CIRCUIT DIAGRAM

Click image to zoom in or out.

Sketch/Code
The sketch bellow is code for Wifi Robot Remote Control Mode with realtime camera view, you can use this Android phone to directly upload firmware to ESP32-Cam either via USB or Wifi OTA, by pressing the upload icon below.

If you need to edit the sketch with computer using Arduino IDE, you can share this sketch to email or online storage by pressing the share icon below.

esp32cam_wifi_robot_car.ino

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

/* Wifi Crdentials */
String sta_ssid = "";     // set Wifi network you want to connect to
String sta_password = "";   // set password for Wifi network


/* define 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

/* Defining motor and servo pins */
```

```
extern int DRV_A = 12;
extern int DRV_B = 13;
extern int DIR_A = 14;
extern int DIR_B = 15;

extern int ledVal = 20;  // setting bright of flash LED 0-255

extern int ledPin = 4;  // set digital pin GPIO4 as LED pin (use biult-in LED)
extern int servoPin = 2;  // set digital pin GPIO2 as servo pin (use SG90)

unsigned long previousMillis = 0;

void startCameraServer();

void initServo() {
  ledcSetup(8, 50, 16); /*50 hz PWM, 16-bit resolution and range from 3250 to 6500 */
  ledcAttachPin(servoPin, 8);
}

void initLed() {
  ledcSetup(7, 5000, 8); /* 5000 hz PWM, 8-bit resolution and range from 0 to 255 */
  ledcAttachPin(ledPin, 7);
}

void setup() {
  Serial.begin(115200);         // set up seriamonitor at 115200 bps
  Serial.setDebugOutput(true);
  Serial.println();
  Serial.println("*ESP32 Camera Remote Control - L293D Bluino Shield*");
  Serial.println("-----------------------------------------------------");

  // Set all the motor control pin to Output
  pinMode(DRV_A, OUTPUT);
  pinMode(DRV_B, OUTPUT);
  pinMode(DIR_A, OUTPUT);
  pinMode(DIR_B, OUTPUT);

  pinMode(ledPin, OUTPUT); // set the LED pin as an Output
  pinMode(servoPin, OUTPUT); // set the servo pin as an Output

  // Initial state - turn off motors, LED & buzzer
  digitalWrite(DRV_A, LOW);
  digitalWrite(DRV_B, LOW);
```

```
digitalWrite(DIR_A, LOW);
digitalWrite(DIR_B, LOW);
digitalWrite(ledPin, LOW);
digitalWrite(servoPin, LOW);

/* Initializing Servo and LED */
initServo();
initLed();

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);

// Set NodeMCU Wifi hostname based on chip mac address
char chip_id[15];
snprintf(chip_id, 15, "%04X", (uint16_t)(ESP.getEfuseMac()>>32));
String hostname = "esp32cam-" + String(chip_id);

Serial.println();
Serial.println("Hostname: "+hostname);

// first, set NodeMCU as STA mode to connect with a Wifi network
WiFi.mode(WIFI_STA);
WiFi.begin(sta_ssid.c_str(), sta_password.c_str());
Serial.println("");
Serial.print("Connecting to: ");
Serial.println(sta_ssid);
Serial.print("Password: ");
Serial.println(sta_password);

// try to connect with Wifi network about 10 seconds
unsigned long currentMillis = millis();
previousMillis = currentMillis;
while (WiFi.status() != WL_CONNECTED && currentMillis - previousMillis <= 10000) {
  delay(500);
  Serial.print(".");
  currentMillis = millis();
}

// if failed to connect with Wifi network set NodeMCU as AP mode
IPAddress myIP;
if (WiFi.status() == WL_CONNECTED) {
  Serial.println("");
  Serial.println("*WiFi-STA-Mode*");
  Serial.print("IP: ");
  myIP=WiFi.localIP();
  Serial.println(myIP);
```

```
    delay(2000);
  } else {
    WiFi.mode(WIFI_AP);
    WiFi.softAP(hostname.c_str());
    myIP = WiFi.softAPIP();
    Serial.println("");
    Serial.println("WiFi failed connected to " + sta_ssid);
    Serial.println("");
    Serial.println("*WiFi-AP-Mode*");
    Serial.print("AP IP address: ");
    Serial.println(myIP);
    delay(2000);
  }

  // Start camera server to get realtime view
  startCameraServer();
  Serial.print("Camera Ready! Use 'http://");
  Serial.print(myIP);
  Serial.println("' to connect ");

