor, to detect obstacles and measure distances in the robot's environment.
- Motor Control with PWM: Implement CCP PWM-driven loads to control DC motors for smooth and precise movement, allowing the robot to navigate forward, reverse, and turn.
- Interrupt-Based Tasks: Develop at least one interrupt-based task using an IR sensor to enable real-time obstacle detection and immediate response.
- Timer-Based Tasks: Incorporate at least one timer-based task to manage periodic actions, such as delays.
- Sensor Integration: Integrate at least three sensors, including ultrasonic sensors for obstacle detection and an IR sensor for interrupt-based tasks.
- H-Bridge for Motor Control: Implement an H-Bridge to control the direction and speed of the DC motors, enabling forward, reverse, and rotation.
- Power Management: Include a power ON/OFF switch and a reset button for easy control and system reset in addition to voltage regulation.
- Prototype Functionality: Ensure the prototype works effectively, demonstrating reliable obstacle avoidance, smooth navigation, and real-time responsiveness.

Hardware Components

In our project, we used the following hardware components:

Ultrasonic Range Finder

The ultrasonic range finder is a sensor that measures distance by sending out ultrasonic waves and detecting their reflection. It works on the principle of time of flight and is commonly used for obstacle detection and distance measurement. The model we used is the HC-SR04, which has a transmitter and receiver pair to measure distances accurately.



LEDs (Light Emitting Diode)

LEDs are semiconductor devices that emit light when current passes through them. They are used as indicators in the project to signal various conditions, such as power status, obstacle detection, or process completion.



Buzzer

A buzzer is an audio signaling device that generates sound when activated. It can be either piezoelectric or electromechanical and is used in the project to provide audible alerts, such as warnings or notifications based on sensor inputs.



H-Bridge

An H-Bridge is a circuit that allows bidirectional control of DC motors using transistors or MOSFETs. It enables the motor to move forwards or backward by reversing the polarity of the voltage applied to it.



DC Motor

A DC motor converts electrical energy into mechanical motion. It is used for driving wheels or other mechanical parts in the project. The speed and direction of the motor are controlled using the H-Bridge circuit or a PWM (Pulse Width Modulation) signal.

Servo Motor

A servo motor is a rotary actuator that allows precise control of angular position. It consists of a DC motor, gearbox, and position control system.

LDR (Light Dependent Resistor)

An LDR is a light-sensitive resistor whose resistance changes based on the intensity of light falling on it.

IR Sensor (Infrared Sensor)

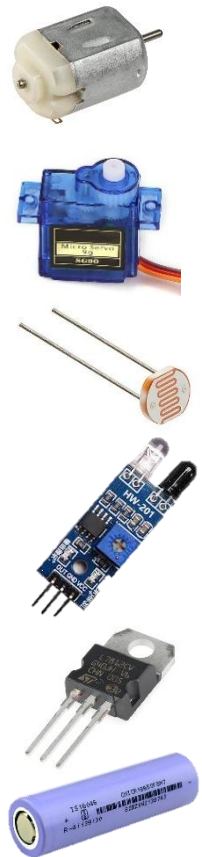
An IR sensor is used for object detection. It consists of an IR LED that emits infrared light and a photodiode that detects the reflected light.

Voltage Regulator

A voltage regulator is an electronic device that provides a stable voltage output despite fluctuations in input voltage or load conditions. It is essential for ensuring that components receive the correct voltage and do not get damaged.

Battery

The battery serves as the power source for the entire system.

