PROJECT REPORT
ON

STM32-Based Smart Adaptive Headlight Control System

SUBMITTED IN PARTIAL FULFILLMENT
OF
THE REQUIREMENTS OF
THE AWARD OF THE

PG-DIPLOMA IN EMBEDDED SYSTEM DESIGN

OFFERED BY

C-DAC HYDERABAD

BY

Y.CHAITANYA	250850330079
Y.AMSHULA	250850330080
YOGESH BISHT	250850330081

ADVANCED COMPUTING TRAINING SCHOOL C-DAC HYDERABAD-500005

August 2025

CERTIFICATE

This is to certify that this is a Bonafide record of project entitled “STM32-Based Smart Adaptive Headlight Control System”. Y.Chaitanya(250850330079) and Y.Amshula(250850330080) and Yogesh Bisht(250850330081) have completed project work as part of Diploma in Embedded System Design (August,2025-Batch), a PG course offered by C-DAC Hyderabad. They have completed project work under the supervision of Mr. Ashwin. Their Performance was found to be good.

Signature of Project guide

(Mr. Ashwin)

DATE: 06-02-2026

PLACE: C-DAC, HARDWARE PARK, HYDERABAD.

ACKNOWLEDGMENT

STM32-Based Smart Adaptive Headlight Control System project has been presented. This project marks the final hurdle we have tackled and represents one of the many challenges we have undertaken and are yet to undertake. However, we could not have made it without the support and guidance from the following. Firstly, I express my sincere and heartfelt gratitude to Mr. Ashwin. who has guided me in completing the project with his cooperation, valuable guidance and immense help in giving the project a shape and success. I am very much indebted to him for suggesting a challenging and interactive project and his valuable advice at every stage of this work.

PG-DESD February 2025

Y.CHAITANYA	250850330079
Y.AMSHULA	250850330080
YOGESH BISHT	250850330081

TABLE OF CONTENTS

ABSTRACT

The STM32-Based Smart Adaptive Headlight Control System is designed to enhance road safety by automatically adjusting vehicle headlight intensity based on environmental and road conditions. Traditional headlight systems require manual control, which may cause glare to oncoming vehicles or insufficient visibility during adverse conditions. This project aims to overcome these limitations through automation.

The system uses Light Dependent Resistors (LDRs) to detect ambient light levels and the presence of oncoming vehicle headlights. An MQ-2 smoke sensor is used to detect smoke and MH-Water Sensor is used for fog and rain conditions. These sensors continuously monitor real-time conditions around the vehicle.

An STM32 microcontroller serves as the core processing unit of the system. The analog outputs from the sensors are converted into digital values using the built-in Analog-to-Digital Converter (ADC) of the STM32. Based on the processed sensor data, the controller generates Pulse Width Modulation (PWM) signals to regulate the brightness of the LED headlights.

By automatically increasing headlight intensity during low-visibility conditions and reducing brightness when oncoming vehicles are detected, the system improves driver visibility while minimizing glare for others. The proposed system is cost-effective, reliable, and efficient, making it suitable for real-time automotive lighting applications and future smart vehicle systems.

1. INTRODUCTION

1.1 Problem Statement:

Conventional vehicle headlight systems operate with fixed or manually controlled brightness, which can lead to poor visibility during night driving, rain, fog, or smoke conditions. High-beam headlights may cause glare to oncoming drivers, increasing the risk of accidents, while low-beam headlights may not provide sufficient illumination in low-visibility environments.

To address these challenges, this project implements a STM32-Based Smart Adaptive Headlight Control System an automatic and intelligent headlight control system that can adapt its brightness based on real-time environmental conditions such as day,night,ambient light, oncoming vehicle headlights, smoke, fog, and rain. The absence of such adaptive systems in conventional vehicles highlights the importance of developing a smart, sensor-based solution to improve road safety and driving comfort.

To address these issues by designing an STM32-based Smart Adaptive Headlight Control System that automatically adjusts headlight intensity using inputs from LDRs, an MQ-2 sensor, and an MH-Water sensor.

1.2 Software & Hardware Requirements:

1.2.1 Hardware Specification:

- STM32F411RE Nucleo Board
- Light Dependent Resistors (LDRs)
- MQ-2 Smoke Sensor
- MH-Water (Rain/Fog) Sensor
- LED-Headlight Module
- Power Supply
- Jumper Wires & Breadboard

1.2.2 Software Specification:

- STM32CubeIDE

1.3 Coding and Testing:

- C programming language

2. CASE STUDY

2.1 Case Scenario: Smart Adaptive Headlight Control Using STM32 .

This case study focuses on the design and implementation of a Smart Adaptive Headlight Control System using the STM32 microcontroller. The STM32 acts as the central controller, continuously monitoring sensor inputs and making real-time decisions to adjust headlight brightness. Its built-in ADC and PWM capabilities enable efficient processing of sensor data and smooth control of the headlight intensity under varying driving conditions.

- Automatic headlight control
- Improved visibility during night, rain, fog, and smoke
- Reduced glare for oncoming vehicles
- Low power consumption
- Cost-effective and reliable system

Disadvantages

- Sensor performance may vary with environmental conditions
- Requires proper calibration
- Limited accuracy compared to camera-based systems

2.2 Case Scenario: Sensor-Based Adaptive Headlight Operation.

The system operates using inputs from multiple sensors such as LDRs for day and night detection, an MQ-2 sensor for smoke, and an MH-Water sensor for rain and fog. Based on the sensor readings, the headlight brightness is automatically increased or decreased to ensure better visibility and reduced glare. This sensor-based approach enhances driving safety by adapting the headlight operation to real-time environmental conditions without manual intervention.

- Centralized control using a powerful microcontroller
- Accurate decision-making using ADC and PWM features
- Flexible system design and easy firmware updates
- Supports integration of multiple sensors
- Real-time processing with fast response

Disadvatages

- Higher cost compared to simple analog systems
- Requires programming knowledge
- System complexity increases with more features

processing of sensor data and smooth control of the headlight intensity under varying driving conditions.

- Automatic headlight control
- Improved visibility during night, rain, fog, and smoke
- Reduced glare for oncoming vehicles
- Low power consumption
- Cost-effective and reliable system

Disadvantages

- Sensor performance may vary with environmental conditions
- Requires proper calibration
- Limited accuracy compared to camera-based systems

2.3 Case Scenario: Sensor-Based Adaptive Headlight Operation.

The system operates using inputs from multiple sensors such as LDRs for day and night detection, an MQ-2 sensor for smoke, and an MH-Water sensor for rain and fog. Based on the sensor readings, the headlight brightness is automatically increased or decreased to ensure better visibility and reduced glare. This sensor-based approach enhances driving safety by adapting the headlight operation to real-time environmental conditions without manual intervention.