  ArduinoOTA.begin();   // enable to receive update/upload firmware via Wifi OTA
}

void loop() {
  // put your main code here, to run repeatedly:
  ArduinoOTA.handle();
}
```

app_httpd.cpp

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

/* Initializing pins */
extern int DRV_A;
extern int DRV_B;
extern int DIR_A;
extern int DIR_B;
extern int ledPin;
extern int servoPin;
```

```
extern int ledVal;

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";

httpd_handle_t stream_httpd = NULL;
httpd_handle_t camera_httpd = NULL;


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;
   }

   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);
        Serial.println(buf);
        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);
                Serial.println(buf);
                httpd_resp_send_404(req);
                return ESP_FAIL;
            }
        } else {
            free(buf);
            Serial.println(buf);
            httpd_resp_send_404(req);
            return ESP_FAIL;
        }
        Serial.println(buf);
        free(buf);
    } else {
        httpd_resp_send_404(req);
        Serial.println(ESP_FAIL);
        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 state_handler(httpd_req_t *req){
    char*  buf;
    size_t buf_len;
    char cmd[32] = {0,};

    buf_len = httpd_req_get_url_query_len(req) + 1;
    if (buf_len > 1) {
        buf = (char*)malloc(buf_len);
        Serial.println(buf);

        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, "cmd", cmd, sizeof(cmd)) == ESP_OK) {

        } else {
            free(buf);
            Serial.print("*");
            Serial.println(ESP_FAIL);
            httpd_resp_send_404(req);
            return ESP_FAIL;
        }
    } else {
        free(buf);
        Serial.print("**");
        Serial.println(ESP_FAIL);
        httpd_resp_send_404(req);
        return ESP_FAIL;
    }
    free(buf);

} else {
    Serial.print("***");
    Serial.println(ESP_FAIL);
    httpd_resp_send_404(req);
    return ESP_FAIL;
}

int res = 0;

if(!strcmp(cmd, "F")) {
  Serial.println("Forward");
  digitalWrite(DRV_A, HIGH);
  digitalWrite(DRV_B, HIGH);
  digitalWrite(DIR_A, HIGH);
  digitalWrite(DIR_B, HIGH);
  httpd_resp_set_type(req, "text/html");
  return httpd_resp_send(req, "OK", 2);
}

else if(!strcmp(cmd, "B")) {
```

```
  Serial.println("Backward");
  digitalWrite(DRV_A, HIGH);
  digitalWrite(DRV_B, HIGH);
  digitalWrite(DIR_A, LOW);
  digitalWrite(DIR_B, LOW);
  httpd_resp_set_type(req, "text/html");
  return httpd_resp_send(req, "OK", 2);
}

else if(!strcmp(cmd, "R")) {
  Serial.println("Turn Right");
  digitalWrite(DRV_A, HIGH);
  digitalWrite(DRV_B, HIGH);
  digitalWrite(DIR_A, LOW);
  digitalWrite(DIR_B, HIGH);
  httpd_resp_set_type(req, "text/html");
  return httpd_resp_send(req, "OK", 2);
}

else if(!strcmp(cmd, "L")) {
  Serial.println("Turn Left");
  digitalWrite(DRV_A, HIGH);
  digitalWrite(DRV_B, HIGH);
  digitalWrite(DIR_A, HIGH);
  digitalWrite(DIR_B, LOW);
  httpd_resp_set_type(req, "text/html");
  return httpd_resp_send(req, "OK", 2);
}

else if(!strcmp(cmd, "G")) {
  Serial.println("Forward Left");
  digitalWrite(DRV_A, HIGH);
  digitalWrite(DRV_B, LOW);
  digitalWrite(DIR_A, HIGH);
  digitalWrite(DIR_B, HIGH);
  httpd_resp_set_type(req, "text/html");
  return httpd_resp_send(req, "OK", 2);
}

else if(!strcmp(cmd, "H")) {
  Serial.println("Backward Left");
  digitalWrite(DRV_A, HIGH);
  digitalWrite(DRV_B, LOW);
```

```
  digitalWrite(DIR_A, LOW);
  digitalWrite(DIR_B, LOW);
  httpd_resp_set_type(req, "text/html");
  return httpd_resp_send(req, "OK", 2);
}

else if(!strcmp(cmd, "I")) {
  Serial.println("Forward Right");
  digitalWrite(DRV_A, LOW);
  digitalWrite(DRV_B, HIGH);
  digitalWrite(DIR_A, HIGH);
  digitalWrite(DIR_B, HIGH);
  httpd_resp_set_type(req, "text/html");
  return httpd_resp_send(req, "OK", 2);
}

else if(!strcmp(cmd, "J")) {
  Serial.println("Backward Right");
  digitalWrite(DRV_A, LOW);
  digitalWrite(DRV_B, HIGH);
  digitalWrite(DIR_A, LOW);
  digitalWrite(DIR_B, LOW);
  httpd_resp_set_type(req, "text/html");
  return httpd_resp_send(req, "OK", 2);
}

