```c
// Justin Maeder & Zach Takkesh
// CECS 347
// Final Project -- Obstacle Avoiding Robot Vehicle


// 1. Pre-processor Directives Section
// Constant declarations to access port registers using
// symbolic names instead of addresses
//_________LIBRARIES_________
#include <stdint.h>
#include "Nokia5110.h"
#include "tm4c123gh6pm.h"
#include <math.h>
//________________________PWM INITIALIZATION_____________________________
#define SYSCTL_RCC_USEPWMDIV    0x00100000  // Enable PWM Clock Divisor
#define SYSCTL_RCC_PWMDIV_M     0x000E0000  // PWM Unit Clock Divisor
#define SYSCTL_RCC_PWMDIV_2     0x00000000  // /2
//____________GPIO LOCK___________
#define GPIO_LOCK_KEY           0x4C4F434B
//____Percentage values of 16000____
#define duty_30    4800
#define duty_40    6400
#define duty_60    9600
#define duty_80   12800
#define duty_98   15680
//____________DIRECTION_____________
#define direction      GPIO_PORTB_DATA_R
#define forward        0x03
#define backward         0x0C
#define r_f                    0x01     // RIGHT FORWARD
#define r_b                    0x04     // RIGHT BACKWARD
#define l_f                    0x02     // LEFT FORWARD
#define l_b                      0x08       // LEFT BACKWARD

// 2. Declarations Section
//   Global Variables
uint16_t adc_4, adc_5, adc_1, PE4_cal, PE5_cal, pot_cal, speed;

unsigned int time;

//   Function Prototypes
float  Distance_Cal(int adc_val);

void DisableInterrupts(void); // Disable interrupts
void EnableInterrupts(void);  // Enable interrupts

void ADC(void);
void PortB_Init(void); // PB0-3

short POT(int adc_1);
int update_speed(short percent);
void steering(int left_dist, int right_int, int speed);
int speedFromADC(int adc_val);

// PWM clock cycles output PB6,7
void PWM0A_Init(uint16_t period, uint16_t duty);
void PWM0B_Init(uint16_t period, uint16_t duty);
void PWM0A_Duty (uint16_t duty);
void PWM0B_Duty (uint16_t duty);

void SysTick_Init(unsigned long period);
void SysTick_Handler(void);

void WaitForInterrupt(void);



// 3. Subroutines Section
// MAIN: Mandatory for a C Program to be executable
int main(void){
    DisableInterrupts();
    Nokia5110_Init();
    ADC();
    SysTick_Init(16000);        //systick triggers every 8uS
    PortB_Init();
    PWM0A_Init(16000,0);
    PWM0B_Init(16000,0);

    //________FIRST ROW___________
    Nokia5110_Clear();
    Nokia5110_OutString("*ROBO__UNIT*");

    EnableInterrupts();
    PWM0A_Duty(duty_30);
    PWM0B_Duty(duty_30);
    direction = forward;
    while(1){
        PE4_cal  = Distance_Cal(adc_4);  // right
        PE5_cal  = Distance_Cal(adc_5);  // left
        pot_cal  = POT(adc_1);
        speed        = update_speed(pot_cal);
        steering(PE5_cal, PE4_cal, speed);

        //_________THIRD ROW___________
        Nokia5110_SetCursor(0,2);
        Nokia5110_OutString("LEFT~~~RIGHT");

        //________FOURTH ROW___________
        Nokia5110_SetCursor(0,3);
        Nokia5110_OutUDec(PE5_cal);
        Nokia5110_SetCursor(7,3);
        Nokia5110_OutUDec(PE4_cal);

        //_________FIFTH ROW___________
        Nokia5110_SetCursor(0,4);
        Nokia5110_OutString("  POT-PWM  ");

        Nokia5110_SetCursor(1,5);
        Nokia5110_OutUDec(pot_cal);
        Nokia5110_OutChar('%');
    } // End while
}

short POT(int adc_1){ // DISPLAY POT %
    short percent;
    unsigned int maxADC = 4095;