- Centralized control using a powerful microcontroller
- Accurate decision-making using ADC and PWM features
- Flexible system design and easy firmware updates
- Supports integration of multiple sensors
- Real-time processing with fast response

Disadvatages

- Higher cost compared to simple analog systems
- Requires programming knowledge
- System complexity increases with more features

3 IMPLEMENTATION

The **Smart Adaptive Headlight Control System** is implemented using an **STM32 microcontroller** as the main control unit. The system uses **two LDR sensors** to detect ambient light conditions and oncoming vehicle headlights. An **MQ-2 smoke sensor** is used to sense smoke or low-visibility conditions, while a **moisture sensor** detects rain or wet conditions.

All sensor outputs are analog in nature and are converted into digital values using the **ADC of the STM32**. Based on these sensor inputs, the microcontroller generates **PWM signals** to control the brightness of the LED headlights. The headlight intensity is automatically increased during low-visibility conditions and reduced when oncoming vehicle light is detected, ensuring better visibility and reduced glare.

3.1 STM32F411RE NUCLEO BOARD:

The **STM32F411RE Nucleo Board** is a development platform from STMicroelectronics, featuring the **STM32F411RE microcontroller** based on the high-performance ARM Cortex-M4 core running at 100 MHz's It offers 512 KB of Flash memory and 128 KB of SRAM. The board is **Arduino Uno R3** and **ST morpho** pin compatible, enabling easy prototyping with shields and expansion boards. It includes an **ST-LINK/V2-1 debugger/programmer**, so no external programmer is needed. Integrated peripherals like I²C, SPI, UART, ADC, and timers make it suitable for IoT, embedded systems, and sensor interfacing projects. Its affordability and extensive STM32CubeIDE support make it ideal for beginners and professionals.

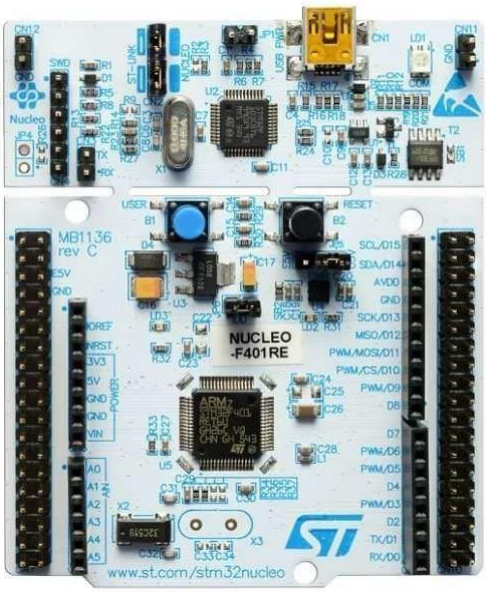


Fig 3.1.1: STM32F411RE Nucleo Board

Specifications of STM32 Board:

The STM32F411RE Nucleo board is a development platform designed by STMicroelectronics to evaluate and prototype applications based on the STM32F4 series microcontrollers. It features the STM32F411RE MCU, which is based on the ARM Cortex-M4 core with FPU, running at a maximum frequency of 100 MHz, delivering high performance and efficiency for complex embedded applications. It includes 512 KB of Flash memory and 128 KB of SRAM, providing ample storage and runtime memory for code and data.

For connectivity, the board offers a wide range of interfaces including USART, SPI, I2C, ADC, DAC, PWM, and GPIO pins, allowing integration with various sensors, displays, and communication modules. It also supports USB OTG functionality. The Nucleo board includes an ST-LINK/V2-1 debugger/programmer embedded on the board, eliminating the need for an external programmer, and enabling drag-and-drop programming via a USB connection.

The board operates on 3.3V logic but supports 5V-tolerant I/O pins, making it compatible with both 3.3V and 5V peripherals. It has two power supply options: USB or an external source via VIN pin. The Arduino Uno R3 and ST morpho headers make the board highly versatile, supporting both Arduino shields and ST expansion boards.

With its compact size, low power consumption, and rich set of peripherals, the STM32F411RE Nucleo board is ideal for projects in IoT, robotics, industrial control, data acquisition, and real-time applications. It is also fully compatible with STM32CubeIDE and the STM32 HAL libraries, allowing easy code development and debugging for beginners and professionals alike.

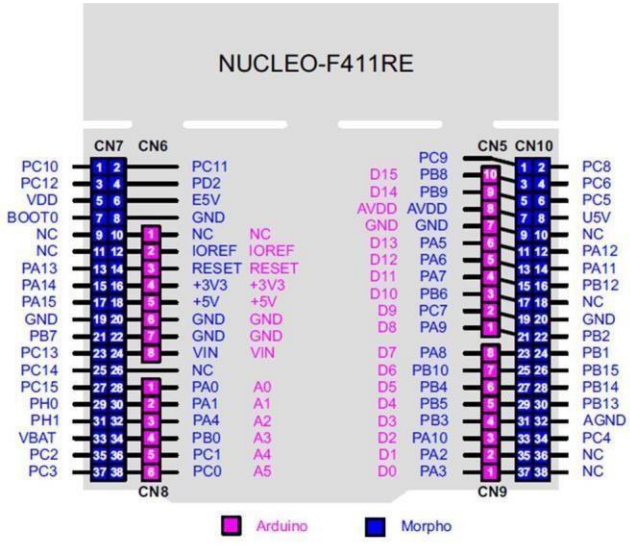


Fig 3.1.2: Pinout diagram of STM32F411RE

3.2 Light Dependent Resistors (LDRs):

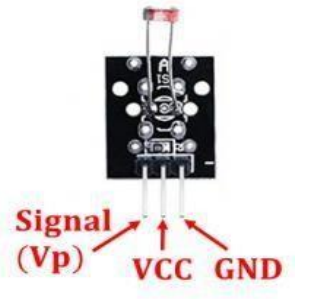
A **Light Dependent Resistor (LDR)**, also known as a **photoresistor**, is a light-sensitive passive component whose resistance changes according to the intensity of light falling on it. The LDR is typically made of **cadmium sulfide (CdS)**, which exhibits photoconductivity. When light photons strike the material, more charge carriers are generated, causing the resistance to decrease.

