SURVEILLANCE ROBOT

Group no.:A2

Code:

```
#include "esp_camera.h"
#include <Arduino.h>
#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <iostream>
#include <sstream>
#include <ESP32Servo.h>

#define PAN_PIN 14
#define TILT_PIN 15

Servo panServo;
Servo tiltServo;

struct MOTOR_PINS
{
  int pinEn;
  int pinIN1;
  int pinIN2;
};

std::vector<MOTOR_PINS> motorPins =
{
  {2, 12, 13}, //RIGHT_MOTOR Pins (EnA, IN1, IN2)
  {2, 1, 3},  //LEFT_MOTOR  Pins (EnB, IN3, IN4)
};
#define LIGHT_PIN 4

#define UP 1
#define DOWN 2
#define LEFT 3
```

```
#define RIGHT 4
#define STOP 0

#define RIGHT_MOTOR 0
#define LEFT_MOTOR 1

#define FORWARD 1
#define BACKWARD -1

const int PWMFreq = 1000; /* 1 KHz */
const int PWMResolution = 8;
const int PWMSpeedChannel = 2;
const int PWMLightChannel = 3;

//Camera related constants
#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

const char* ssid     = "MyWiFiCar";
const char* password = "12345678";

AsyncWebServer server(80);
AsyncWebSocket wsCamera("/Camera");
AsyncWebSocket wsCarInput("/CarInput");
```

```
uint32_t cameraClientId = 0;

const char* htmlHomePage PROGMEM = R"HTMLHOMEPAGE(
<!DOCTYPE html>
<html>
 <head>
  <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no">
  <style>
  .arrows {
   font-size:30px;
   color:red;
  }
  td.button {
   background-color:black;
   border-radius:25%;
   box-shadow: 5px 5px #888888;
  }
  td.button:active {
   transform: translate(5px,5px);
   box-shadow: none;
  }

  .noselect {
   -webkit-touch-callout: none; /* iOS Safari */
    -webkit-user-select: none; /* Safari */
     -khtml-user-select: none; /* Konqueror HTML */
       -moz-user-select: none; /* Firefox */
        -ms-user-select: none; /* Internet Explorer/Edge */
            user-select: none; /* Non-prefixed version, currently
                                  supported by Chrome and Opera */
  }

  .slidecontainer {
   width: 100%;
  }

  .slider {
```

```
  -webkit-appearance: none;
  width: 100%;
  height: 15px;
  border-radius: 5px;
  background: #d3d3d3;
  outline: none;
  opacity: 0.7;
  -webkit-transition: .2s;
  transition: opacity .2s;
 }

 .slider:hover {
  opacity: 1;
 }

 .slider::-webkit-slider-thumb {
  -webkit-appearance: none;
  appearance: none;
  width: 25px;
  height: 25px;
  border-radius: 50%;
  background: red;
  cursor: pointer;
 }

 .slider::-moz-range-thumb {
  width: 25px;
  height: 25px;
  border-radius: 50%;
  background: red;
  cursor: pointer;
 }

 </style>

</head>
<body class="noselect" align="center" style="background-color:white">
 <table id="mainTable" style="width:400px;margin:auto;table-layout:fixed" CELLSPACING=10>
```

```
    <tr>
     <img id="cameraImage" src="" style="width:400px;height:250px"></td>
    </tr>
    <tr>
     <td></td>
                          <td    class="button"    ontouchstart='sendButtonInput("MoveCar","1")'
ontouchend='sendButtonInput("MoveCar","0")'><span class="arrows" >&#8679;</span></td>
     <td></td>
    </tr>
    <tr>
                          <td    class="button"    ontouchstart='sendButtonInput("MoveCar","3")'
ontouchend='sendButtonInput("MoveCar","0")'><span class="arrows" >&#8678;</span></td>
     <td class="button"></td>
                          <td    class="button"    ontouchstart='sendButtonInput("MoveCar","4")'
ontouchend='sendButtonInput("MoveCar","0")'><span class="arrows" >&#8680;</span></td>
    </tr>
    <tr>
     <td></td>
                          <td    class="button"    ontouchstart='sendButtonInput("MoveCar","2")'
ontouchend='sendButtonInput("MoveCar","0")'><span class="arrows" >&#8681;</span></td>
     <td></td>
    </tr>
    <tr/><tr/>
    <tr>
     <td style="text-align:left"><b>Speed:</b></td>
     <td colspan=2>
      <div class="slidecontainer">
               <input  type="range"  min="0"  max="255"  value="150"  class="slider"  id="Speed"
oninput='sendButtonInput("Speed",value)'>
       </div>
     </td>
    </tr>
    <tr>
     <td style="text-align:left"><b>Light:</b></td>
     <td colspan=2>
      <div class="slidecontainer">
                  <input  type="range"  min="0"  max="255"  value="0"  class="slider"  id="Light"
oninput='sendButtonInput("Light",value)'>
```

```
      </div>
     </td>
    </tr>
    <tr>
     <td style="text-align:left"><b>Pan:</b></td>
     <td colspan=2>
      <div class="slidecontainer">
