1) 安装依赖（树莓派上执行）

1.1 基础编译环境 + OpenCV4

```
sudo apt update
sudo apt install -y build-essential cmake pkg-config git curl
sudo apt install -y libopencv-dev
```

1.2 验证 OpenCV4 安装成功

```
pkg-config --modversion opencv4
pkg-config --cflags --libs opencv4
```

能输出版本号和一串编译/链接参数，就 OK。

> 如果你后面遇到 CSI 相机（libcamera）打不开的情况，我在第 8 节给你两种解决路线（含 GStreamer）。

2) 创建工程目录（固定到 ~/Project/CProject)

我们把项目放到：

```
~/Project/CProject/rpi_vision_cpp/
```

2.1 创建目录结构

```
mkdir -p ~/Project/CProject/rpi_vision_cpp/third_party
cd ~/Project/CProject/rpi_vision_cpp
```

3) 下载 header-only 依赖（不使用 Python）

我们用两个“单头文件库”：

- **cpp-httplib**：HTTP server（网页+API+MJPEG）
- **nlohmann/json**：JSON 解析

3.1 下载到 third_party

```
cd ~/Project/CProject/rpi_vision_cpp/third_party

# cpp-httplib
curl -L -o httplib.h \
  https://raw.githubusercontent.com/yhirose/cpp-httplib/master/httplib.h

# nlohmann/json (single header)
curl -L -o json.hpp \
  https://raw.githubusercontent.com/nlohmann/json/develop/single_include/nlohmann/json.hpp
```