In dark or low-light conditions, the resistance of the LDR is very high, whereas in bright light, its resistance drops significantly. Due to this property, LDRs are widely

used in automatic lighting and light-sensing applications. In this project, the LDR is connected in a **voltage divider circuit**, and the output voltage varies with light intensity.

In the **Smart Adaptive Headlight Control System**, LDRs are used to detect **ambient light conditions** and **oncoming vehicle headlights**. The analog voltage from the LDR circuit is fed to the **ADC of the STM32 microcontroller**, which processes the data and adjusts the headlight brightness using **PWM control**. This helps in reducing glare and improving night-time driving safety.

LDRs are chosen for this project because they are **cost-effective**, **easy to interface**, **low-power**, and suitable for real-time light intensity detection in automotive applications



z

Fig 3.2.1: Light Dependent Resistors (LDRs)

Specifications of LDR

- **Type:** Cadmium Sulfide (CdS) Photoresistor
- **Operating Voltage:** Depends on circuit (typically 3.3 V / 5 V)
- **Dark Resistance:** ~1 MΩ or higher

- **Light Resistance:** ~1 kΩ to 10 kΩ (in bright light)
- **Spectral Peak:** ~540 nm (visible light range)
- **Response Time:** Moderate (milliseconds range)
- **Operating Temperature:** −30 °C to +70 °C
- **Power Consumption:** Very low

3.3 MQ-2 Smoke Sensor

The **MQ-2 smoke sensor** is a gas and smoke sensing module used to detect the presence of smoke and combustible gases in the surrounding environment. It operates based on the principle of **change in resistance** of a sensitive material when exposed to smoke or gases such as LPG, methane, hydrogen, and carbon monoxide.

In the **Smart Adaptive Headlight Control System**, the MQ-2 sensor is used to detect **smoke or low-visibility conditions** on the road. The sensor provides an **analog output**, which varies according to the concentration of smoke present. This output is connected to the **ADC of the STM32 microcontroller**, where it is converted into digital data for processing.

When smoke is detected, the STM32 automatically increases the headlight brightness using **PWM control**, improving visibility for the driver during smoky or fog-like conditions. The MQ-2 sensor is chosen due to its **high sensitivity**, **low cost**, and **easy interfacing** with microcontrollers.

Fig 3.3.1: MQ-2 smoke sensor

Specifications of MQ-2 Smoke Sensor

- **Type:** Smoke and Combustible Gas Sensor
- **Detectable Gases:** Smoke, LPG, Methane, Hydrogen, CO
- **Operating Voltage:** 5 V
- **Output Type:** Analog (and Digital in some modules)
- **Sensitivity:** Adjustable (using onboard potentiometer)

- **Preheat Time:** ~20–30 seconds
- **Operating Temperature:** −10 °C to +50 °C
- **Response Time:** Fast
- **Power Consumption:** Moderate

3.4 Moisture / Rain Sensor (MH-Water Sensor)

The **Moisture / Rain Sensor**, commonly known as the **MH-Water Sensor**, is used to detect the presence of **rain, water droplets, or moisture** in the surrounding environment. It operates on the principle of **change in electrical conductivity** when water comes in contact with the sensor surface. As moisture increases, the conductivity of the sensor increases, resulting in a change in output voltage.

In the **Smart Adaptive Headlight Control System**, the MH-Water sensor is used to detect **rain and fog-like conditions**. The sensor provides an **analog output**, which is connected to the **ADC of the STM32 microcontroller**. When rain or moisture is detected, the controller automatically increases the headlight brightness to improve visibility and ensure safer driving conditions.

The MH-Water sensor is chosen for this project due to its **simple design**, **low cost**, **fast response**, and **easy interfacing** with microcontrollers, making it suitable for real-time automotive safety applications.

Fig 3.4.1 : Moisture / Rain Sensor (MH-Water Sensor)

Specifications of Moisture / Rain Sensor:

- **Sensor Type:** MH-Water / Rain Sensor
- **Operating Voltage:** 3.3 V – 5 V
- **Output Type:** Analog
- **Detection Method:** Conductivity based

- **Response Time:** Fast
- **Sensitivity:** High (depends on moisture level)
- **Operating Temperature:** −10 °C to +70 °C
- **Power Consumption:** Low
- hen GPS fix is obtained

3.5 LED Headlight Module

The **LED Headlight Module** is used as the primary light source in the **Smart Adaptive Headlight Control System**. LEDs (Light Emitting Diodes) are preferred over conventional bulbs due to their **high efficiency, fast response, low power consumption, and long lifespan**.

In this project, the LED headlight brightness is controlled by the **STM32 microcontroller** using **PWM (Pulse Width Modulation)**. Based on sensor inputs such as **LDR (day/night and glare detection)**, **MQ-2 (smoke detection)**, and **MH-Water sensor (rain/fog detection)**, the STM32 dynamically adjusts the PWM duty cycle to increase or decrease the LED brightness.

During night-time, rain, fog, or smoke conditions, the LED intensity is increased to improve visibility. When sufficient ambient light is detected or glare conditions exist, the brightness is reduced to avoid discomfort to oncoming drivers. This adaptive control improves **road safety and driving comfort**.

Fig 3.4.3. LED Headlight Module

Specifications of LED Headlight Module

- **Type:** High-brightness LED
- **Operating Voltage:** 5 V – 12 V
- **Control Method:** PWM
- **Light Color:** White

- **Power Consumption:** Low
- **Response Time:** Very fast
- **Efficiency:** High
- **Lifespan:** Long (compared to halogen lamps)

3.6 STM32CubeIDE (Integrated Development Environment)

STM32CubeIDE is a comprehensive and unified Integrated Development Environment provided by STMicroelectronics for designing, developing, and debugging applications based on STM32 microcontrollers. It combines multiple development tools such as a C/C++ editor, project manager, build system, compiler, debugger, and the STM32CubeMX graphical configuration utility into a single software platform. This integration simplifies the embedded software development workflow and reduces development time.

STM32CubeIDE offers a user-friendly and structured environment that supports the complete application lifecycle—from peripheral configuration and code generation to compilation, flashing, and debugging. The IDE is based on the Eclipse framework, making it highly reliable and extensible for embedded system development.

Peripheral Configuration and Code Generation

The built-in STM32CubeMX tool allows developers to graphically configure the selected STM32 microcontroller. Users can easily select and configure:

- System and peripheral clocks
- GPIO pin modes and alternate functions
- ADC channels for sensor data acquisition
- Timers and PWM for brightness control
- Communication interfaces such as UART, SPI, and I2C
- Interrupts and DMA settings

Once the configuration is completed, STM32CubeIDE automatically generates HAL (Hardware Abstraction Layer)-based initialization code, ensuring correct peripheral setup and reducing the chances of configuration errors. This auto-generated code provides a strong foundation for application development while allowing developers to add custom logic.

