A
Project Report
on

"Surveillance Robot using ESP32 Cam Module"

Submitted By

Sr No	PRN	Name
1	21410038	Mr. Shardul Shastri
2	21410039	Mr. Atharva Tambade
3	21410043	Mr. Ayush Shirbhate
4	21410050	Mr. Vikas Gouda

Under the guidance of

Ms. Deepika Chavan

Department of Electronics Engineering
WALCHAND COLLEGE OF ENGINEERING, SANGLI
(Government-Aided Autonomous Institute)

2023-2024

(This page is intentionally kept blank)

Contents

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

A surveillance robot using ESP32-CAM is a system that utilizes the ESP32-CAM board and a robot chassis to create a mobile surveillance device. The ESP32-CAM is a low-cost development board that integrates a small camera module and Wi-Fi connectivity. The robot chassis allows the device to move around and capture video in different locations. The system can be controlled through a web interface hosted on the ESP32-CAM board. The web interface allows the user to control the robot's movement, view live video streams, and take snapshots of the video feed. Additionally, the system can be programmed to detect motion using computer vision algorithms, such as object detection and tracking, and send alerts to the user. The surveillance robot using ESP32-CAM has potential applications in home security, monitoring of remote locations, and industrial surveillance. With its low cost and easy-to-use interface, it provides a convenient solution for anyone who needs to monitor their surroundings remotely.

ACKNOWLEDGEMENT

We would like to express our gratitude to our mentor, Ms. Deepika Chavan Mam, for her invaluable guidance and support throughout the development of our surveillance car robot project using the ESP32 CAM module. We also thank the Electronics Department for providing resources and assistance. Special thanks to faculty, staff, teammates, and the ESP32 CAM module for their contributions.

CHAPTER 1

1. Introduction

Our primary objective is to develop a mobile surveillance solution capable of autonomous navigation, obstacle detection, video capture, and real-time data transmission via IoT connectivity. By integrating these components, we aim to enhance security measures while providing users with convenient remote monitoring capabilities.

In today's dynamic security landscape, the need for autonomous surveillance systems is paramount. Our project endeavors to address this need by designing a surveillance robot car utilizing components such as the Arduino UNO, L298N Motor Driver, and incorporating IoT connectivity for remote control and data transmission

1.1. Background

In response to evolving security challenges, our project aims to pioneer a cutting-edge mobile surveillance solution. Leveraging advanced technologies including Arduino UNO, we're developing an autonomous robot capable of navigating, detecting obstacles, and capturing audio/video data. With IoT integration, our system enables real-time remote monitoring and data transmission, bolstering security measures while enhancing user convenience.

1.2. Motivation

Following are reasons due to which we find this project important:

- **Innovation:** Developing a hand gesture control robot is a cutting-edge project that involves the use of modern technologies and provides an opportunity to innovate in the field of robotics and automation.
- **Accessibility:** The project aims to create a more intuitive and user-friendly method of controlling robots that is accessible to a wider range of users without specialized training.
- **Skill Development:** Working on this project involves learning new skills and gaining valuable experience in coding, electronics, and sensor technology.

1.3. Problem Description

In contemporary security landscapes, traditional surveillance methods often fall short in providing comprehensive coverage and real-time monitoring capabilities. Manual surveillance systems are labor-intensive, prone to human error, and limited in scalability, making them inadequate for effectively addressing modern security challenges. Additionally, the reliance on fixed cameras and stationary monitoring stations limits the flexibility and adaptability required to cover dynamic environments or large areas efficiently. Consequently, there's a pressing need for a versatile and autonomous surveillance solution capable of navigating diverse terrains, detecting obstacles, and transmitting data in real time to enhance security measures while providing users with convenient remote monitoring capabilities. This project aims to bridge this gap by developing a state-of-the-art Surveillance Robot, integrating advanced components and IoT connectivity to revolutionize surveillance practices.

1.4. Objectives

The objectives of the project are:

1. Develop a cost-effective and user-friendly device
2. Implement real-time video capture capabilities within the surveillance system to enhance situational awareness and security monitoring.
3. Establish robust IOT connectivity to enable remote control and data transmission, facilitating seamless integration with existing security infrastructure.
4. Enable easy integration of additional features for future enhancement

CHAPTER 2

2.1 Technology and Literature Survey

1. **“Intelligent combat robot 2015” by V. SHANKAR:** It has been described as developing a robotic vehicle for remote operation using RF technology and a wireless camera for monitoring purposes. The robot and camera can send real-time footage with night vision capabilities through a wireless network. This type of robot could be useful in war zones for spying purposes. In this technology, a robot can only be operated from a distance of ten meters. Robot with Bluetooth control: With the increased speed, a new classification technique was presented to improve the robot's range. The camera's link was frequently lost with this technology.

2. **Dr. S. Bhargavi and S. Manjunath Electronics and Communication:** The goal of this research is to reduce human casualties in terrorist attacks like the one on September 11, 2001. The combat robot was created to deal with such heinous terror acts. This robot is radio-controlled self-powered and equipped with all of the controls found in a typical car. It's been outfitted with a wireless camera so that it can keep an eye on the adversary from afar if necessary. It can enter enemy territory and transmit all information to us via its small camera eyes. This spy robot can be deployed at high-end hotels, shopping malls, and jewelry showrooms, among places where intruders or terrorists may pose a threat.

