```
#include "stm32f4xx.h"

// Define pins
#define TRIG_PIN (1 << 8)   // PA8 - Trigger Pin
#define ECHO_PIN (1 << 9)   // PA9 - Echo Pin
#define LED_PIN (1 << 5) // PA5 (PWM for LED)
#define DISTANCE_THRESHOLD 5 // Distance threshold in cm
#define LDR_CHANNEL 0    // ADC Channel 0 (LDR connected to PA0)
#define TIMEOUT_LIMIT 10000   // Timeout limit for echo signal in cycles

// Function prototypes
void GPIO_Init(void);
void Timer_Init(void);
void ADC_Init(void);
uint16_t read_LDR_value(void);
float measure_distance(void);
void delay_us(uint32_t delay);

int main(void) {
   GPIO_Init();
   Timer_Init();
   ADC_Init();

   while (1) {
      // Measure the distance
      float distance = measure_distance();

      // Check if the distance is below the threshold
      if (distance > 0 && distance <= DISTANCE_THRESHOLD) {
         // Read LDR value
         uint16_t ldrValue = read_LDR_value();
```

```
                                        if (ldrValue <= 300){
            // Map the LDR value (0-4095) to a PWM duty cycle (0-100%)
            uint16_t pwmValue = 100 - (ldrValue * 100) / 4095;
            // Set PWM duty cycle
            TIM2->CCR1 = (pwmValue * TIM2->ARR) / 100; // Scale to timer period
                                        }
        } else {
            // Distance exceeds the threshold, turn off PWM and LED
            TIM2->CCR1 = 0;  // Set PWM duty cycle to 0
        }
    }
}

void GPIO_Init(void) {
    // Enable GPIOA clock
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;

    // Configure PA8 (TRIG_PIN) as output
    GPIOA->MODER |= (1 << 16);     // Set PA8 as output
    GPIOA->OTYPER &= ~TRIG_PIN;    // Set PA8 as push-pull
    GPIOA->OSPEEDR |= (3 << 16);   // Set PA8 as high speed

    // Configure PA9 (ECHO_PIN) as input
    GPIOA->MODER &= ~(3 << 18);    // Set PA9 as input

    // Configure PA5 (LED_PIN) as alternate function (PWM output)
    GPIOA->MODER |= GPIO_MODER_MODER5_1; // Alternate function mode
    GPIOA->AFR[0] |= (1 << 20);         // AF1 for TIM2_CH1

    // Configure PA0 (LDR input) as analog input
```

```
    GPIOA->MODER |= GPIO_MODER_MODER0;   // Analog mode
}

void Timer_Init(void) {
    // Enable TIM2 clock
    RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;

    // Configure TIM2 for PWM
    TIM2->PSC = 84 - 1;        // Prescaler: 1 MHz (1 µs per tick)
    TIM2->ARR = 1000 - 1;      // Auto-reload: 1000 steps (1 kHz PWM frequency)
    TIM2->CCMR1 |= (6 << 4);    // PWM mode 1 on channel 1
    TIM2->CCER |= TIM_CCER_CC1E; // Enable channel 1 output
    TIM2->CR1 |= TIM_CR1_CEN;   // Enable the timer
}

void ADC_Init(void) {
    // Enable ADC1 clock
    RCC->APB2ENR |= RCC_APB2ENR_ADC1EN;

    // Configure ADC1 for single-channel continuous conversion
    ADC1->SQR3 = LDR_CHANNEL;  // Select channel 0 (PA0)
    ADC1->CR2 |= ADC_CR2_ADON; // Enable ADC1
    ADC1->CR2 |= ADC_CR2_CONT; // Continuous conversion mode
    ADC1->CR2 |= ADC_CR2_SWSTART; // Start conversion
}

uint16_t read_LDR_value(void) {
    while (!(ADC1->SR & ADC_SR_EOC)); // Wait for conversion to complete
    return ADC1->DR;               // Return ADC value
}
```

```
// Delay function in microseconds
void delay_us(uint32_t delay) {
    TIM2->CNT = 0;                  // Reset the timer count
    while (TIM2->CNT < delay);      // Wait until the timer reaches the delay
}

// Function to measure distance using the ultrasonic sensor
float measure_distance(void) {
    uint32_t start_time, end_time;

    // Send a 10 µs pulse on the trigger pin
    GPIOA->ODR &= ~TRIG_PIN;    // Trigger low
    delay_us(2);
    GPIOA->ODR |= TRIG_PIN;     // Trigger high
    delay_us(10);
    GPIOA->ODR &= ~TRIG_PIN;    // Trigger low

    // Wait for echo pin to go high (start of the pulse)
    uint32_t timeout = TIMEOUT_LIMIT;
    while (!(GPIOA->IDR & ECHO_PIN) && timeout--) {
        // Timeout protection
    }
    if (timeout == 0) return -1;  // Timeout, return error code

    start_time = TIM2->CNT;   // Record the start time

    // Wait for echo pin to go low (end of the pulse)
    timeout = TIMEOUT_LIMIT;
    while (GPIOA->IDR & ECHO_PIN && timeout--) {
        // Timeout protection
    }
```

```
    if (timeout == 0) return -1;  // Timeout, return error code

    end_time = TIM2->CNT;     // Record the end time

    // Calculate the duration and convert to distance
    uint32_t duration = end_time - start_time;
    if (duration == 0) return -1;  // Avoid divide by zero

    // Calculate the distance based on the duration
    float distance = (duration * 0.034) / 2.0; // Speed of sound = 0.034 cm/µs

    return distance;
}
```