Supported Boards and Microcontrollers

STM32CubeIDE supports a wide range of STM32 microcontroller families and evaluation boards, including:

- STM32F series (STM32F411RE, STM32F407, etc.)
- STM32F1, F3, F7, L, and H series
- STM32 Nucleo Boards
- STM32 Discovery Boards

Selecting a specific board or microcontroller automatically configures device-specific parameters such as CPU clock frequency, flash memory size, RAM size, and peripheral availability, ensuring compatibility and correct operation

Libraries, Drivers, and Middleware Support

STM32CubeIDE includes STM32 HAL (Hardware Abstraction Layer) and LL (Low-Layer) drivers, which provide standardized and portable APIs for controlling microcontroller peripherals. HAL drivers simplify application development, while LL drivers offer fine-grained control for performance-critical applications.

Additional middleware such as sensor drivers, communication protocols, and custom libraries can be integrated into the project as required, making STM32CubeIDE flexible for complex embedded systems.

Debug Console and Monitoring

STM32CubeIDE supports UART-based serial communication for real-time system monitoring. Sensor values, ADC readings, system states, and debugging messages can be transmitted to a serial terminal for observation during runtime. The IDE also provides debug watch windows and a console for analyzing system behavior during execution.

4. APPLICATION CODE

4.1 STM32 CODE:

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
* @file           : main.c
* @brief          : Main program body
  ******************************************************************************
* @attention
  *
* Copyright (c) 2026 STMicroelectronics.
* All rights reserved.
  *
* This software is licensed under terms that can be found in the LICENSE file    * in the root
   directory of this software component.
* If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/ #include
"main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <stdint.h>
#include "tcs34725.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
typedef enum
```

```
{
   STATE_IDLE = 0,
   STATE_DAY,
   STATE_DARK,
        STATE_RAINY_FOG_SMOKE,
   STATE_CLOUDY,
        STATE_STREET_LIGHT
} system_state_t;

typedef enum
{
   BEAM_NONE = 0,
   LOW_BEAM,
   HIGH_BEAM,
        BOTH_BEAM
} beam_t;

typedef enum{                    //enum for True and False
        FALSE=0,
        TRUE
}boolean;

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define ADC_CHANNEL_COUNT     4              //Number of ADC Used

//Indexes for DMA
#define FRONT_LDR_INDEX       0          //Front LDR index
#define TOP_LDR_INDEX         1          //top LDR index
#define SMOKE_INDEX           2          //smoke sensor index
#define MOISTURE_INDEX        3          //moisture sensor index

#define PWM_MAX               99       //MAX Brightness of LED
```

```
#define LOW_BEAM_PWM        59 //Low Beam Brightness
#define PWM_50                  39    //50% Brightness of LED #define PWM_90
  79    //90 Brightness of LED

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1;
DMA_HandleTypeDef hdma_adc1;

I2C_HandleTypeDef hi2c1;

TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim4;

/* USER CODE BEGIN PV */

/* ADC */
uint16_t adc_dma_buffer[ADC_CHANNEL_COUNT]; //ADC array for storing via DMA
boolean adc_data_ready = FALSE;
boolean RGB_ready=FALSE;

/* Sensor values */
uint16_t front_ldr;                 //Variable for storing front LDR value
uint16_t top_ldr;                   //Variable for storing top LDR value
uint16_t smoke;                         //Smoke value variable
uint16_t moisture;                  //Moisture value variable
uint16_t CCT;                           //Correlated Color Temperature
uint16_t lux;                       //Lumination of light
```

```
/* State machine */
system_state_t system_state = STATE_IDLE;       //Default State and variable for current state
/* PWM */
beam_t active_beam = BEAM_NONE;                 //current active beam state uint16_t
target_pwm = 0;             //PWM for Setting


/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void); static void MX_GPIO_Init(void);
static void MX_DMA_Init(void); static void MX_ADC1_Init(void);
static void MX_I2C1_Init(void); static void MX_TIM1_Init(void);
static void MX_TIM3_Init(void); static void MX_TIM4_Init(void);
/* USER CODE BEGIN PFP */

void time_start(void);
void set_beam_pwm(beam_t, uint16_t );
void  process_sensors(void);  void
update_state_machine(void);  void
active_beam_selection(void); void
apply_pwm(void);


/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

//PWM  starting  void
time_start(void){
        /*                                ====  Formulae for Time in PWM  ====
  *
```

```
        *
*                   T = (ARR+1) * (PSC+1) / F
        *
*                   where, T   = Time we need PWM for
*                   ARR = Auto Reload Register for TImer in PWM
*                   PSC = Pre-scaler set for Timer to get required Time
*                   F   = Frequency of the Clock of STM
        *
*                   --> So here we will be configuring our LED brightness in 10 micro-seconds
*                   --> ADC Value sensing every 5ms
*                   --> RGB Value every 10ms for it to have adequate time for sensing values and
                    converting      * */
  HAL_TIM_PWM_Start(&htim1,                                TIM_CHANNEL_1);
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);    htim3.Instance->CCR1=9;
            //5ms Interrupt    htim4.Instance->CCR1=19;      //10ms       interrupt
HAL_TIM_OC_Start_IT(&htim3, TIM_CHANNEL_1);
  HAL_TIM_OC_Start_IT(&htim4,TIM_CHANNEL_1);
}


//DMA starting Function
void  HAL_TIM_OC_DelayElapsedCallback(TIM_HandleTypeDef  *htim){
if (htim->Instance == TIM3)
            HAL_ADC_Start_DMA(&hadc1,  (uint32_t*)adc_dma_buffer,  4);
if (htim->Instance == TIM4)
            RGB_ready=TRUE;
}


//ADC Conversion Complete
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc){
      front_ldr   =   adc_dma_buffer[FRONT_LDR_INDEX];
top_ldr  = adc_dma_buffer[TOP_LDR_INDEX];       smoke
=  adc_dma_buffer[SMOKE_INDEX];            moisture   =
adc_dma_buffer[MOISTURE_INDEX];       adc_data_ready
= TRUE;
```

```
}

//Configuring PWM of LED
void set_beam_pwm(beam_t beam, uint16_t pwm){
if (beam == HIGH_BEAM)
   {
      __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, pwm);
      __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, 0);
   }
   else if (beam == LOW_BEAM)
   {
      __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, 0);
      __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, LOW_BEAM_PWM);
   }
   else if(beam == BOTH_BEAM){
        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, pwm);
        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, pwm);
   }
else    {
      __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, 0);
      __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, 0);
   }
}