2.2 Block Diagram

Surveillance Robot project is divided into the following blocks:

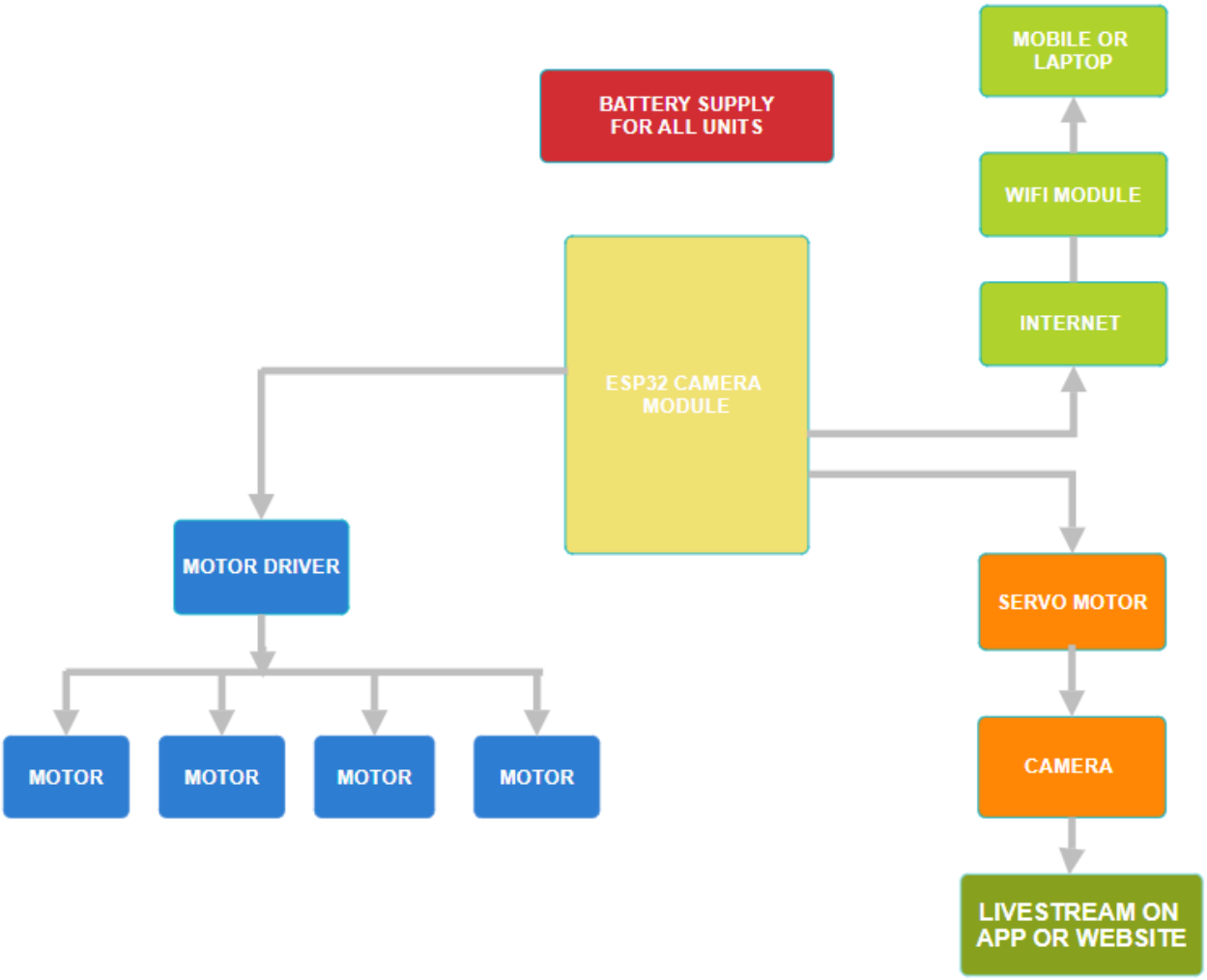

Figure 2.1: Block Diagram of Surveillance Robot

2.3 Hardware Required

2.3.1. Arduino Uno

Arduino is an open-source platform used for building electronics projects. Arduino consists of both a physical programmable circuit board and an IDE that runs on your computer, used to write and upload computer code to the physical board. The Arduino IDE uses a simplified version of C++, making it easier to learn to program.

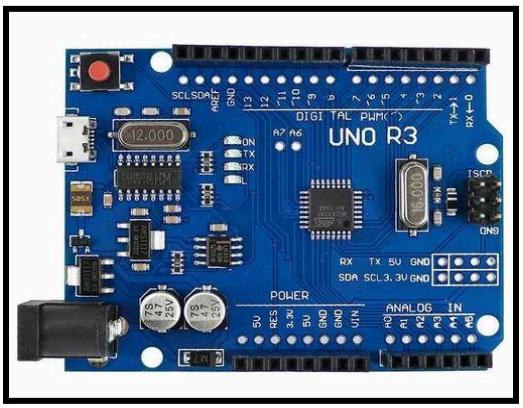

Figure 2.3.1: Arduino Uno

2.3.2 ESP32 Cam Module

The ESP32 Cam Module is a compact camera module capable of capturing images and streaming video over Wi-Fi. It integrates an ESP32 microcontroller and an OV2640 camera sensor, offering easy connectivity and high-quality imaging for mini surveillance or IoT projects.

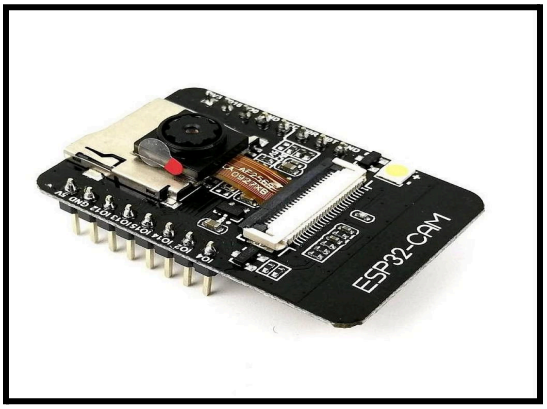

Figure 2.3.2: ESP32 Cam Module

2.3.3 Motor Driver

A motor driver is an electronic circuit or device that controls the speed, direction, and torque of an electric motor. The use of a motor driver is essential in many applications where precise motor control is required. It is essential in many applications, including robotics, automation, and industrial control systems. In our project, we are using a driver motor to accelerate the wheels of the robot.