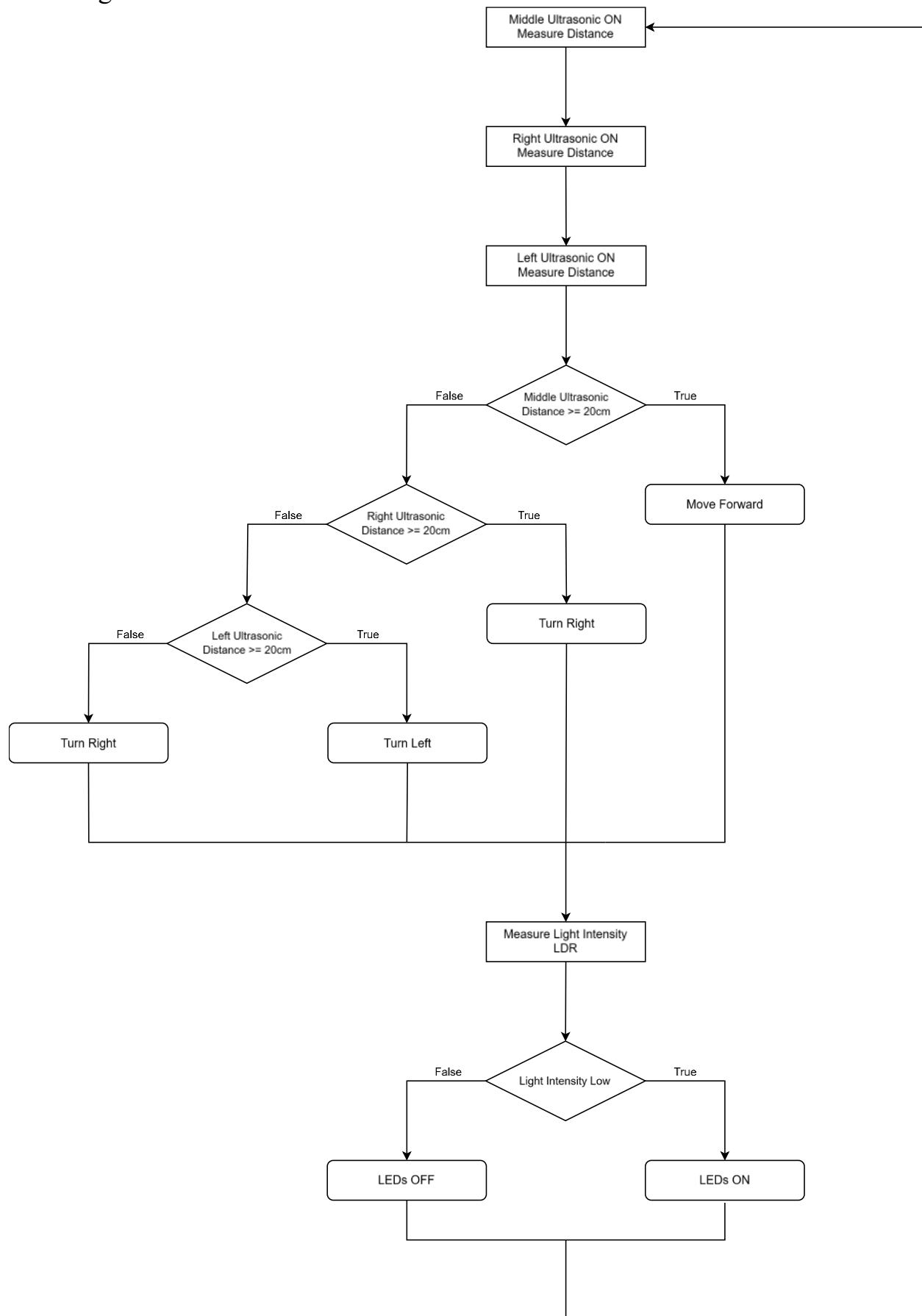


Design

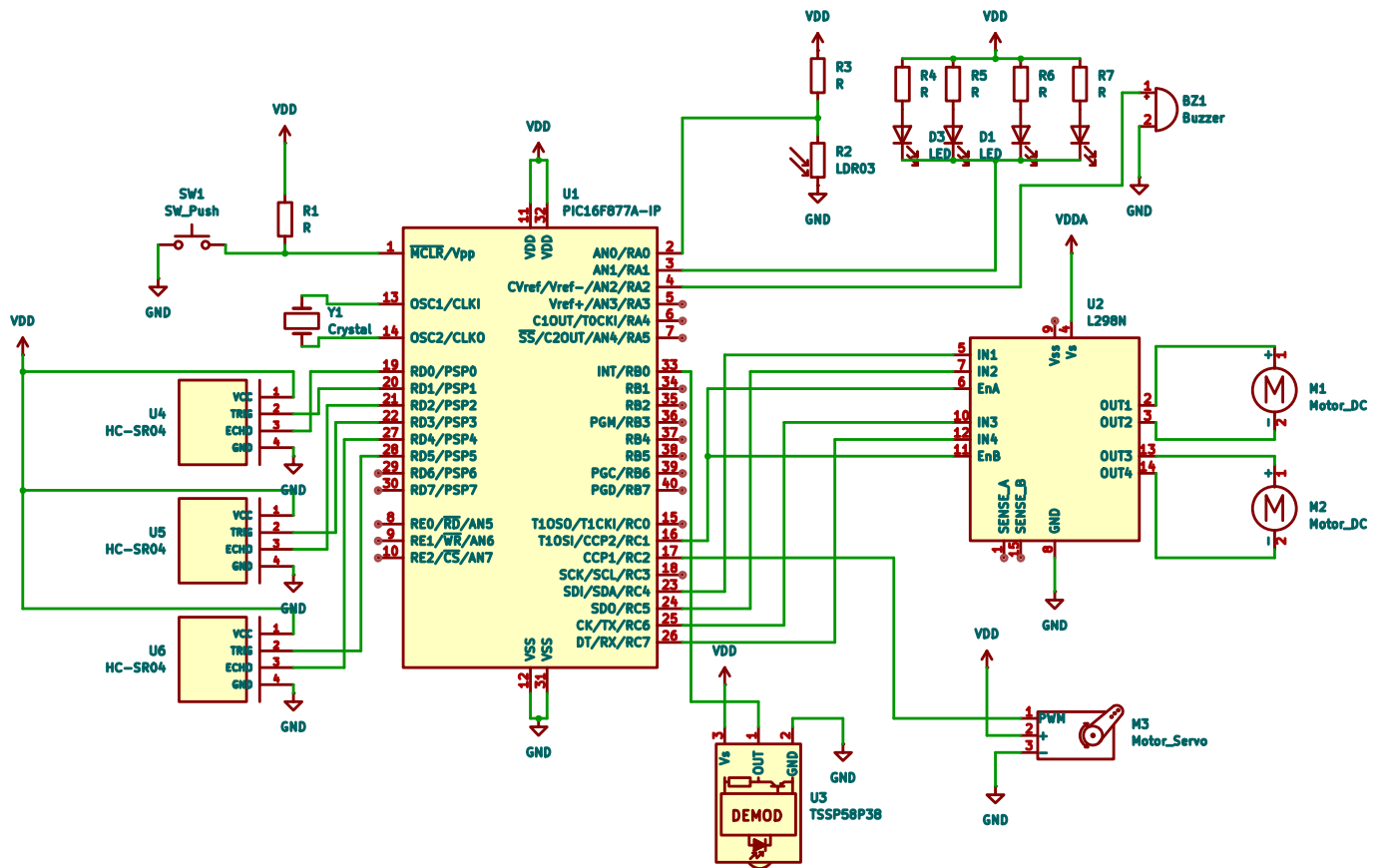
Pin Connection

PIN	Port	Use	Connection
1	\overline{MCLR}	Master Clear	Push Button
2	RA0	LDR Sensor	Analog Input
3	RA1	LEDs	Sinking Voltage (Digital)
4	RA2	Buzzer	Sourcing Voltage (Digital)
5	RA3	-	-
6	RA4	-	-
7	RA5	-	-
8	RE0	-	-
9	RE1	-	-
10	RE2	-	-
11	VDD	High Voltage Source	5V
12	VSS	Ground	0V
13	OSC1	Oscillator	8MHz External Oscillator
14	OSC2	Oscillator	8MHz External Oscillator
15	RC0	-	-
16	RC1	CCP PWM	H-Bridge EN1 & EN2
17	RC2	CCP Compare	Servo Motor
18	RC3	-	-
19	RD0	Ultrasonic 0	Echo Pin for Ultrasonic 0
20	RD1	Ultrasonic 0	Trigger Pin for Ultrasonic 0
21	RD2	Ultrasonic 1	Echo Pin for Ultrasonic 1
22	RD3	Ultrasonic 1	Trigger Pin for Ultrasonic 1
23	RC4	IN1	DC Motor 1
24	RC5	IN2	DC Motor 1
25	RC6	IN3	DC Motor 2
26	RC7	IN4	DC Motor 2
27	RD4	Ultrasonic 2	Echo Pin for Ultrasonic 2
28	RD5	Ultrasonic 2	Trigger Pin for Ultrasonic 2
29	RD6	-	-
30	RD7	-	-
31	VSS	High Voltage Source	5V
32	VDD	Ground	0V
33	RB0	External Interrupt	IR Sensor Output Pin
34	RB1	-	-
35	RB2	-	-
36	RB3	-	-
37	RB4	-	-
38	RB5	-	-
39	RB6	-	-
40	RB7	-	-