//Data Processing Logic void
process_sensors(void)
{
   if (RGB_ready){                        //if RGB data ready
        uint16_t                  red,green,blue,clear;
getRGB(&red, &green, &blue,&clear);
        CCT=calculateColorTemperature(red,green,blue,clear);
        lux=calculateLux(red,green,blue);
        RGB_ready=FALSE;
   }
```

```
   if (!adc_data_ready)    //if  ADC  data  ready  in  DMA  or  Not
return;
   adc_data_ready = FALSE;
   //HAVE TO CONFIGURE FOR PROCESSING CLOUDNESS WITH RAIN AND FOG AND LED
BRIGHTNESS

   //LIGHT SOURCE
   if (top_ldr<=3000){

        if(lux >60 && lux<200 && CCT<=100 && CCT>=0) {
                system_state = STATE_DAY;
       }

        //else if(lux < 100 && lux >= 60 && CCT >= 4000 && CCT <= 8000) {      //testing
else if(lux < 60 && lux >= 30 && CCT<=12000 && CCT>=8000){ //rainy          if
(moisture >=3000)
                        system_state    =    STATE_RAINY_FOG_SMOKE;
        else
                        system_state = STATE_CLOUDY;
       }
        //else if(lux < 100 && lux >= 65 && CCT >= 4000 && CCT <= 8000) { //testing
else if(lux >= 100 && lux <= 2000 && CCT<=12000 && CCT>=8000) { //Fog
if (moisture>=1500)
                        system_state = STATE_RAINY_FOG_SMOKE;
                }
        else if(lux < 100 && lux >= 50 && CCT >= 4000 && CCT <= 8000) {       //testing
//else if(lux >= 200 && lux <= 1000 && CCT >= 7000 && CCT <= 9000) {//Smoke
if (smoke >=2000)
                        system_state    =    STATE_RAINY_FOG_SMOKE;
        else
                        system_state = STATE_STREET_LIGHT;
       }

        else if(lux >= 10 && lux <= 100 && CCT >= 1800 && CCT <= 6000) {
                        system_state = STATE_STREET_LIGHT;
       }
```

```
        else if(lux >= 1 && lux <= 10 && CCT >= 3000 && CCT <= 5000) {
                system_state = STATE_DARK;
         }
   }

   //DARK ENVIRONMENT
   else{
        system_state = STATE_DARK;
   }
}

//Updating States of Current Environment void
update_state_machine(void){
   switch (system_state)
   {
      case              STATE_DAY:
target_pwm  = PWM_50;
         break;

      case              STATE_CLOUDY:
target_pwm  = PWM_90;
         break;

      case STATE_DARK:

      case STATE_RAINY_FOG_SMOKE:
       target_pwm  = PWM_MAX;
       break;

      case STATE_STREET_LIGHT:
       target_pwm  = PWM_50;
       break;

      default:
target_pwm  = 0;
```

```
        break;
    }
}

//Active Beam Selection void
active_beam_selection(void){
switch(system_state){

            case STATE_DAY:
active_beam=LOW_BEAM;
                        break;

            case STATE_RAINY_FOG_SMOKE:
active_beam=BOTH_BEAM;
                    break;

            default:
                    active_beam=(front_ldr<2000)?LOW_BEAM:HIGH_BEAM;
                    break;
        }

}

//applying PWM to Circuit void
apply_pwm(void)
{
        active_beam_selection();
   set_beam_pwm(active_beam, target_pwm);
}


/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
```

```
  */            int
main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */   HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_ADC1_Init();
  MX_I2C1_Init();
  MX_TIM1_Init();
  MX_TIM3_Init();
  MX_TIM4_Init();      /*  USER  CODE  BEGIN  2  */
time_start();                  //Time start when Powered On
  /* USER CODE END 2 */
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */  while
(1)
  {
```

```
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
process_sensors();
update_state_machine();
apply_pwm();
  }
  /* USER CODE END 3 */
}

/**
* @brief System Clock Configuration
* @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /** Initializes the RCC Oscillators according to the specified parameters   *
in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON; RCC_OscInitStruct.PLL.PLLSource =
  RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 4;
```

```
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
* @brief ADC1 Initialization Function
* @param None
* @retval None
  */
static void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */

  /* USER CODE END ADC1_Init 0 */
  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */

  /* USER CODE END ADC1_Init 1 */
```

```
  /** Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of conversion)
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
hadc1.Init.Resolution = ADC_RESOLUTION_12B;
hadc1.Init.ScanConvMode = ENABLE;   hadc1.Init.ContinuousConvMode
= DISABLE;   hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
hadc1.Init.NbrOfConversion = 4;
hadc1.Init.DMAContinuousRequests = DISABLE;
hadc1.Init.EOCSelection = ADC_EOC_SEQ_CONV;      if
(HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.
  */
  sConfig.Channel = ADC_CHANNEL_0;          //front LDR sensor channel
sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;    if
(HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.
  */
  sConfig.Channel = ADC_CHANNEL_1;          //top LDR sensor channel
sConfig.Rank = 2;
```

```
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.
  */
  sConfig.Channel = ADC_CHANNEL_8;              //smoke          sensor          channel
sConfig.Rank = 3;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.
  */
  sConfig.Channel = ADC_CHANNEL_11;  //moisture sensor channel   sConfig.Rank = 4;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_Init 2 */

}
/**
* @brief I2C1 Initialization Function
* @param None
* @retval None
  */
static void MX_I2C1_Init(void)
{
```

```
  /* USER CODE BEGIN I2C1_Init 0 */

  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */

  /* USER CODE END I2C1_Init 1 */
hi2c1.Instance = I2C1;  hi2c1.Init.ClockSpeed =
400000;                 hi2c1.Init.DutyCycle      =
I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode                =                I2C_DUALADDRESS_DISABLE;
hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode                =                 I2C_GENERALCALL_DISABLE;
hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C1_Init 2 */

  /* USER CODE END I2C1_Init 2 */

}

/**
  * @brief TIM1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM1_Init(void)
{

  /* USER CODE BEGIN TIM1_Init 0 */
```

```
  /* USER CODE END TIM1_Init 0 */

  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};
  TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};