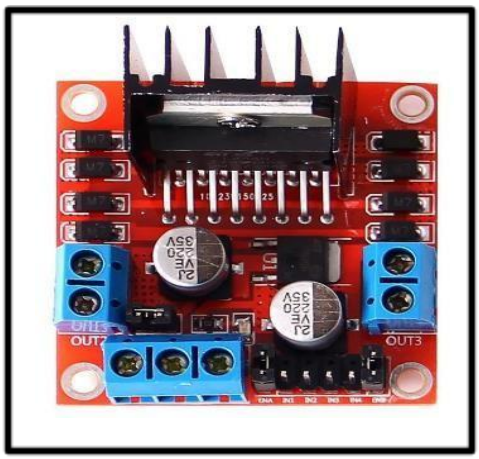

Figure 2.3.3 Motor Driver

2.3.4 Wheels and Chassis

Wheels and chassis are two important components of a vehicle that work together to provide stability, maneuverability, and support. Wheels are round structures that are typically made of metal or rubber and are attached to the vehicle's axles. The chassis is the framework of a vehicle that supports the body and engine.
In our project for making the main body of the robot, we are requiring 4 wheels and chassis

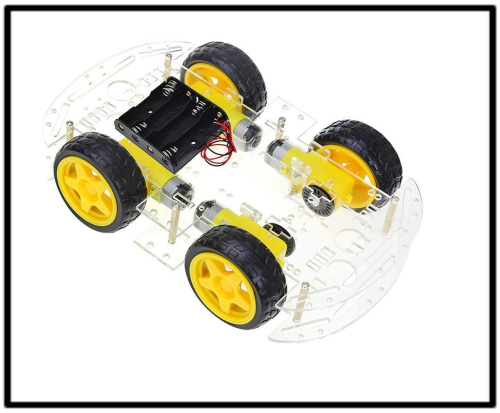

Figure 2.3.4: Wheels and Chassis

2.3.5 DC Motor

A DC (direct current) motor is a type of electric motor that converts electrical energy into mechanical energy through the use of a magnetic field. It operates by applying a voltage to the motor's terminals, which creates a magnetic field that interacts with the motor's armature, causing it to rotate. DC motors are commonly used in robotics and automation applications because they can be easily controlled and provide high torque at low speeds. They are also relatively simple and inexpensive compared to other types of motors. To use a DC motor in a hand gesture control robot, we need a motor driver circuit that can provide the appropriate voltage and current to the motor.

Figure 2. DC Motor

2.3 Software Required

For coding and uploading the sketch, the Arduino IDE is used.

2.4 Flow Chart

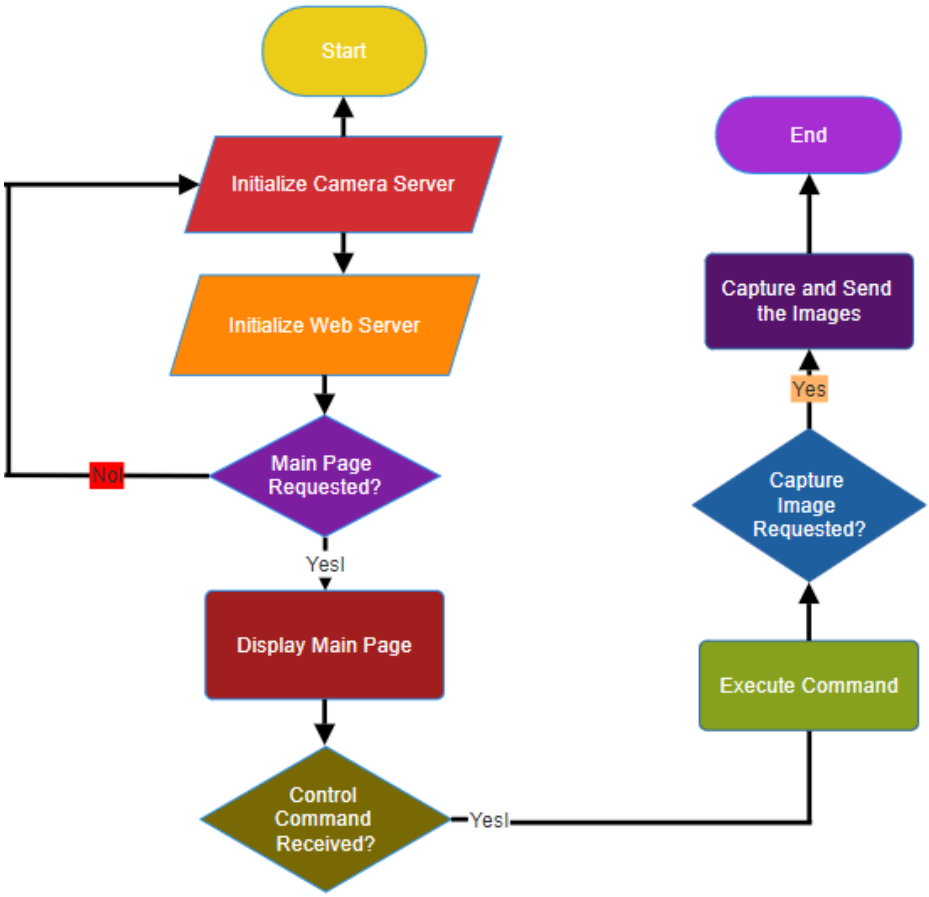

Fig. 2.4.1 Flow Chart

CHAPTER 3

3 Design and Implementation

3.1 Components

Required Components for the schematic:

- Arduino Uno.
- ESP32 Cam Module
- L298N Motor Driver
- DC Motor
- Connectors to join the different boards to form one functional device. Each of the hardware is dissected and was designed/implemented separately for its functionality and later incorporated as one whole application. This helped in the debugging processes. We can prepare by using this.