Software Design



Hardware Design



Procedure and Methods

Functions

calculate_distance(char sensorNumber)

The `calculate_distance` function is responsible for measuring the distance using an ultrasonic sensor. First, it determines the trigger and echo pins based on the `sensorNumber`, which identifies the specific sensor being used. The function then sends a trigger pulse to initiate the sensor. When the sensor detects an obstacle, it returns a response via the echo pin. The function starts and stops a timer based on the echo response. The measured time is then converted into a distance value in centimeters using the formula $distance = \frac{time}{58.82}$. This formula assumes that the speed of sound is 343 ms^{-1} .

```
1 // Ultrasonic sensor distance calculation function
2 unsigned int calculate_distance(char sensorNumber){
3     unsigned int time = 0, distance;
4     unsigned char triggerPin = 0, echoPin = 0; // Initialize to prevent undefined behavior
5
6     // Assign trigger and echo pins based on sensorNumber
7     if(sensorNumber == 0){
8         triggerPin = 0x02;
9         echoPin = 0x01;
10    }
11    else if(sensorNumber == 1){
12        triggerPin = 0x08;
13        echoPin = 0x04;
14    }
15    else{
16        triggerPin = 0x20;
17        echoPin = 0x10;
18    }
19
20    // Send Trigger Pulse
21    PORTD = PORTD | triggerPin; // Set trigger pin HIGH
22    Delay_us(10); // Wait for 10 microseconds
23    PORTD = PORTD & ~triggerPin; // Set trigger pin LOW
24
25    // Wait for Echo Start
26    while (!(PORTD & echoPin)); // Wait until echo pin goes HIGH
27
28    // Reset Timer Before Measurement
29    TMR1H = 0;
30    TMR1L = 0;
31
32    // Start Timer
33    T1CON = 0x01;
34
35    // Wait for Echo End
36    while (PORTD & echoPin); // Wait until echo pin goes LOW
37
38    // Stop Timer
39    T1CON = 0x00;
40
41    // Read Timer Value
42    time = (TMR1H << 8) | TMR1L;
43
44    // Convert Time to Distance (assuming speed of sound = 343 m/s)
45    distance = time / 58.82; // Distance in cm
46
47    return distance; // Return Distance
48 }
```


ADCinit()

The ADCinit() function initializes the Analog-to-Digital Converter (ADC) module. It configures the ADCON0 and ADCON1 registers to enable ADC functionality. This allows the microcontroller to read analog signals from sensors like the LDR or ultrasonic sensors.

```
1 void ADCinit(void){
2     ADCON0 = 0x41; // ATD ON, Don't GO, Channel 0, Fosc/16
3     ADCON1 = 0xCE; // All channels are Digital except RA0/AN0 is Analog, 500 KHz, right justified
4 }
```

ADCread()

The ADCread() function reads the analog input from the ADC module. It starts an ADC conversion by setting the GO bit and waits for the conversion to complete. Once done, it returns the result as a 10-bit value, which represents the intensity of the analog signal.

```
1 unsigned int ADCread(void){
2     ADCON0 = ADCON0 | 0x04; // GO
3     while(ADCON0 & 0x04);
4
5     return ((ADRESH << 8) | ADRESL);
6 }
7
```

initPWM()