  /* USER CODE BEGIN TIM1_Init 1 */

  /* USER CODE END TIM1_Init 1 */


  htim1.Instance = TIM1;
  htim1.Init.Prescaler = 84;
  htim1.Init.CounterMode              =              TIM_COUNTERMODE_UP;
htim1.Init.Period = 99;
  htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim1.Init.RepetitionCounter = 0;
  htim1.Init.AutoReloadPreload   =   TIM_AUTORELOAD_PRELOAD_ENABLE;
if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode   =   TIM_MASTERSLAVEMODE_DISABLE;        if
(HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity                =                TIM_OCPOLARITY_HIGH;
sConfigOC.OCNPolarity               =               TIM_OCNPOLARITY_HIGH;
sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;   sConfigOC.OCIdleState
=      TIM_OCIDLESTATE_RESET;               sConfigOC.OCNIdleState      =
TIM_OCNIDLESTATE_RESET;
  if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
```

```
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_2) != HAL_OK)
  {
    Error_Handler();
  }
  sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;  sBreakDeadTimeConfig.DeadTime
= 0;
  sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
sBreakDeadTimeConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;  if
(HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM1_Init 2 */

  /* USER CODE END TIM1_Init 2 */
  HAL_TIM_MspPostInit(&htim1);

}

/**
  * @brief TIM3 Initialization Function
  * @param None
  * @retval None
  */ static void
MX_TIM3_Init(void)
{

  /* USER CODE BEGIN TIM3_Init 0 */

  /* USER CODE END TIM3_Init 0 */
```

```
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};

  /* USER CODE BEGIN TIM3_Init 1 */

  /* USER CODE END TIM3_Init 1 */
  htim3.Instance = TIM3;
  htim3.Init.Prescaler = 41999;
  htim3.Init.CounterMode               =               TIM_COUNTERMODE_UP;
htim3.Init.Period = 9;
  htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim3.Init.AutoReloadPreload   =   TIM_AUTORELOAD_PRELOAD_ENABLE;
if (HAL_TIM_OC_Init(&htim3) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode   =   TIM_MASTERSLAVEMODE_DISABLE;        if
(HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_TIMING;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH; sConfigOC.OCFastMode
  =    TIM_OCFAST_DISABLE;    if    (HAL_TIM_OC_ConfigChannel(&htim3,
  &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_OC_ConfigChannel(&htim3, &sConfigOC, TIM_CHANNEL_2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM3_Init 2 */
```

```
  /* USER CODE END TIM3_Init 2 */

}

/**
* @brief TIM4 Initialization Function
* @param None
* @retval None
  */
static void MX_TIM4_Init(void)
{

  /* USER CODE BEGIN TIM4_Init 0 */

  /* USER CODE END TIM4_Init 0 */

  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};

  /* USER CODE BEGIN TIM4_Init 1 */

  /* USER CODE END TIM4_Init 1 */
  htim4.Instance = TIM4;
  htim4.Init.Prescaler = 41999;
  htim4.Init.CounterMode = TIM_COUNTERMODE_UP; htim4.Init.Period
  = 19;
  htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE; if
  (HAL_TIM_OC_Init(&htim4) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode   =   TIM_MASTERSLAVEMODE_DISABLE;        if
(HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK)
```

```
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_TIMING;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_OC_ConfigChannel(&htim4, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM4_Init 2 */

  /* USER CODE END TIM4_Init 2 */

}

/**
  * Enable DMA controller clock
  */
static void MX_DMA_Init(void)
{

  /* DMA controller clock enable */
  __HAL_RCC_DMA2_CLK_ENABLE();

  /* DMA interrupt init */
  /* DMA2_Stream0_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA2_Stream0_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA2_Stream0_IRQn);

}

/**
* @brief GPIO Initialization Function
* @param None
```

```
* @retval None
  */
static void MX_GPIO_Init(void)
{
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
* @brief  This function is executed in case of error occurrence.
* @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
```

```
/**
* @brief  Reports the name of the source file and the source line number   *       where the assert_param
   error has occurred.
* @param  file: pointer to the source file name
* @param  line: assert_param error line source number
* @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT *

     //END
```

5. SCHEMATIC DIAGRAM

a. Block diagram

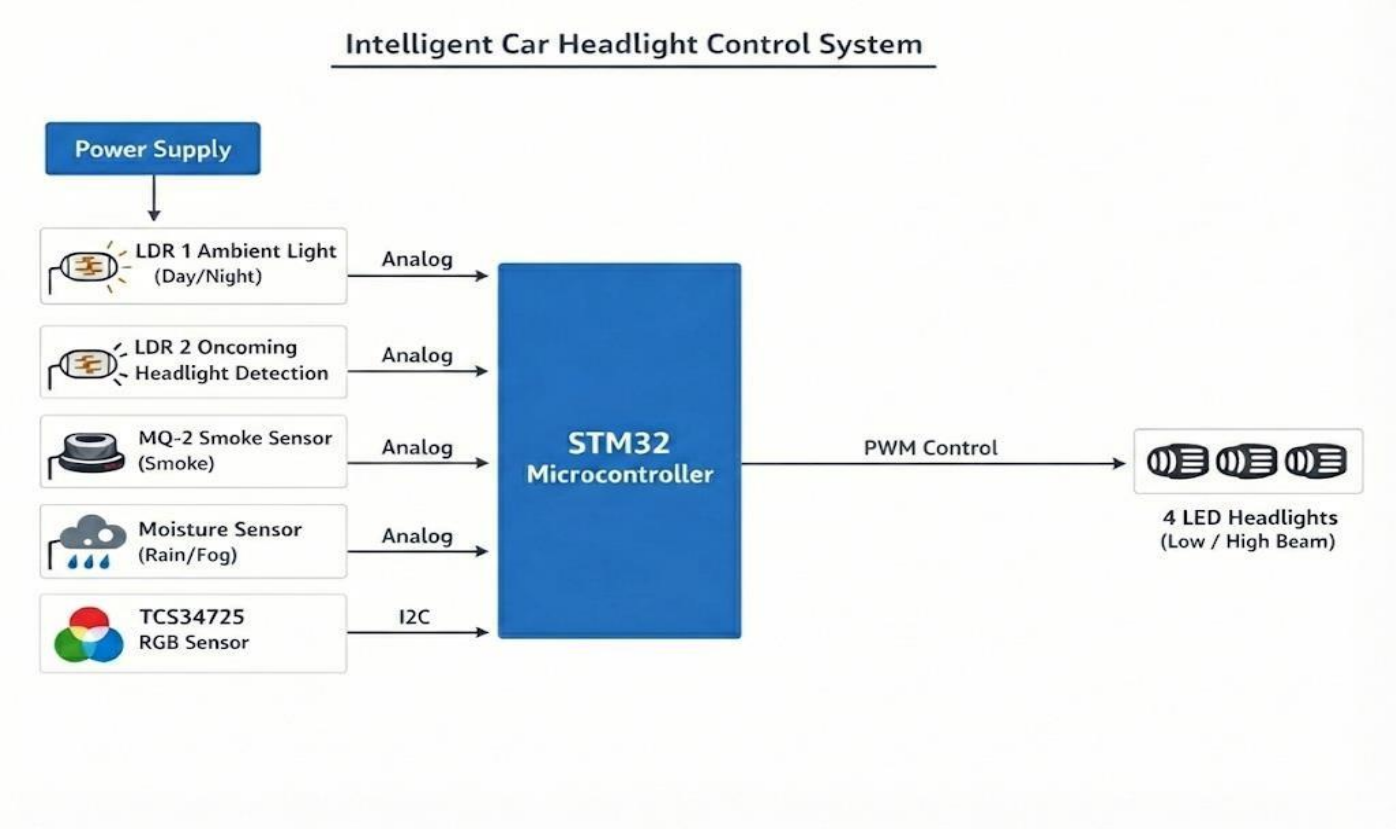


Figure 5.1. Block Diagram

Implemented System:

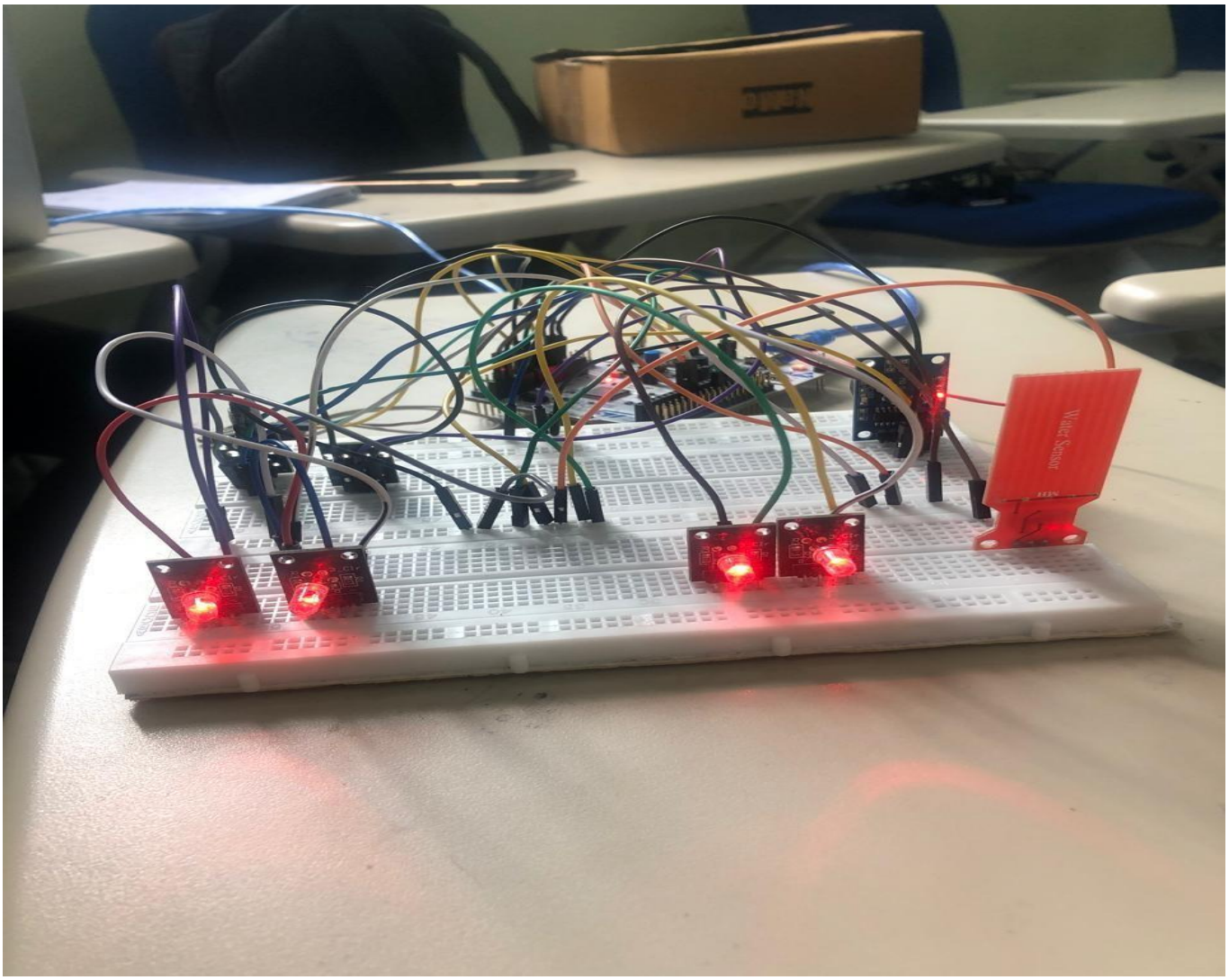


Fig 5.2: Implemented System of Geo-Guard

6. Flow of the Project Work

1. System Startup
   - STM32F411RE Nucleo board powers up.
   - GPIO, ADC, Timers, and PWM peripherals are initialized using STM32CubeIDE.
   - Sensors (LDR, MQ-2 Smoke Sensor, MH-Water Rain/Fog Sensor) are initialized.
   - LED headlight module is set to default brightness level.

2. Data Acquisition (Sensor Inputs)

i. LDR Sensor

· Measures ambient light intensity to detect day or night conditions.

· Provides analog voltage to STM32 ADC.

ii. MH-Water Sensor

· Detects rain or fog presence on the road.

· Outputs analog/digital signal to STM32.

iii. MQ-2 Smoke Sensor

· Detects smoke or low-visibility conditions.

· Sends analog signal to STM32 ADC.

3. **3. Sensor Data Processing (STM32)**

- **STM32 continuously reads sensor values using ADC channels.**
- **Threshold values are compared for:**
  - **Day / Night condition**
  - **Rain / Fog detection o Smoke detection**
- **Based on sensor combination, visibility level is determined.**

4. **Decision Making & Control Logic**

- **STM32 executes control logic:**
  - **Night detected → increase headlight brightness**
  - **Rain / Fog detected → moderate brightness**
  - **Smoke detected → optimized brightness to reduce glare**
  - **Daytime → minimum brightness or headlight OFF**
- **Priority logic ensures safe operation in combined conditions.**

5. **PWM-Based Brightness Control**

- **STM32 generates PWM signal using internal timers.**
- **PWM duty cycle is adjusted according to visibility level.**
- **Higher duty cycle → higher LED brightness**
- **Lower duty cycle → reduced brightness**

6. **Headlight Output Operation**

- PWM signal drives the LED headlight module.
- Smooth and gradual brightness variation is achieved.
- Prevents sudden glare and improves driving comfort.

7. **Continuous Monitoring & Update**

- **System continuously monitors sensor inputs in real time.**
- **Headlight brightness dynamically adapts to changing conditions.**
- **Ensures improved visibility, safety, and energy efficiency.**

7. LITERATURE SURVEY

The proposed Smart Adaptive Headlight Control System focuses on improving vehicle safety and visibility by automatically adjusting headlight intensity based on environmental and lighting conditions. It utilizes STM32 microcontrollers, light and environmental sensors, and PWM-based control techniques to adapt headlight brightness during day, night, rain, fog, and smoke conditions. The following literature survey highlights existing research and systems related to adaptive headlight control and sensor-based automotive safety systems.

S. No.	Title/Authors	Technology/Method	Application/Remarks	Limitations/Challenges
1	Automatic Headlight Control System	LDR sensor, Microcontroller, Relay/PWM	Adjusts headlight ON/OFF based on ambient light (day/night detection)	Limited to light intensity only, no weather adaptation
2	Sensor-Based Adaptive Headlight System	LDR, Rain sensor, Embedded controller	Improves visibility during night and rainy conditions	Sensor accuracy affected by dust and environmental noise

3	Intelligent Vehicle Lighting System	PWM control, LED headlights, Embedded systems	Smooth brightness control and energy efficiency	Requires proper calibration of PWM and sensors
4	Weather-Based Headlight Control System	Rain/Fog sensors, Microcontroller	Enhances safety during fog and rain by adjusting brightness	Does not consider smoke or multi-sensor priority

8. CONCLUSION

The STM32-Based Smart Adaptive Headlight Control System successfully demonstrates an intelligent approach to improving vehicle safety and driving comfort. The system automatically adjusts headlight brightness based on environmental conditions such as day and night lighting, rain, fog, and smoke. By eliminating the need for manual headlight control, the project reduces driver workload and enhances overall road safety.