3.2 Hardware

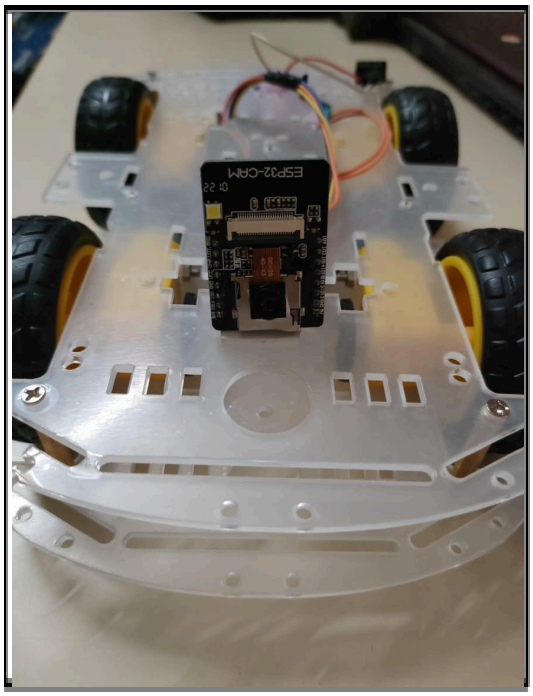

Fig. 3.2 Hardware Implementation

3.3 Working of Surveillance Robot using ESP 32 Cam:

The surveillance robot car using ESP32 Cam module operates based on the following principles:

1. **Image and Video Capture (ESP32 Cam Module):**
   The ESP32 Cam module captures images and videos using its onboard camera sensor.
   It processes the captured media to extract visual information about the surroundings.
2. **Integration and Control:**
   The ESP32 Cam module is integrated into the surveillance robot car's control system.
3. **Data Processing and Transmission:**
   The captured images and videos are processed by the ESP32 microcontroller for surveillance analysis.
   Relevant data or alerts may be generated based on detected events or anomalies.
   The data may be transmitted wirelessly for remote monitoring and analysis, using Wi-Fi or other communication protocols supported by the ESP32.

Overall, the ESP32 Cam module captures visual data, allowing the surveillance robot car to monitor its surroundings effectively. This integrated system enables autonomous surveillance capabilities with potential applications in home security, monitoring, and surveillance tasks.

CHAPTER 4

4.1. Applications

1. **Home Security:** Patrol homes and properties, detecting intruders and sending alerts to homeowners.
2. **Commercial Security:** Monitor business premises, warehouses, or retail stores to deter theft and unauthorized access.
3. **Search and Rescue:** Assist in disaster scenarios by navigating hazardous terrain to locate survivors and assess damage.
4. **Monitoring and Inspection:** Provide real-time feedback in environments like construction sites or industrial facilities to identify issues quickly.
5. **Education and Research:** Enhance learning in classrooms by teaching robotics, computer vision, and IoT concepts. Also, support research in robotics, artificial intelligence, and environmental monitoring.

4.2. Advantages

1. **Compact Size**: ESP32 Cam's small design enables robots to navigate tight spaces effectively.
2. **Affordability:** It's cost-effective, making surveillance accessible to a wide range of users.
3. **Wireless Connectivity**: Supports Wi-Fi for remote monitoring and live video streaming.
4. **High-Quality Imaging:** Equipped with an OV2640 sensor for clear and detailed visuals.
5. **Versatility:** Customizable with additional sensors for various surveillance tasks.

4.3. Disadvantages

1. **Limited range and accuracy:** The range and accuracy of hand gestures can be limited, which may make it difficult to control the robot precisely or from a distance.
2. **Limited range of motion:** Hand gestures can only control the robot within a limited range of motion. This can make it difficult to control the robot in certain situations or to perform complex tasks

4.4 Conclusion

In conclusion, the surveillance robot using ESP32-CAM is a promising project that can be used for various applications, such as monitoring and surveillance of homes or offices. The ESP32-CAM module provides a compact and cost-effective solution for capturing images and streaming video. By integrating it with a robot platform, it is possible to remotely control the robot and view live video footage from the camera. With the increasing demand for remote monitoring and surveillance. This project demonstrates the power of technological innovation in addressing contemporary challenges and contributing to a smarter and safer environment.

It has the potential to provide a practical and affordable solution for many different scenarios.

COST ESTIMATION

Sr. No.	Name of Component	Quantity	Price Rs.
1	Arduino UNO	1	**789**
2	Driver motor	1	**115**
3	3.7V Li battery	3	**180**
4	Wheels	4	**180**
5	DC motors	4	**200**
6	ESP32 CAM module	1	**472**
Total Rs.			**1936/-**

Table 2: Cost of Project

APPENDICES

Program Code:

ESP32CAM_Car.ino

```
#include "esp_camera.h"
#include <WiFi.h>

#define CAMERA_MODEL_AI_THINKER

const char* ssid     = "WIFI_NAME";
const char* password = "WIFI_PASSWORD";



#if defined(CAMERA_MODEL_WROVER_KIT)
#define PWDN_GPIO_NUM    -1
#define RESET_GPIO_NUM   -1
#define XCLK_GPIO_NUM    21
#define SIOD_GPIO_NUM    26
#define SIOC_GPIO_NUM    27

#define Y9_GPIO_NUM      35
#define Y8_GPIO_NUM      34
#define Y7_GPIO_NUM      39
#define Y6_GPIO_NUM      36
#define Y5_GPIO_NUM      19
#define Y4_GPIO_NUM      18
#define Y3_GPIO_NUM       5
#define Y2_GPIO_NUM       4
#define VSYNC_GPIO_NUM   25
#define HREF_GPIO_NUM    23
#define PCLK_GPIO_NUM    22


#elif defined(CAMERA_MODEL_AI_THINKER)
#define PWDN_GPIO_NUM     32
#define RESET_GPIO_NUM    -1
#define XCLK_GPIO_NUM      0
#define SIOD_GPIO_NUM     26
#define SIOC_GPIO_NUM     27

#define Y9_GPIO_NUM       35
#define Y8_GPIO_NUM       34
#define Y7_GPIO_NUM       39
#define Y6_GPIO_NUM       36
#define Y5_GPIO_NUM       21
#define Y4_GPIO_NUM       19
#define Y3_GPIO_NUM       18
#define Y2_GPIO_NUM        5
#define VSYNC_GPIO_NUM    25
```

```
#define HREF_GPIO_NUM     23
#define PCLK_GPIO_NUM     22

#else
#error "Camera model not selected"
#endif

// GPIO Setting
extern int gpLb =  2; // Left 1
extern int gpLf = 14; // Left 2
extern int gpRb = 15; // Right 1
extern int gpRf = 13; // Right 2
extern int gpLed =  4; // Light
extern String WiFiAddr ="";

void startCameraServer();

void setup() {
  Serial.begin(115200);
  Serial.setDebugOutput(true);
  Serial.println();


  pinMode(gpLb, OUTPUT); //Left Backward
  pinMode(gpLf, OUTPUT); //Left Forward
  pinMode(gpRb, OUTPUT); //Right Forward
  pinMode(gpRf, OUTPUT); //Right Backward
  pinMode(gpLed, OUTPUT); //Light

  //initialize
  digitalWrite(gpLb, LOW);
  digitalWrite(gpLf, LOW);
  digitalWrite(gpRb, LOW);
  digitalWrite(gpRf, LOW);
  digitalWrite(gpLed, LOW);

  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
```

```
  config.pixel_format = PIXFORMAT_JPEG;
  //init with high specs to pre-allocate larger buffers
  if(psramFound()){
    config.frame_size = FRAMESIZE_UXGA;
    config.jpeg_quality = 10;
    config.fb_count = 2;
  } else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
  }

  // camera init
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }

  //drop down frame size for higher initial frame rate
  sensor_t * s = esp_camera_sensor_get();
  s->set_framesize(s, FRAMESIZE_CIF);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");

  startCameraServer();

  Serial.print("Camera Ready! Use 'http://");
  Serial.print(WiFi.localIP());
  WiFiAddr = WiFi.localIP().toString();
  Serial.println("' to connect");
}

void loop() {
  // put your main code here, to run repeatedly:

}
```