The initPWM() function initializes the Pulse Width Modulation (PWM) for motor control. It configures Timer2 and the CCP2 module for PWM operation. The function sets the PWM frequency and duty cycle to ensure proper speed and torque control for the motors.

```
1 void initPWM(void){
2     PORTB = 0x00;
3     T2CON = 0x07; // Turn on Timer 2 + Prescaler 1:16
4     CCP2CON = 0x0C; // Two LSBs of duty cycle
5     PR2 = 0xFA; // ==> 0.5x10^-6 x 250 x 16 = 2ms
6     CCPR2L = 0x7D; // 50% PWM
7 }
```


goForward()

The goForward() function moves the robot forward by setting the appropriate control signals on PORTC. This ensures that both motors rotate in the forward direction, propelling the robot ahead.

```
1 // Move forward
2 void goForward(void){
3     PORTC = 0x90; // Enable left and right forward
4 }
```

goRight()


The `goRight()` function turns the robot to the right by adjusting PORTC. It enables the right wheel to move backward while the left wheel moves forward, causing a rightward turn.



```
1 // Keep turning right in position
2 void goRight(void){
3     PORTC = 0x50; // Enable right backward and left forward
4 }
```

goLeft()


The `goLeft()` function turns the robot to the left by adjusting PORTC. It enables the left wheel to move backward while the right wheel moves forward, causing a leftward turn.



```
1 // Keep turning left in position
2 void goLeft(void){
3     PORTC = 0xA0; // Enable left backward and right forward
4 }
```

servoINIT()

The `servoINIT()` function initializes the servo motor control by configuring Timer1 and the CCP1 module. It enables CCP1 interrupts for precise control of the servo motor, ensuring it can rotate to the desired angle based on the input signal.



```
1 void servoINIT(void){
2     TMR1H = 0x00;
3     TMR1L = 0x00;
4
5
6     HL = 1; // Start with high state
7     CCP1CON = 0x08; // Compare mode Set output on match
8
9
10    T1CON = 0x01; // Timer1 On Fosc/4 (inc 0.5µs) with no prescaler
11
12
13    PIE1 = PIE1 | 0x04; // Enable CCP1 interrupts
14
15
16    CCPR1H = 2000 >> 8; // Initial compare value for high state
17    CCPR1L = 2000;
18 }
```

interrupt()

The `interrupt()` function handles external and CCP1 interrupts. If an external interrupt occurs, it controls PORTC and PORTA to handle obstacle detection, allowing the robot to react accordingly. If a CCP1 interrupt occurs, it adjusts the servo motor pulse width to change its position. The function resets Timer1 and clears interrupt flags to ensure continuous operation without interference from past interrupts.

```
1  void interrupt(void){
2      if(INTCON & 0x02){ // Check for external interrupt
3          PORTC = PORTC & 0x0F;
4          while(!(PORTB & 0x01)){
5              PORTA = PORTA | 0x06;
6              //delayTMR1(1);
7              mydelay_ms(500);
8              PORTA = PORTA & 0xF9;
9              // //delayTMR1(1);
10             mydelay_ms(500);
11         }
12         PORTA = PORTA | 0x02;
13         PORTA = PORTA & 0x03;
14         INTCON = INTCON & 0xFD;
15     }
16
17     if (INTCON & 0x04) { // Timer0 overflow
18         Counter++;
19         INTCON = INTCON & 0xFB; // Clear the Timer0 overflow flag
20         TMR0 = 236; // Reset Timer0 for the next overflow
21     }
22     else if(PIR1 & 0x04) { // CCP1 interrupt
23         if(!HL){ // High state
24             CCPR1H = angle >> 8;
25             CCPR1L = angle;
26             HL = 0; // Switch to low state
27             CCP1CON = 0x09; // Compare mode, clear output on match
28         }
29         else{ // Low state
30             CCPR1H = (40000 - angle) >> 8; // Calculate low state duration
31             CCPR1L = (40000 - angle);
32             CCP1CON = 0x08; // Compare mode, set output on match
33             HL = 1; // Switch to high state
34         }
35         TMR1H = 0x00;
36         TMR1L = 0x00;
37         PIR1 = PIR1 & 0xFB; // Clear CCP1 interrupt flag
38     }
39     else {INTCON = 0xF0;} // Any other interrupts are not allowed
40 }
```

main()