The use of the STM32 microcontroller plays a key role in achieving accurate and real-time control. Features such as ADC for sensor data acquisition and PWM for LED brightness control enable smooth and efficient headlight operation. Sensors like LDR, MQ-2 smoke sensor, and MH-Water (rain/fog) sensor provide reliable input data, allowing the system to respond effectively to changing surroundings.

This project proves that a sensor-based adaptive headlight system can be implemented using low-cost components while still delivering reliable performance. Compared to conventional headlight systems, the proposed design offers better visibility, reduced glare, and improved energy efficiency. The modular design also allows easy expansion by adding more sensors or advanced control techniques.

In conclusion, the Smart Adaptive Headlight Control System is a practical and scalable solution for modern vehicles. It highlights the effective integration of embedded systems, sensors, and control algorithms using STM32. With further enhancements such as camera-based detection or

vehicle-to-vehicle communication, the system can be extended to meet future automotive safety requirements.

9. REFERENCES

1. P. K. Mane & S. R. Deshpande (2025): “Automatic Vehicle Headlights by Light Conditions Using STM32” — Focuses on energy-efficient intensity control and the interface between LDR sensors and STM32 GPIOs.
2. Li Zhang & Wei Chen (2025): “Design of Lightweight, High Precision, Multi Scene Module Based on STM32” — Discusses high-precision data acquisition and system architecture for adaptive front-lighting.
3. M. Zakaria & H. A. Rahman (2024): “Adaptive Headlight Control and Real-Time Pedestrian Detection” — Details the integration of sensor fusion algorithms on ARM Cortex-M based controllers.
4. S. Kumar & R. Singh (2022): “Smart Vehicle Headlight Auto Switching and Intensity Control” — A technical study on MOSFET-based dimming logic and ignitioncontrolled power management
    1. I. Chisalita and N. Shahmehri, *"A Peer-to-Peer Approach to Vehicular Communication for the Support of Traffic Safety Applications,"* IEEE Intelligent Transportation Systems, 2002.
    2. C. Campolo, A. Molinaro, R. Scopigno, *"Vehicular ad hoc Networks: Standards, Solutions, and Research,"* Springer, 2015.
    3. M. Boban, T. T. Vinhoza, M. Ferreira, J. Barros, and O. K. Tonguz, *"Impact of Vehicles as Obstacles in Vehicular Ad Hoc Networks,"* IEEE Journal on Selected Areas in Communications, vol. 29, no. 1, pp. 15–28, Jan. 2011.
    4. A. Bazzi, B. M. Masini, and A. Zanella, *"Performance Analysis of V2V Beaconing Using LTE in Direct Mode with Full Duplex Radios,"* IEEE Wireless Communications, vol. 24, no. 6, pp. 38–43, Dec. 2017.
    5. K. Gupta & A. V. Deshpande (2025): “Smart Vehicle Headlights Control System” — Researches the biological impact of glare (Troxler effect) and the corresponding software logic for auto-dimming.
    6. R. V. Prasad & J. S. Murthy (2023): “Design and Fabrication of Adaptive Headlight System with Fog and Slope Detection” — Explores using humidity sensors and gravity Pedestrian Detection” — Details the integration of sensor fusion algorithms on ARM Cortex-M based controllers.

7. S. Kumar & R. Singh (2022): “Smart Vehicle Headlight Auto Switching and Intensity Control” — A technical sensors to adjust the light beam on inclines.

8. J. D. Rathod & V. P. Patel (2018): “Design and Implementation of Intelligent Adaptive Front-lighting System” — Focuses on digital PWM signal processing for horizontal light swiveling.
9. M. T. H. Beg & N. Ahmad (2021): “Adaptive Headlight Control System” — Provides a methodology for mapping steering wheel angles to servo motor positions.
10. T. S. Rathore & S. Bhargava (2015): “Adaptive Headlight System for Automobiles” — A foundational paper for report writing regarding the mechanical pivoting of lamp assemblies.
11. A. S. Pawar & R. B. Patil (2016): “Headlight Intensity Control Methods - A Review” — Compares fuzzy logic vs. simple thresholding for smart headlight response.