```
/*
  ESP32-CAM (AI Thinker) - Face Tracking Sender
  - Streams MJPEG over Wi-Fi
  - Runs lightweight face detection at QQVGA (fast, ugly)
  - Sends ex/ey/found to ESP32-WROOM via ESP-NOW

  Arduino core: ESP32 by Espressif Systems 1.0.6
*/

#include <WiFi.h>
#include <esp_now.h>
#include <WebServer.h>
#include <esp_wifi.h>

#include "esp_camera.h"
#include "img_converters.h"
#include "fd_forward.h"

// ========= 1) Hotspot (2.4 GHz) =========
const char* ssid     = "YOUR_HOTSPOT_NAME";
const char* password = "YOUR_HOTSPOT_PASSWORD";

// ========= 2) WROOM Receiver MAC =========
uint8_t receiverMac[] = { 0xA0, 0xDD, 0x6C, 0x0C, 0x13, 0x6C };

// ========= 3) Speed / tracking tuning =========
#define STREAM_FRAMESIZE   FRAMESIZE_QQVGA  // 160x120 FAST
#define JPEG_QUALITY_NUM   22               // higher = worse quality; 18-28 is good range
int DETECT_EVERY_MS = 250;                  // face detect rate (200–400)
int SEND_EVERY_MS   = 50;                   // ESP-NOW send rate (20 Hz)
int deadband        = 6;                    // pixels (reduce jitter)

// Face detector config
static mtmn_config_t mtmn_config = mtmn_init_config();

// Web server
WebServer server(80);

// Shared state
volatile int g_found = 0;
volatile int g_ex = 0;
volatile int g_ey = 0;

unsigned long lastDetectMs = 0;
```

```
unsigned long lastSendMs   = 0;

// ESP-NOW packet
typedef struct __attribute__((packed)) {
  int16_t ex;
  int16_t ey;
  uint8_t found;
} TrackPacket;

// ========= AI Thinker ESP32-CAM pin map =========
#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

// ----- Helpers -----
uint8_t getWiFiChannel() {
  uint8_t primary = 0;
  wifi_second_chan_t second;
  esp_wifi_get_channel(&primary, &second);
  return primary;
}

void initEspNow() {
  if (esp_now_init() != ESP_OK) {
    Serial.println("ESP-NOW init failed");
    while (true) delay(1000);
  }

  esp_now_peer_info_t peerInfo = {};
  memcpy(peerInfo.peer_addr, receiverMac, 6);
  peerInfo.channel = getWiFiChannel();   // <-- important for reliability
```

```
  peerInfo.encrypt = false;

  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Add peer failed");
    while (true) delay(1000);
  }

  Serial.print("ESP-NOW peer added on channel ");
  Serial.println(peerInfo.channel);
}

bool initCamera() {
  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;

  // core 1.0.6 uses pin_sscb_*
  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;

  // Fast streaming: JPEG direct
  config.pixel_format = PIXFORMAT_JPEG;
  config.frame_size   = STREAM_FRAMESIZE;
  config.jpeg_quality = JPEG_QUALITY_NUM;
```

```
  // Use 2 frame buffers if PSRAM exists; otherwise 1
  config.fb_count = psramFound() ? 2 : 1;

  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed: 0x%x\n", err);
    return false;
  }

  sensor_t* s = esp_camera_sensor_get();
  s->set_framesize(s, STREAM_FRAMESIZE);
  s->set_quality(s, JPEG_QUALITY_NUM);

  return true;
}

void sendTrackIfDue() {
  unsigned long now = millis();
  if (now - lastSendMs < (unsigned long)SEND_EVERY_MS) return;
  lastSendMs = now;

  TrackPacket pkt;
  pkt.ex = (int16_t)g_ex;
  pkt.ey = (int16_t)g_ey;
  pkt.found = g_found ? 1 : 0;

  esp_now_send(receiverMac, (uint8_t*)&pkt, sizeof(pkt));
}

void runFaceDetectIfDue(const uint8_t* jpg, size_t jpg_len, int w, int h) {
  unsigned long now = millis();
  if (now - lastDetectMs < (unsigned long)DETECT_EVERY_MS) return;
  lastDetectMs = now;

  // Decode JPEG -> RGB888 (heavy, but ok at QQVGA)
  dl_matrix3du_t* rgb = dl_matrix3du_alloc(1, w, h, 3);
  if (!rgb) return;

  bool ok = fmt2rgb888(jpg, jpg_len, PIXFORMAT_JPEG, rgb->item);
  if (!ok) {