App_http.cpp：

```
#include "esp_http_server.h"
#include "esp_timer.h"
#include "esp_camera.h"
#include "img_converters.h"
#include "camera_index.h"
#include "Arduino.h"

extern int gpLb;
extern int gpLf;
extern int gpRb;
extern int gpRf;
extern int gpLed;
extern String WiFiAddr;

void WheelAct(int nLf, int nLb, int nRf, int nRb);

typedef struct {
        size_t size; //number of values used for filtering
        size_t index; //current value index
        size_t count; //value count
        int sum;
        int * values; //array to be filled with values
} ra_filter_t;

typedef struct {
        httpd_req_t *req;
        size_t len;
} jpg_chunking_t;

#define PART_BOUNDARY "123456789000000000000987654321"
static const char* _STREAM_CONTENT_TYPE = "multipart/x-mixed-replace;boundary="
PART_BOUNDARY;
static const char* _STREAM_BOUNDARY = "\r\n--" PART_BOUNDARY "\r\n";
static const char* _STREAM_PART = "Content-Type: image/jpeg\r\nContent-Length:
%u\r\n\r\n";

static ra_filter_t ra_filter;
httpd_handle_t stream_httpd = NULL;
httpd_handle_t camera_httpd = NULL;

static ra_filter_t * ra_filter_init(ra_filter_t * filter, size_t sample_size){
    memset(filter, 0, sizeof(ra_filter_t));

    filter->values = (int *)malloc(sample_size * sizeof(int));
    if(!filter->values){
        return NULL;
    }
    memset(filter->values, 0, sample_size * sizeof(int));

    filter->size = sample_size;
    return filter;
}

static int ra_filter_run(ra_filter_t * filter, int value){
```

```
    if(!filter->values){
        return value;
    }
    filter->sum -= filter->values[filter->index];
    filter->values[filter->index] = value;
    filter->sum += filter->values[filter->index];
    filter->index++;
    filter->index = filter->index % filter->size;
    if (filter->count < filter->size) {
        filter->count++;
    }
    return filter->sum / filter->count;
}

static size_t jpg_encode_stream(void * arg, size_t index, const void* data, size_t
len){
    jpg_chunking_t *j = (jpg_chunking_t *)arg;
    if(!index){
        j->len = 0;
    }
    if(httpd_resp_send_chunk(j->req, (const char *)data, len) != ESP_OK){
        return 0;
    }
    j->len += len;
    return len;
}

static esp_err_t capture_handler(httpd_req_t *req){
    camera_fb_t * fb = NULL;
    esp_err_t res = ESP_OK;
    int64_t fr_start = esp_timer_get_time();

    fb = esp_camera_fb_get();
    if (!fb) {
        Serial.printf("Camera capture failed");
        httpd_resp_send_500(req);
        return ESP_FAIL;
    }

    httpd_resp_set_type(req, "image/jpeg");
    httpd_resp_set_hdr(req, "Content-Disposition", "inline;
filename=capture.jpg");

    size_t fb_len = 0;
    if(fb->format == PIXFORMAT_JPEG){
        fb_len = fb->len;
        res = httpd_resp_send(req, (const char *)fb->buf, fb->len);
    } else {
        jpg_chunking_t jchunk = {req, 0};
        res = frame2jpg_cb(fb, 80, jpg_encode_stream, &jchunk)?ESP_OK:ESP_FAIL;
        httpd_resp_send_chunk(req, NULL, 0);
        fb_len = jchunk.len;
    }
    esp_camera_fb_return(fb);
    int64_t fr_end = esp_timer_get_time();
    Serial.printf("JPG: %uB %ums", (uint32_t)(fb_len), (uint32_t)((fr_end -
```

```
fr_start)/1000));
    return res;
}

static esp_err_t stream_handler(httpd_req_t *req){
    camera_fb_t * fb = NULL;
    esp_err_t res = ESP_OK;
    size_t _jpg_buf_len = 0;
    uint8_t * _jpg_buf = NULL;
    char * part_buf[64];

    static int64_t last_frame = 0;
    if(!last_frame) {
        last_frame = esp_timer_get_time();
    }

    res = httpd_resp_set_type(req, _STREAM_CONTENT_TYPE);
    if(res != ESP_OK){
        return res;
    }

    while(true){
        fb = esp_camera_fb_get();
        if (!fb) {
            Serial.printf("Camera capture failed");
            res = ESP_FAIL;
        } else {
            if(fb->format != PIXFORMAT_JPEG){
                bool jpeg_converted = frame2jpg(fb, 80, &_jpg_buf, &_jpg_buf_len);
                esp_camera_fb_return(fb);
                fb = NULL;
                if(!jpeg_converted){
                    Serial.printf("JPEG compression failed");
                    res = ESP_FAIL;
                }
            } else {
                _jpg_buf_len = fb->len;
                _jpg_buf = fb->buf;
            }
        }
        if(res == ESP_OK){
            size_t hlen = snprintf((char *)part_buf, 64, _STREAM_PART,
_jpg_buf_len);
            res = httpd_resp_send_chunk(req, (const char *)part_buf, hlen);
        }
        if(res == ESP_OK){
            res = httpd_resp_send_chunk(req, (const char *)_jpg_buf,
_jpg_buf_len);
        }
        if(res == ESP_OK){
            res = httpd_resp_send_chunk(req, _STREAM_BOUNDARY,
strlen(_STREAM_BOUNDARY));
        }
        if(fb){
            esp_camera_fb_return(fb);
            fb = NULL;
```

```
            _jpg_buf = NULL;
        } else if(_jpg_buf){
            free(_jpg_buf);
            _jpg_buf = NULL;
        }
        if(res != ESP_OK){
            break;
        }
        int64_t fr_end = esp_timer_get_time();

        int64_t frame_time = fr_end - last_frame;
        last_frame = fr_end;
        frame_time /= 1000;
        uint32_t avg_frame_time = ra_filter_run(&ra_filter, frame_time);
        Serial.printf("MJPG: %uB %ums (%.1ffps), AVG: %ums (%.1ffps)"
            ,(uint32_t)(_jpg_buf_len),
            (uint32_t)frame_time, 1000.0 / (uint32_t)frame_time,
            avg_frame_time, 1000.0 / avg_frame_time
        );
    }

    last_frame = 0;
    return res;
}

static esp_err_t cmd_handler(httpd_req_t *req){
    char*  buf;
    size_t buf_len;
    char variable[32] = {0,};
    char value[32] = {0,};

    buf_len = httpd_req_get_url_query_len(req) + 1;
    if (buf_len > 1) {
        buf = (char*)malloc(buf_len);
        if(!buf){
            httpd_resp_send_500(req);
            return ESP_FAIL;
        }
        if (httpd_req_get_url_query_str(req, buf, buf_len) == ESP_OK) {