                <input type="range" min="0" max="180" value="90" class="slider" id="Pan" oninput='sendButtonInput("Pan",value)'>
      </div>
     </td>
    </tr>
    <tr>
     <td style="text-align:left"><b>Tilt:</b></td>
     <td colspan=2>
      <div class="slidecontainer">
                <input type="range" min="0" max="180" value="90" class="slider" id="Tilt" oninput='sendButtonInput("Tilt",value)'>
      </div>
     </td>
    </tr>
   </table>

   <script>
    var webSocketCameraUrl = "ws:\/\/" + window.location.hostname + "/Camera";
    var webSocketCarInputUrl = "ws:\/\/" + window.location.hostname + "/CarInput";
    var websocketCamera;
    var websocketCarInput;

    function initCameraWebSocket()
    {
     websocketCamera = new WebSocket(webSocketCameraUrl);
     websocketCamera.binaryType = 'blob';
     websocketCamera.onopen    = function(event){};
     websocketCamera.onclose   = function(event){setTimeout(initCameraWebSocket, 2000);};
     websocketCamera.onmessage = function(event)
     {
      var imageId = document.getElementById("cameraImage");
```

```
      imageId.src = URL.createObjectURL(event.data);
     };
    }

    function initCarInputWebSocket()
    {
     websocketCarInput = new WebSocket(webSocketCarInputUrl);
     websocketCarInput.onopen    = function(event)
     {
      sendButtonInput("Speed", document.getElementById("Speed").value);
      sendButtonInput("Light", document.getElementById("Light").value);
      sendButtonInput("Pan", document.getElementById("Pan").value);
      sendButtonInput("Tilt", document.getElementById("Tilt").value);
     };
          websocketCarInput.onclose      =  function(event){setTimeout(initCarInputWebSocket,
2000);};
     websocketCarInput.onmessage = function(event){};
    }

    function initWebSocket()
    {
     initCameraWebSocket ();
     initCarInputWebSocket();
    }

    function sendButtonInput(key, value)
    {
     var data = key + "," + value;
     websocketCarInput.send(data);
    }

    window.onload = initWebSocket;
    document.getElementById("mainTable").addEventListener("touchend", function(event){
     event.preventDefault()
    });
   </script>
  </body>
</html>
```

```
)HTMLHOMEPAGE";


void rotateMotor(int motorNumber, int motorDirection)
{
 if (motorDirection == FORWARD)
 {
  digitalWrite(motorPins[motorNumber].pinIN1, HIGH);
  digitalWrite(motorPins[motorNumber].pinIN2, LOW);
 }
 else if (motorDirection == BACKWARD)
 {
  digitalWrite(motorPins[motorNumber].pinIN1, LOW);
  digitalWrite(motorPins[motorNumber].pinIN2, HIGH);
 }
 else
 {
  digitalWrite(motorPins[motorNumber].pinIN1, LOW);
  digitalWrite(motorPins[motorNumber].pinIN2, LOW);
 }
}

void moveCar(int inputValue)
{
 Serial.printf("Got value as %d\n", inputValue);
 switch(inputValue)
 {

  case UP:
   rotateMotor(RIGHT_MOTOR, FORWARD);
   rotateMotor(LEFT_MOTOR, FORWARD);
   break;

  case DOWN:
   rotateMotor(RIGHT_MOTOR, BACKWARD);
   rotateMotor(LEFT_MOTOR, BACKWARD);
   break;
```

```
    case LEFT:
      rotateMotor(RIGHT_MOTOR, FORWARD);
      rotateMotor(LEFT_MOTOR, BACKWARD);
      break;

    case RIGHT:
      rotateMotor(RIGHT_MOTOR, BACKWARD);
      rotateMotor(LEFT_MOTOR, FORWARD);
      break;

    case STOP:
      rotateMotor(RIGHT_MOTOR, STOP);
      rotateMotor(LEFT_MOTOR, STOP);
      break;

    default:
      rotateMotor(RIGHT_MOTOR, STOP);
      rotateMotor(LEFT_MOTOR, STOP);
      break;
  }
}

void handleRoot(AsyncWebServerRequest *request)
{
  request->send_P(200, "text/html", htmlHomePage);
}

void handleNotFound(AsyncWebServerRequest *request)
{
    request->send(404, "text/plain", "File Not Found");
}

void onCarInputWebSocketEvent(AsyncWebSocket *server,
                              AsyncWebSocketClient *client,
                              AwsEventType type,
                              void *arg,
                              uint8_t *data,
                              size_t len)
```

```
{
  switch (type)
  {
    case WS_EVT_CONNECT:
              Serial.printf("WebSocket client #%u connected from %s\n", client->id(), client->remoteIP().toString().c_str());
      break;
    case WS_EVT_DISCONNECT:
      Serial.printf("WebSocket client #%u disconnected\n", client->id());
      moveCar(0);
      ledcWrite(PWMLightChannel, 0);
      panServo.write(90);