    dl_matrix3du_free(rgb);
    return;
  }
```

```
  box_array_t* faces = face_detect(rgb, &mtmn_config);

  int found = (faces && faces->len > 0) ? 1 : 0;
  int ex = 0, ey = 0;

  if (found) {
    int x1 = faces->box[0].box_p[0];
    int y1 = faces->box[0].box_p[1];
    int x2 = faces->box[0].box_p[2];
    int y2 = faces->box[0].box_p[3];

    int fx = (x1 + x2) / 2;
    int fy = (y1 + y2) / 2;

    int cx = w / 2;
    int cy = h / 2;

    ex = fx - cx;
    ey = fy - cy;

    if (abs(ex) < deadband) ex = 0;
    if (abs(ey) < deadband) ey = 0;
  }

  g_found = found;
  g_ex = ex;
  g_ey = ey;

  if (faces) {
    free(faces->score);
    free(faces->box);
    free(faces->landmark);
    free(faces);
  }
  dl_matrix3du_free(rgb);
}

// ----- Web -----
void handleRoot() {
  const char HTML[] PROGMEM = R"HTML(
<!doctype html><html><head>
<meta name="viewport" content="width=device-width, initial-scale=1"/>
<title>ESP32-CAM Face Turret</title>
<style>
```

```
body{font-family:Arial;margin:16px}
img{width:100%;max-width:520px;border:1px solid #ccc}
pre{background:#f5f5f5;padding:10px;max-width:520px}
</style></head><body>
<h2>ESP32-CAM Face Tracking Stream</h2>
<p>Stream: /stream    Status: /status</p>
<img src="/stream">
<pre id="st">...</pre>
<script>
async function tick(){
  const j = await (await fetch('/status')).json();
  document.getElementById('st').textContent = JSON.stringify(j,null,2);
}
setInterval(tick, 200); tick();
</script>
</body></html>
)HTML";
  server.send(200, "text/html", HTML);
}

void handleStatus() {
  String json="{";
  json += "\"found\":" + String(g_found) + ",";
  json += "\"ex\":" + String(g_ex) + ",";
  json += "\"ey\":" + String(g_ey) + ",";
  json += "\"detect_ms\":" + String(DETECT_EVERY_MS) + ",";
  json += "\"send_ms\":" + String(SEND_EVERY_MS) + ",";
  json += "\"quality\":" + String(JPEG_QUALITY_NUM) + ",";
  json += "\"channel\":" + String(getWiFiChannel());
  json += "}";
  server.send(200, "application/json", json);
}

void handleStream() {
  WiFiClient client = server.client();
  String resp = "HTTP/1.1 200 OK\r\n";
  resp += "Content-Type: multipart/x-mixed-replace; boundary=frame\r\n\r\n";
  client.print(resp);

  while (client.connected()) {
    camera_fb_t* fb = esp_camera_fb_get();
    if (!fb) continue;

    runFaceDetectIfDue(fb->buf, fb->len, fb->width, fb->height);
```

```
    sendTrackIfDue();

    client.print("--frame\r\n");
    client.print("Content-Type: image/jpeg\r\n");
    client.print("Content-Length: " + String(fb->len) + "\r\n\r\n");
    client.write(fb->buf, fb->len);
    client.print("\r\n");

    esp_camera_fb_return(fb);

    delay(10); // small throttle
  }
}

// ----- Setup/Loop -----
void setup() {
  Serial.begin(115200);
  delay(200);

  WiFi.mode(WIFI_STA);
  WiFi.setSleep(false);
  WiFi.begin(ssid, password);

  Serial.print("Connecting");
  while (WiFi.status() != WL_CONNECTED) { delay(300); Serial.print("."); }
  Serial.println();

  Serial.print("IP: "); Serial.println(WiFi.localIP());
  Serial.print("CAM MAC: "); Serial.println(WiFi.macAddress());
  Serial.print("Wi-Fi Channel: "); Serial.println(getWiFiChannel());

  // Critical for QQVGA detection: allow smaller faces
  mtmn_config.min_face = 40;

  initEspNow();

  if (!initCamera()) {
    Serial.println("Camera init failed");
    while (true) delay(1000);
  }

  server.on("/", handleRoot);
  server.on("/status", handleStatus);
  server.on("/stream", HTTP_GET, handleStream);
```

```
  server.begin();

  Serial.println("Open http://<IP>/");
}

void loop() {
  server.handleClient();
  // Keep sending even if nobody is watching stream
  sendTrackIfDue();
}
```