            if (httpd_query_key_value(buf, "var", variable, sizeof(variable)) ==
ESP_OK &&
                httpd_query_key_value(buf, "val", value, sizeof(value)) == ESP_OK)
{
            } else {
                free(buf);
                httpd_resp_send_404(req);
                return ESP_FAIL;
            }
        } else {
            free(buf);
            httpd_resp_send_404(req);
            return ESP_FAIL;
        }
        free(buf);
    } else {
        httpd_resp_send_404(req);
```

```
        return ESP_FAIL;
    }

    int val = atoi(value);
    sensor_t * s = esp_camera_sensor_get();
    int res = 0;

    if(!strcmp(variable, "framesize")) {
        if(s->pixformat == PIXFORMAT_JPEG) res = s->set_framesize(s,
(framesize_t)val);
    }
    else if(!strcmp(variable, "quality")) res = s->set_quality(s, val);
    else if(!strcmp(variable, "contrast")) res = s->set_contrast(s, val);
    else if(!strcmp(variable, "brightness")) res = s->set_brightness(s, val);
    else if(!strcmp(variable, "saturation")) res = s->set_saturation(s, val);
    else if(!strcmp(variable, "gainceiling")) res = s->set_gainceiling(s,
(gainceiling_t)val);
    else if(!strcmp(variable, "colorbar")) res = s->set_colorbar(s, val);
    else if(!strcmp(variable, "awb")) res = s->set_whitebal(s, val);
    else if(!strcmp(variable, "agc")) res = s->set_gain_ctrl(s, val);
    else if(!strcmp(variable, "aec")) res = s->set_exposure_ctrl(s, val);
    else if(!strcmp(variable, "hmirror")) res = s->set_hmirror(s, val);
    else if(!strcmp(variable, "vflip")) res = s->set_vflip(s, val);
    else if(!strcmp(variable, "awb_gain")) res = s->set_awb_gain(s, val);
    else if(!strcmp(variable, "agc_gain")) res = s->set_agc_gain(s, val);
    else if(!strcmp(variable, "aec_value")) res = s->set_aec_value(s, val);
    else if(!strcmp(variable, "aec2")) res = s->set_aec2(s, val);
    else if(!strcmp(variable, "dcw")) res = s->set_dcw(s, val);
    else if(!strcmp(variable, "bpc")) res = s->set_bpc(s, val);
    else if(!strcmp(variable, "wpc")) res = s->set_wpc(s, val);
    else if(!strcmp(variable, "raw_gma")) res = s->set_raw_gma(s, val);
    else if(!strcmp(variable, "lenc")) res = s->set_lenc(s, val);
    else if(!strcmp(variable, "special_effect")) res = s->set_special_effect(s,
val);
    else if(!strcmp(variable, "wb_mode")) res = s->set_wb_mode(s, val);
    else if(!strcmp(variable, "ae_level")) res = s->set_ae_level(s, val);
    else {
        res = -1;
    }

    if(res){
        return httpd_resp_send_500(req);
    }

    httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
    return httpd_resp_send(req, NULL, 0);
}

static esp_err_t status_handler(httpd_req_t *req){
    static char json_response[1024];

    sensor_t * s = esp_camera_sensor_get();
    char * p = json_response;
    *p++ = '{';

    p+=sprintf(p, "\"framesize\":%u,", s->status.framesize);
```

```
    p+=sprintf(p, "\"quality\":%u,", s->status.quality);
    p+=sprintf(p, "\"brightness\":%d,", s->status.brightness);
    p+=sprintf(p, "\"contrast\":%d,", s->status.contrast);
    p+=sprintf(p, "\"saturation\":%d,", s->status.saturation);
    p+=sprintf(p, "\"special_effect\":%u,", s->status.special_effect);
    p+=sprintf(p, "\"wb_mode\":%u,", s->status.wb_mode);
    p+=sprintf(p, "\"awb\":%u,", s->status.awb);
    p+=sprintf(p, "\"awb_gain\":%u,", s->status.awb_gain);
    p+=sprintf(p, "\"aec\":%u,", s->status.aec);
    p+=sprintf(p, "\"aec2\":%u,", s->status.aec2);
    p+=sprintf(p, "\"ae_level\":%d,", s->status.ae_level);
    p+=sprintf(p, "\"aec_value\":%u,", s->status.aec_value);
    p+=sprintf(p, "\"agc\":%u,", s->status.agc);
    p+=sprintf(p, "\"agc_gain\":%u,", s->status.agc_gain);
    p+=sprintf(p, "\"gainceiling\":%u,", s->status.gainceiling);
    p+=sprintf(p, "\"bpc\":%u,", s->status.bpc);
    p+=sprintf(p, "\"wpc\":%u,", s->status.wpc);
    p+=sprintf(p, "\"raw_gma\":%u,", s->status.raw_gma);
    p+=sprintf(p, "\"lenc\":%u,", s->status.lenc);
    p+=sprintf(p, "\"hmirror\":%u,", s->status.hmirror);
    p+=sprintf(p, "\"dcw\":%u,", s->status.dcw);
    p+=sprintf(p, "\"colorbar\":%u", s->status.colorbar);
    *p++ = '}';
    *p++ = 0;
    httpd_resp_set_type(req, "application/json");
    httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
    return httpd_resp_send(req, json_response, strlen(json_response));
}

static esp_err_t index_handler(httpd_req_t *req){
    httpd_resp_set_type(req, "text/html");
    String page = "";
     page += "<meta name=\"viewport\" content=\"width=device-width,
initial-scale=1.0, maximum-scale=1.0, user-scalable=0\">\n";
 page += "<script>var xhttp = new XMLHttpRequest();</script>";
 page += "<script>function getsend(arg) { xhttp.open('GET', arg +'?' + new
Date().getTime(), true); xhttp.send() } </script>";
 //page += "<p align=center><IMG SRC='http://" + WiFiAddr + ":81/stream'
style='width:280px;'></p><br/><br/>";
 page += "<p align=center><IMG SRC='http://" + WiFiAddr + ":81/stream'
style='width:300px; transform:rotate(180deg);'></p><br/><br/>";

 page += "<p align=center> <button
style=background-color:lightgrey;width:90px;height:80px onmousedown=getsend('go')
onmouseup=getsend('stop') ontouchstart=getsend('go') ontouchend=getsend('stop')
><b>Forward</b></button> </p>";
 page += "<p align=center>";
 page += "<button style=background-color:lightgrey;width:90px;height:80px;
onmousedown=getsend('left') onmouseup=getsend('stop') ontouchstart=getsend('left')
ontouchend=getsend('stop')><b>Left</b></button> ";
 page += "<button style=background-color:indianred;width:90px;height:80px
onmousedown=getsend('stop') onmouseup=getsend('stop')><b>Stop</b></button> ";
 page += "<button style=background-color:lightgrey;width:90px;height:80px
onmousedown=getsend('right') onmouseup=getsend('stop')
ontouchstart=getsend('right') ontouchend=getsend('stop')><b>Right</b></button>";