      tiltServo.write(90);
      break;
    case WS_EVT_DATA:
      AwsFrameInfo *info;
      info = (AwsFrameInfo*)arg;
      if (info->final && info->index == 0 && info->len == len && info->opcode == WS_TEXT)
      {
        std::string myData = "";
        myData.assign((char *)data, len);
        std::istringstream ss(myData);
        std::string key, value;
        std::getline(ss, key, ',');
        std::getline(ss, value, ',');
        Serial.printf("Key [%s] Value[%s]\n", key.c_str(), value.c_str());
        int valueInt = atoi(value.c_str());
        if (key == "MoveCar")
        {
          moveCar(valueInt);
        }
        else if (key == "Speed")
        {
          ledcWrite(PWMSpeedChannel, valueInt);
        }
        else if (key == "Light")
        {
          ledcWrite(PWMLightChannel, valueInt);
```

```
      }
      else if (key == "Pan")
      {
        panServo.write(valueInt);
      }
      else if (key == "Tilt")
      {
        tiltServo.write(valueInt);
      }
    }
    break;
  case WS_EVT_PONG:
  case WS_EVT_ERROR:
    break;
  default:
    break;
 }
}

void onCameraWebSocketEvent(AsyncWebSocket *server,
                AsyncWebSocketClient *client,
                AwsEventType type,
                void *arg,
                uint8_t *data,
                size_t len)
{
 switch (type)
 {
  case WS_EVT_CONNECT:
            Serial.printf("WebSocket client #%u connected from %s\n", client->id(), client->remoteIP().toString().c_str());
    cameraClientId = client->id();
    break;
  case WS_EVT_DISCONNECT:
    Serial.printf("WebSocket client #%u disconnected\n", client->id());
    cameraClientId = 0;
    break;
  case WS_EVT_DATA:
```

```
    break;
   case WS_EVT_PONG:
   case WS_EVT_ERROR:
    break;
   default:
    break;
  }
}

void setupCamera()
{
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_4;
  config.ledc_timer = LEDC_TIMER_2;
  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;

  config.frame_size = FRAMESIZE_VGA;
  config.jpeg_quality = 10;
  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;
 }

 if (psramFound())
 {
  heap_caps_malloc_extmem_enable(20000);
  Serial.printf("PSRAM initialized. malloc to take memory from psram above this size");
 }
}

void sendCameraPicture()
{
 if (cameraClientId == 0)
 {
  return;
 }
 unsigned long  startTime1 = millis();
 //capture a frame
 camera_fb_t * fb = esp_camera_fb_get();
 if (!fb)
 {
    Serial.println("Frame buffer could not be acquired");
    return;
 }

 unsigned long  startTime2 = millis();
 wsCamera.binary(cameraClientId, fb->buf, fb->len);
 esp_camera_fb_return(fb);

 //Wait for message to be delivered
 while (true)
 {
  AsyncWebSocketClient * clientPointer = wsCamera.client(cameraClientId);
  if (!clientPointer || !(clientPointer->queueIsFull()))
```

```
   {
     break;
   }
   delay(1);
  }

  unsigned long  startTime3 = millis();
     Serial.printf("Time taken Total: %d|%d|%d\n",startTime3 - startTime1, startTime2 - startTime1, startTime3-startTime2 );
}

void setUpPinModes()
{
  panServo.attach(PAN_PIN);
  tiltServo.attach(TILT_PIN);

  //Set up PWM
  ledcSetup(PWMSpeedChannel, PWMFreq, PWMResolution);
  ledcSetup(PWMLightChannel, PWMFreq, PWMResolution);

  for (int i = 0; i < motorPins.size(); i++)
  {
    pinMode(motorPins[i].pinEn, OUTPUT);
    pinMode(motorPins[i].pinIN1, OUTPUT);
    pinMode(motorPins[i].pinIN2, OUTPUT);
    /* Attach the PWM Channel to the motor enb Pin */
    ledcAttachPin(motorPins[i].pinEn, PWMSpeedChannel);
  }
  moveCar(STOP);

  pinMode(LIGHT_PIN, OUTPUT);
  ledcAttachPin(LIGHT_PIN, PWMLightChannel);
}


void setup(void)
{
  setUpPinModes();
```

```
  //Serial.begin(115200);

  WiFi.softAP(ssid, password);
  IPAddress IP = WiFi.softAPIP();
  Serial.print("AP IP address: ");
  Serial.println(IP);

  server.on("/", HTTP_GET, handleRoot);
  server.onNotFound(handleNotFound);

  wsCamera.onEvent(onCameraWebSocketEvent);
  server.addHandler(&wsCamera);

  wsCarInput.onEvent(onCarInputWebSocketEvent);
  server.addHandler(&wsCarInput);

  server.begin();
  Serial.println("HTTP server started");

  setupCamera();
}


void loop()
{
  wsCamera.cleanupClients();
  wsCarInput.cleanupClients();
  sendCameraPicture();
     Serial.printf("SPIRam Total heap %d, SPIRam Free Heap %d\n", ESP.getPsramSize(),
ESP.getFreePsram());
}
```