The `main()` function is the main control loop for the robot. It initializes I/O ports, ADC, PWM, and servo components before entering a continuous loop. The loop continuously reads sensor values to determine the robot's movement direction. If an obstacle is detected, the function decides whether the robot should go forward, left, or right based on the sensor readings. Additionally, it implements night vision control by monitoring LDR values. If the light intensity is below a threshold, it turns on LEDs. The function also periodically adjusts the servo angles in a loop with delays to scan the surroundings.

```
1  void main(void){
2      TRISA = 0x01; // A0 only analog input, everything else is a digital output
3      TRISB = 0x01; // RB0 input for interrupt
4      TRISC = 0x00; // Port C all outputs, RC1 --> PWM for H-Bridge, RC2 --> Compare for Servo Motor
5      TRISD = 0x15; // D0 --> D5 for ultrasonic sensor
6      TRISE = 0x00; // Unimplemented
7
8      PORTA = PORTA | 0x02;
9      PORTA = PORTA & 0xFB;
10
11     OPTION_REG = 0x80;
12     INTCON = 0xD0;
13
14     ADCinit();
15     initPWM();
16     servoINIT();
17
18     while(1){
19         // Any value distance that is less than 10cm is considered to be too close, change direction
20         middleSensor = calculate_distance(0)/10;
21         rightSensor = calculate_distance(1)/10;
22         leftSensor = calculate_distance(2)/10;
23
24         // Movement code for H-Bridge
25         if(middleSensor >= 2) {goForward();}
26         else if(rightSensor >= 2) {goRight();}
27         else if(leftSensor >= 2) {goLeft();}
28         else{goRight();} // Keep turning right (counter clockwise in position) until front is clear
29
30         // Night Vision
31         lightIntensity = ADCread();
32         // ! Re-calibrate
33         if(lightIntensity > 818){ // 1KOhm, ~4.0V && previousState
34             //previousState = 0;
35             PORTA = PORTA & 0x01; // Turn on LEDs ==> 0
36             angle = 1000; // Set angle to 0 degrees
37         }
38         else{ // if(lightIntensity <= 818 && !previousState)
39             //previousState = 1;
40             PORTA = PORTA | 0x02; // Turn off LEDs ==> 1
41             angle = 3100; // Set angle to 0 degrees
42         }
43     }
44 }
45
```

Problems and Recommendations

- **Constant Brownouts on Startup:** The voltage regulator requires additional capacitors to stabilize the output signal and prevent voltage drops during startup. Adding appropriate capacitors will improve power stability and prevent brownout issues.
- **Challenges with Code Assembly:** Initially, assembling the entire code at once led to multiple errors, making the project non-functional. To resolve this, we restructured our approach by implementing and testing small modules incrementally, ensuring functionality at each stage before integration.
- **Difficulties with EasyPIC Board:** The EasyPIC development board presented several challenges, as it was often unclear whether issues were caused by incorrect board configurations, logical errors in the code, or faulty physical connections. A more systematic debugging approach and thorough documentation of board settings would help mitigate these challenges in future projects.
- **Microcontroller Overload Due to High Current Consumption:** The PIC16F877A struggled to handle multiple interface devices simultaneously due to excessive current consumption. Using external drivers or selecting a microcontroller with higher current handling capacity would improve performance and reliability.
- **Timer Limitations:** The inability to use the same timer for multiple functions caused timing conflicts within the system. Implementing additional timers or restructuring the timing logic using software-based delays or alternate timer configurations could help manage multiple timing-dependent tasks more effectively.

Conclusion

The Avido project has succeeded in establishing an independent robot that can detect and respond to obstacles using ultrasonic sensors in real time. Combining DC motors, LEDs, and buzzers, the robot achieves smooth navigation with obstacle avoidance. The project satisfies all the technical requirements of PWM control, interrupt-based tasks, and a standalone embedded system. Demonstrating teamwork and innovation, Avido is a vivid example of how the principles of embedded systems translate into practice. Future enhancements would include more sophisticated sensors and the addition of efficient pathfinding algorithms. Overall, the project is a working instance of autonomous robotics design.

References

<https://www.microchip.com/en-us/product/pic16f877a>

Designing Embedded Systems with PIC Microcontrollers Principles and Applications, 2nd edition, 2009, Tim Wilmshurst, ISBN: 978-1-85617-750-4