app_httpd.cpp.pdf

```
/*
 * @Date: 2020-11-27 11:45:09
 * @Description: ESP32 Camera Surveillance Car
 * @FilePath:
 */
#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(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 right_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 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);
}
```