#include "esp\_camera.h"

#include <Arduino.h>

#include <WiFi.h>

#include <AsyncTCP.h>

#include <ESPAsyncWebServer.h>

#include <iostream>

#include <sstream>

struct MOTOR\_PINS

{

int pinEn;

int pinIN1;

int pinIN2;

};

std::vector<MOTOR\_PINS> motorPins =

{

{12, 13, 15}, //RIGHT\_MOTOR Pins (EnA, IN1, IN2)

{12, 14, 2}, //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:40px;

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">

<!--h2 style="color: teal;text-align:center;">Wi-Fi Camera &#128663; Control</h2-->

<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>

</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)

{

var speedButton = document.getElementById("Speed");

sendButtonInput("Speed", speedButton.value);

var lightButton = document.getElementById("Light");

sendButtonInput("Light", lightButton.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);

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);

}

}

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\_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;

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()

{

//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());

}