    percent = (adc_1 * 100) / maxADC;
    if(percent >= 98) percent = 98;
    return percent;
}

int update_speed(short percent){
    unsigned int duty;
    duty = 160 * percent;
    return duty;
}

void steering(int left_dist, int right_dist, int speed){
    direction = forward;
    if(left_dist < 10 & right_dist < 10){ // STOP CONDITION
        direction = backward;
        PWM0A_Duty(speed);      // backward
        PWM0B_Duty(speed);      // backward
    }

    else if(left_dist < 22){  // LEFT WALL - WARNING
        direction = l_f + r_b;
        PWM0A_Duty(speed);      // forward
        PWM0B_Duty(speed);      // backward
```

```
    }

    else if(right_dist < 22){ // RIGHT WALL - WARNING
        direction = l_b + r_f;
        PWM0A_Duty(speed);      // backward
        PWM0B_Duty(speed);      // forward
    }

    else if( (left_dist > 33) && (right_dist > 33) ){ // END OF TRACK
        PWM0A_Duty(0);          // STOP
        PWM0B_Duty(0);          // STOP
    }

    else{
        PWM0A_Duty(speed);      // forward
        PWM0B_Duty(speed);      // forward
    }
}

float Distance_Cal(int adc_val){ // Calculate distance
    float dist = 0.0;
    float volt = 0.0;
    volt = (0.0009 * adc_val) + 0.0000486;
    dist = 12.648 * pow(volt,-0.705);
    return dist;
}

void ADC(void){
    volatile unsigned long delay;
    SYSCTL_RCGCGPIO_R |= 0x10;                  //1. clock for port E
    while ((SYSCTL_RCGCGPIO_R&0x10) ==0 ){};    // BLOCKED

    GPIO_PORTE_DIR_R   &= ~0x32;                //2. PE1,PE4,PE5 inputs
    GPIO_PORTE_AFSEL_R |=  0x32;                //3. ENABLE alt func PE1,4,5
    GPIO_PORTE_DEN_R   &= ~0x32;                //4. DISABLE digital I/O PE1,4,5
    GPIO_PORTE_AMSEL_R |=  0x32;                //5. ENABLE analog functionality PE1,4,5

    SYSCTL_RCGCADC_R   |=  0x01;                //6. activate ADC0
    SYSCTL_RCGC0_R     |=  0x00010000;

    delay = SYSCTL_RCGCADC_R;                   // extra time to stabalize
    delay = SYSCTL_RCGCADC_R;                   // extra time to stabalize
    delay = SYSCTL_RCGCADC_R;                   // extra time to stabalize

    ADC0_PC_R     =  0x01;                      //7. 125k sample/s
    ADC0_SSPRI_R  =  0x3210;                    //8. seq 0 highest priority
    ADC0_ACTSS_R &= ~0x0004;                    //9. DISABLE seq 2
    ADC0_EMUX_R  &= ~0x0F00;                    //10. seq1 is software trigger
    ADC0_SSMUX2_R =  0x0892;                    //11. Ain8,9,2 (PE5,4,1)
    ADC0_SSCTL2_R =  0x0600;                    //12. IE0 END0 = ON || TS0, D0 = OFF

    ADC0_IM_R    &= ~0x0004;                    //13. DISABLE SS2 int
    ADC0_ACTSS_R |=  0x0004;                    //14. ENABLE SS2
}

void PortB_Init (void){
    volatile unsigned long delay;
    SYSCTL_RCGC2_R |= 0x00000002;       // Enable clock to PORTB
    delay = SYSCTL_RCGC2_R;               // Delay
    GPIO_PORTB_LOCK_R = GPIO_LOCK_KEY;  // Unlock PortB

    GPIO_PORTB_CR_R    |=  0x0F;        // Allow changes for PB0-3
    GPIO_PORTB_AMSEL_R &=  0x0F;        // Disable analog function for PB0-3
    GPIO_PORTB_DIR_R   |=  0x0F;    // Set PB0-3 output
    GPIO_PORTB_AFSEL_R &= ~0x0F;    // Disable alternate function for PB0-3
    GPIO_PORTB_PCTL_R  &= ~0x0F;    // GPIO clear bit PCTL
    GPIO_PORTB_PUR_R   &= ~0x0F;    // Disable pullup resistors for PB0-3