else if(!strcmp(cmd, "S")) {
  Serial.println("Stop");
  digitalWrite(DRV_A, LOW);
  digitalWrite(DRV_B, LOW);
  digitalWrite(DIR_A, LOW);
  digitalWrite(DIR_B, LOW);
  httpd_resp_set_type(req, "text/html");
  return httpd_resp_send(req, "OK", 2);
}

else if(!strcmp(cmd, "W")) {
  Serial.println("LED On");
  //digitalWrite(ledPin, HIGH);
  ledcWrite(7, ledVal);
  httpd_resp_set_type(req, "text/html");
  return httpd_resp_send(req, "OK", 2);
}
```

```
else if(!strcmp(cmd, "w")) {
  Serial.println("LED Off");
  //digitalWrite(ledPin, LOW);
  ledcWrite(7, 0);
  httpd_resp_set_type(req, "text/html");
  return httpd_resp_send(req, "OK", 2);
}

else if (!strcmp(cmd, "x")){
  Serial.println("Flash Light : Low (20)");
  ledVal = 20;
  ledcWrite(7, ledVal);
  httpd_resp_set_type(req, "text/html");
  return httpd_resp_send(req, "OK", 2);
}
else if (!strcmp(cmd, "y")){
  Serial.println("Flash Light : Medium (50)");
  ledVal = 50;
  ledcWrite(7, ledVal);
  httpd_resp_set_type(req, "text/html");
  return httpd_resp_send(req, "OK", 2);
}
else if (!strcmp(cmd, "z")){
  Serial.println("Flash Light : Bright (100)");
  ledVal = 100;
  ledcWrite(7, ledVal);
  httpd_resp_set_type(req, "text/html");
  return httpd_resp_send(req, "OK", 2);
}
else if (!strcmp(cmd, "Z")){
  Serial.println("Flash Light : Super Bright (255)");
  ledVal = 255;
  ledcWrite(7, ledVal);
  httpd_resp_set_type(req, "text/html");
  return httpd_resp_send(req, "OK", 2);
}

/* Controlling the servo motor angle with PWM */
/* ledcWrite(Channel, Dutycycle) dutycycle range : 3250-6500*/
else if (!strcmp(cmd, "0")){
  Serial.println("Servo 0 (3250)");
  ledcWrite(8, 3250);
```

```
  httpd_resp_set_type(req, "text/html");
  return httpd_resp_send(req, "OK", 2);
}
else if (!strcmp(cmd, "1")){
  Serial.println("Servo 1 (3575)");
  ledcWrite(8, 3575);
  httpd_resp_set_type(req, "text/html");
  return httpd_resp_send(req, "OK", 2);
}
else if (!strcmp(cmd, "2")){
  Serial.println("Servo 2 (3900)");
  ledcWrite(8, 3900);
  httpd_resp_set_type(req, "text/html");
  return httpd_resp_send(req, "OK", 2);
}
else if (!strcmp(cmd, "3")){
  Serial.println("Servo 3 (4225)");
  ledcWrite(8, 4225);
  httpd_resp_set_type(req, "text/html");
  return httpd_resp_send(req, "OK", 2);
}
else if (!strcmp(cmd, "4")){
  Serial.println("Servo 4 (4550)");
  ledcWrite(8, 4550);
  httpd_resp_set_type(req, "text/html");
  return httpd_resp_send(req, "OK", 2);
}
else if (!strcmp(cmd, "5")){
  Serial.println("Servo 5 (4875)");
  ledcWrite(8, 4875);
  httpd_resp_set_type(req, "text/html");
  return httpd_resp_send(req, "OK", 2);
}
else if (!strcmp(cmd, "6")){
  Serial.println("Servo 6 (5200)");
  ledcWrite(8, 5200);
  httpd_resp_set_type(req, "text/html");
  return httpd_resp_send(req, "OK", 2);
}
else if (!strcmp(cmd, "7")){
  Serial.println("Servo 7 (5525)");
  ledcWrite(8, 5525);
  httpd_resp_set_type(req, "text/html");
```

```
      return httpd_resp_send(req, "OK", 2);
    }
    else if (!strcmp(cmd, "8")){
      Serial.println("Servo 8 (5850)");
      ledcWrite(8, 5850);
      httpd_resp_set_type(req, "text/html");
      return httpd_resp_send(req, "OK", 2);
    }
    else if (!strcmp(cmd, "9")){
      Serial.println("Servo 9 (6175)");
      ledcWrite(8, 6175);
      httpd_resp_set_type(req, "text/html");
      return httpd_resp_send(req, "OK", 2);
    }
    else if (!strcmp(cmd, "q")){
      Serial.println("Servo q (6500)");
      ledcWrite(8, 6500);
      httpd_resp_set_type(req, "text/html");
      return httpd_resp_send(req, "OK", 2);
    }

    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);
}


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


    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
};

httpd_uri_t state_uri = {
    .uri       = "/state",
    .method    = HTTP_GET,
    .handler   = state_handler,
    .user_ctx  = NULL
};

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, &cmd_uri);
    httpd_register_uri_handler(camera_httpd, &capture_uri);
    httpd_register_uri_handler(camera_httpd, &status_uri);
    httpd_register_uri_handler(camera_httpd, &state_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);
```

```
    }
}
```