 page += "</p>";
```

```
 page += "<p align=center><button
style=background-color:lightgrey;width:90px;height:80px
onmousedown=getsend('back') onmouseup=getsend('stop') ontouchstart=getsend('back')
ontouchend=getsend('stop') ><b>Backward</b></button></p>";

 page += "<p align=center>";
 page += "<button style=background-color:yellow;width:140px;height:40px
onmousedown=getsend('ledon')><b>Light ON</b></button>";
 page += "<button style=background-color:yellow;width:140px;height:40px
onmousedown=getsend('ledoff')><b>Light OFF</b></button>";
 page += "</p>";

    return httpd_resp_send(req, &page[0], strlen(&page[0]));
}

static esp_err_t go_handler(httpd_req_t *req){
    WheelAct(HIGH, LOW, HIGH, LOW);
    Serial.println("Go");
    httpd_resp_set_type(req, "text/html");
    return httpd_resp_send(req, "OK", 2);
}
static esp_err_t back_handler(httpd_req_t *req){
    WheelAct(LOW, HIGH, LOW, HIGH);
    Serial.println("Back");
    httpd_resp_set_type(req, "text/html");
    return httpd_resp_send(req, "OK", 2);
}

static esp_err_t left_handler(httpd_req_t *req){
    WheelAct(LOW, HIGH, HIGH, LOW);
    Serial.println("Right");
    httpd_resp_set_type(req, "text/html");
    return httpd_resp_send(req, "OK", 2);
}
static esp_err_t right_handler(httpd_req_t *req){


    WheelAct(HIGH, LOW, LOW, HIGH);
    Serial.println("Left");
    httpd_resp_set_type(req, "text/html");
    return httpd_resp_send(req, "OK", 2);
}

static esp_err_t stop_handler(httpd_req_t *req){
    WheelAct(LOW, LOW, LOW, LOW);
    Serial.println("Stop");
    httpd_resp_set_type(req, "text/html");
    return httpd_resp_send(req, "OK", 2);
}

static esp_err_t ledon_handler(httpd_req_t *req){
    digitalWrite(gpLed, HIGH);
    Serial.println("LED ON");
    httpd_resp_set_type(req, "text/html");
    return httpd_resp_send(req, "OK", 2);
```

```
}
static esp_err_t ledoff_handler(httpd_req_t *req){
    digitalWrite(gpLed, LOW);
    Serial.println("LED OFF");
    httpd_resp_set_type(req, "text/html");
    return httpd_resp_send(req, "OK", 2);
}

void startCameraServer(){
    httpd_config_t config = HTTPD_DEFAULT_CONFIG();

    httpd_uri_t go_uri = {
        .uri       = "/go",
        .method    = HTTP_GET,
        .handler   = go_handler,
        .user_ctx  = NULL
    };

    httpd_uri_t back_uri = {
        .uri       = "/back",
        .method    = HTTP_GET,
        .handler   = back_handler,
        .user_ctx  = NULL
    };

    httpd_uri_t stop_uri = {
        .uri       = "/stop",
        .method    = HTTP_GET,
        .handler   = stop_handler,
        .user_ctx  = NULL
    };

    httpd_uri_t left_uri = {
        .uri       = "/left",
        .method    = HTTP_GET,
        .handler   = left_handler,
        .user_ctx  = NULL
    };

    httpd_uri_t right_uri = {
        .uri       = "/right",
        .method    = HTTP_GET,
        .handler   = right_handler,
        .user_ctx  = NULL
    };

    httpd_uri_t ledon_uri = {
        .uri       = "/ledon",
        .method    = HTTP_GET,
        .handler   = ledon_handler,
        .user_ctx  = NULL
    };

    httpd_uri_t ledoff_uri = {
        .uri       = "/ledoff",
        .method    = HTTP_GET,
```

```
    .handler   = ledoff_handler,
    .user_ctx  = NULL
};

httpd_uri_t index_uri = {
    .uri       = "/",
    .method    = HTTP_GET,
    .handler   = index_handler,
    .user_ctx  = NULL
};

httpd_uri_t status_uri = {
    .uri       = "/status",
    .method    = HTTP_GET,
    .handler   = status_handler,
    .user_ctx  = NULL
};

httpd_uri_t cmd_uri = {
    .uri       = "/control",
    .method    = HTTP_GET,
    .handler   = cmd_handler,
    .user_ctx  = NULL
};

httpd_uri_t capture_uri = {
    .uri       = "/capture",
    .method    = HTTP_GET,
    .handler   = capture_handler,
    .user_ctx  = NULL
};

httpd_uri_t stream_uri = {
    .uri       = "/stream",
    .method    = HTTP_GET,
    .handler   = stream_handler,
    .user_ctx  = NULL
};


ra_filter_init(&ra_filter, 20);
Serial.printf("Starting web server on port: '%d'", config.server_port);
if (httpd_start(&camera_httpd, &config) == ESP_OK) {
    httpd_register_uri_handler(camera_httpd, &index_uri);
    httpd_register_uri_handler(camera_httpd, &go_uri);
    httpd_register_uri_handler(camera_httpd, &back_uri);
    httpd_register_uri_handler(camera_httpd, &stop_uri);
    httpd_register_uri_handler(camera_httpd, &left_uri);
    httpd_register_uri_handler(camera_httpd, &right_uri);
    httpd_register_uri_handler(camera_httpd, &ledon_uri);
    httpd_register_uri_handler(camera_httpd, &ledoff_uri);
}

config.server_port += 1;
config.ctrl_port += 1;
Serial.printf("Starting stream server on port: '%d'", config.server_port);
```

```
    if (httpd_start(&stream_httpd, &config) == ESP_OK) {
        httpd_register_uri_handler(stream_httpd, &stream_uri);
    }
}

void WheelAct(int nLf, int nLb, int nRf, int nRb)
{
 digitalWrite(gpLf, nLf);
 digitalWrite(gpLb, nLb);
 digitalWrite(gpRf, nRf);
 digitalWrite(gpRb, nRb);
}
```

References

1. Montrose, M. I. (1996). Printed circuit board and design techniques for EMC compliance (Vol. 1, p. 996). Piscataway, NJ: IEEE Press.
2. Forouzan, B. A., & Fegan, S. C. (2006). TCP/IP protocol suite (Vol. 2). McGrawHill.
3. Balakrishnan, H., Stemm, M., Seshan, S., & Katz, R. H. (1997). Analyzing stability in wide-area network
4. performance. ACM SIGMETRICS Performance and Evaluation Review, 25(1), 2-12.
5. Briscoe, N. (2000). Understanding the OSI 7-layer model. PC Network Advisor, 120(2).
6. G.Huston, Analyzing the Internet BGP routing table, Internet ProtocolJournal 4 (1) (2001).
7. McKeown, N. (2009). Software-defined networking. INFOCOM keynote talk, 17(2), 30-32.
8. https://components101.com/microcontrollers/arduino-uno
   https://w98ujBhCgARIsAD7QeAhulaq0nKHQe9Tv1hL4a9KacWRvE2yUbH7dRY0wIJLfBpKXWlUC4waAt6qEALw_wcB
9. https://www.verical.com/datasheet/adafruit-brushless-dc-motors-