    GPIO_PORTB_DEN_R   |=  0x0F;        // Enable digital pins for PB0-3
}
```

```
void PWM0A_Init (uint16_t period, uint16_t duty){ // RIGHT WHEEL
    volatile unsigned long delay;
    SYSCTL_RCGCPWM_R  |= 0x01;           // 1) Enable the PWM clock
    SYSCTL_RCGCGPIO_R |= 0x02;           // 2) Enable the GPIO clock
    delay = SYSCTL_RCGCGPIO_R;

    GPIO_PORTB_AFSEL_R |=  0x40;         // 3) Enable PB6 alternate function
    GPIO_PORTB_PCTL_R  &= ~0x0F000000;   // 4) GPIO clear bit PCTL
    GPIO_PORTB_PCTL_R  |=  0x04000000;   //    Use Port B pin 6 PWM0

    GPIO_PORTB_AMSEL_R &= ~0x40;         // 5) Clear PB6 AMSEL
    GPIO_PORTB_DEN_R   |=  0x40;         // 6) Digital enable PB6

    SYSCTL_RCC_R = 0x00100000 | (SYSCTL_RCC_R & (~0x000E0000)); // 7) PWM divider to configure for /2 divider

    PWM0_0_CTL_R  = 0;                   // 8) Reload down-counting mode
    PWM0_0_GENA_R = 0xC8;                // 9) LOAD low, CMPA high

    PWM0_0_LOAD_R  = period - 1;
    PWM0_0_CMPA_R  = duty - 1;
    PWM0_0_CTL_R  |= 0x00000001;         // 10) start PWM0
    PWM0_ENABLE_R |= 0x00000001;         // 11) ENABLE PB6, M0PWM0
}

void PWM0B_Init (uint16_t period, uint16_t duty){ // LEFT WHEEL
    volatile unsigned long delay;
    SYSCTL_RCGCPWM_R  |= 0x01;            // 1) Enable the PWM clock
    SYSCTL_RCGCGPIO_R |= 0x02;            // 2) Enable the GPIO clock
    delay = SYSCTL_RCGCGPIO_R;

    GPIO_PORTB_AFSEL_R |=  0x80;          // 3) Enable PB7 alternate function
    GPIO_PORTB_PCTL_R  &= ~0xF0000000;    // 4) GPIO clear bit PCTL
    GPIO_PORTB_PCTL_R  |=  0x40000000;    //    Use Port B pin 7 PWM0
    GPIO_PORTB_AMSEL_R &= ~0x80;          // 5) Clear PB7 AMSEL
    GPIO_PORTB_DEN_R   |=  0x80;          // 6) Digital enable PB7

    SYSCTL_RCC_R |=  SYSCTL_RCC_USEPWMDIV;// 7) PWM divider to configure for /2 divider
    SYSCTL_RCC_R &= ~SYSCTL_RCC_PWMDIV_M;
    SYSCTL_RCC_R +=  SYSCTL_RCC_PWMDIV_2;

    PWM0_0_CTL_R  = 0;                    // 8) Reload down-counting mode
    PWM0_0_GENB_R = 0xC08;                // 9) LOAD low, CMPA high

    PWM0_0_LOAD_R  = period - 1;
    PWM0_0_CMPB_R  = duty - 1;
    PWM0_0_CTL_R  |= 0x00000001;          // 10) start PWM0
    PWM0_ENABLE_R |= 0x00000002;          // 11) ENABLE PB7, M0PWM1
}

void PWM0A_Duty (uint16_t duty){
    PWM0_0_CMPA_R = duty-1;     // 1) count value when output rises
}

void PWM0B_Duty (uint16_t duty){
    PWM0_0_CMPB_R = duty-1;     // 1) count value when output rises
}

// Initialize SysTick with busy wait running at bus clock - 125k/s
void SysTick_Init(unsigned long period){
  NVIC_ST_CTRL_R = 0;         // disable SysTick during setup
  NVIC_ST_RELOAD_R = period-1;// reload value
  NVIC_ST_CURRENT_R = 0;      // any write to current clears it
  NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R&0x00FFFFFF)|0x40000000; // priority 2
                              // enable SysTick with core clock and interrupts
  NVIC_ST_CTRL_R |= 0x07;
}
```

```

void SysTick_Handler(void){
    time = time + 1;
    ADC0_PSSI_R = 0x0004;                         //1. initiate SS2
    adc_1       = ADC0_SSFIFO2_R & 0xFFF; //2. read PE1 (POT)
    adc_4       = ADC0_SSFIFO2_R & 0xFFF; //2. read PE4 (RIGHT IR sensor)
    adc_5       = ADC0_SSFIFO2_R & 0xFFF; //2. read PE5 (LEFT IR sensor)
    ADC0_ISC_R  = 0x0004;                         //3. clear flag
}

```