3.2 检查文件是否存在

```
ls -lh ~/Project/CProject/rpi_vision_cpp/third_party
```

你应该看到 `httplib.h` 和 `json.hpp` 两个文件。

---

4) 写代码（main.cpp）

回到工程根目录：

```
cd ~/Project/CProject/rpi_vision_cpp
nano main.cpp
```

把下面的 **完整 C++ 代码**粘进去（保存退出：`Ctrl+O`、回车、`Ctrl+X`）。

> 说明：
>
> - 默认摄像头 `cam_id=0`，分辨率 640x480，fps=20
> - Web 端口 7860（如冲突可改）

```
#include <opencv2/opencv.hpp>
#include <atomic>
#include <mutex>
#include <thread>
#include <vector>
#include <string>
#include <sstream>
#include <chrono>

#include "third_party/httplib.h"
#include "third_party/json.hpp"

using json = nlohmann::json;

struct HSVRange {
    cv::Scalar lower; // H,S,V
    cv::Scalar upper;
```

```cpp
};

struct Config {
    // ROI / dominant
    bool draw_roi = true;
    bool compute_dominant = true;
    int dominant_k = 3;
    int dominant_every_n_frames = 6;
    bool has_roi = false;
    cv::Rect roi;

    // color alarm
    bool enable_color_alarm = false;
    int alarm_pixel_threshold = 3000;
    bool show_mask_preview = false;
    std::vector<HSVRange> hsv_ranges;

    // algorithms
    bool enable_blur = false;
    int blur_ksize = 5;

    bool enable_edges = false;
    int canny_t1 = 80;
    int canny_t2 = 160;
    std::string edges_mode = "overlay"; // overlay / only
    double edges_alpha = 0.6;
};

struct Meta {
    bool alarm = false;
    int alarm_pixels = 0;
    std::string dominant_hex;
    std::vector<int> dominant_rgb; // [R,G,B]
    bool has_roi = false;
    std::vector<int> roi; // x1,y1,x2,y2
};

static std::mutex g_mtx;
static Config g_cfg;
static Meta g_meta;

static std::vector<uchar> g_latest_jpeg;
static cv::Mat g_latest_raw_bgr; // for click sampling
static std::atomic<bool> g_running{true};
static std::atomic<int> g_frame_count{0};

static int clampi(int v, int lo, int hi) { return std::max(lo, std::min(hi, v)); }

static std::string rgb_to_hex(const cv::Vec3b& rgb) {
    char buf[8];
    std::snprintf(buf, sizeof(buf), "#%02X%02X%02X", rgb[0], rgb[1], rgb[2]);
    return std::string(buf);
}
```

```cpp
static std::vector<HSVRange> presets_to_ranges(const std::string& name) {
    if (name == "Red") {
        return {
            {cv::Scalar(0,120,70),  cv::Scalar(10,255,255)},
            {cv::Scalar(170,120,70),cv::Scalar(179,255,255)}
        };
    }
    if (name == "Yellow") return { {cv::Scalar(20,100,100), cv::Scalar(30,255,255)} };
    if (name == "Green")  return { {cv::Scalar(35,100,100), cv::Scalar(85,255,255)} };
    return {};
}

// Build HSV ranges from center HSV and tolerances; handle hue wrap.
static std::vector<HSVRange> build_hsv_ranges_wrap(int h, int s, int v, int htol, int stol,
int vtol) {
    int s1 = clampi(s - stol, 0, 255), s2 = clampi(s + stol, 0, 255);
    int v1 = clampi(v - vtol, 0, 255), v2 = clampi(v + vtol, 0, 255);

    int h1 = h - htol;
    int h2 = h + htol;

    std::vector<HSVRange> ranges;
    if (h1 < 0) {
        ranges.push_back({cv::Scalar(0, s1, v1), cv::Scalar(std::min(179, h2), s2, v2)});
        ranges.push_back({cv::Scalar(180 + h1, s1, v1), cv::Scalar(179, s2, v2)});
    } else if (h2 > 179) {
        ranges.push_back({cv::Scalar(std::max(0, h1), s1, v1), cv::Scalar(179, s2, v2)});
        ranges.push_back({cv::Scalar(0, s1, v1), cv::Scalar(h2 - 180, s2, v2)});
    } else {
        ranges.push_back({cv::Scalar(h1, s1, v1), cv::Scalar(h2, s2, v2)});
    }
    return ranges;
}

static cv::Mat build_mask(const cv::Mat& bgr, const std::vector<HSVRange>& ranges) {
    cv::Mat hsv;
    cv::cvtColor(bgr, hsv, cv::COLOR_BGR2HSV);
    cv::Mat mask, tmp;
    bool first = true;
    for (auto& r : ranges) {
        cv::inRange(hsv, r.lower, r.upper, tmp);
        if (first) { mask = tmp.clone(); first = false; }
        else cv::bitwise_or(mask, tmp, mask);
    }
    cv::medianBlur(mask, mask, 5);
    return mask;
}

// dominant color via kmeans on ROI (downscale)
static bool dominant_kmeans_rgb(const cv::Mat& roi_bgr, int k, cv::Vec3b& out_rgb) {
    if (roi_bgr.empty()) return false;
    cv::Mat small;
    cv::resize(roi_bgr, small, cv::Size(80, 80), 0, 0, cv::INTER_AREA);
    cv::Mat rgb;
    cv::cvtColor(small, rgb, cv::COLOR_BGR2RGB);

    cv::Mat samples = rgb.reshape(1, rgb.rows * rgb.cols);
    samples.convertTo(samples, CV_32F);

    cv::Mat labels, centers;
    cv::kmeans(samples, k, labels,
               cv::TermCriteria(cv::TermCriteria::EPS + cv::TermCriteria::MAX_ITER, 20,
1.0),
               2, cv::KMEANS_PP_CENTERS, centers);

    std::vector<int> cnt(k, 0);
    for (int i = 0; i < labels.rows; i++) cnt[labels.at<int>(i,0)]++;

    int best = 0;
    for (int i = 1; i < k; i++) if (cnt[i] > cnt[best]) best = i;

    cv::Vec3f c = centers.row(best);
    out_rgb = cv::Vec3b(
        (uchar)clampi((int)c[0],0,255),
        (uchar)clampi((int)c[1],0,255),
        (uchar)clampi((int)c[2],0,255)
    );
    return true;
}

static void apply_processing(const cv::Mat& frame_bgr, cv::Mat& out_bgr, Meta& meta, const
Config& cfg, int frame_count) {
    out_bgr = frame_bgr.clone();
    meta = Meta();

    const int h = frame_bgr.rows, w = frame_bgr.cols;

    // ROI
    cv::Rect roi_rect(0,0,w,h);
    if (cfg.has_roi) {
        roi_rect = cfg.roi & cv::Rect(0,0,w,h);
        meta.has_roi = true;
        meta.roi = {roi_rect.x, roi_rect.y, roi_rect.x + roi_rect.width, roi_rect.y +
roi_rect.height};
        if (cfg.draw_roi) {
            cv::rectangle(out_bgr, roi_rect, cv::Scalar(255,0,0), 2);
        }
    }

    // Color alarm on full image
    if (cfg.enable_color_alarm && !cfg.hsv_ranges.empty()) {
        cv::Mat mask = build_mask(frame_bgr, cfg.hsv_ranges);
        meta.alarm_pixels = cv::countNonZero(mask);
        meta.alarm = meta.alarm_pixels > cfg.alarm_pixel_threshold;
        if (cfg.show_mask_preview) {
            cv::cvtColor(mask, out_bgr, cv::COLOR_GRAY2BGR);
        }
    }

    // blur
    if (cfg.enable_blur) {
        int k = cfg.blur_ksize;
        if (k % 2 == 0) k += 1;
        k = clampi(k, 1, 31);
        cv::GaussianBlur(out_bgr, out_bgr, cv::Size(k,k), 0);
    }

    // edges
    if (cfg.enable_edges) {
        cv::Mat gray, edges;
        cv::cvtColor(out_bgr, gray, cv::COLOR_BGR2GRAY);
        cv::Canny(gray, edges, cfg.canny_t1, cfg.canny_t2);
        cv::Mat edges_bgr;
        cv::cvtColor(edges, edges_bgr, cv::COLOR_GRAY2BGR);

        if (cfg.edges_mode == "only") {
            out_bgr = edges_bgr;
        } else {
            double a = std::max(0.0, std::min(1.0, cfg.edges_alpha));
            cv::addWeighted(out_bgr, 1.0 - a, edges_bgr, a, 0, out_bgr);
        }
    }

    // dominant color (on ROI)
    if (cfg.compute_dominant && (frame_count % std::max(1, cfg.dominant_every_n_frames) ==
0)) {
        cv::Mat roi_bgr = frame_bgr(roi_rect).clone();
        cv::Vec3b dom_rgb;
        if (dominant_kmeans_rgb(roi_bgr, std::max(1, cfg.dominant_k), dom_rgb)) {
            meta.dominant_rgb = {(int)dom_rgb[0], (int)dom_rgb[1], (int)dom_rgb[2]};
            meta.dominant_hex = rgb_to_hex(dom_rgb);
        }
    }

    // overlays
    int ytxt = 28;
    if (!meta.dominant_hex.empty()) {
        cv::putText(out_bgr, "DOM " + meta.dominant_hex, {10,ytxt},
                    cv::FONT_HERSHEY_SIMPLEX, 0.8, {0,255,255}, 2);
        ytxt += 28;
    }
    if (meta.alarm) {
        cv::putText(out_bgr, "ALARM pixels=" + std::to_string(meta.alarm_pixels), {10,ytxt},
                    cv::FONT_HERSHEY_SIMPLEX, 0.8, {0,0,255}, 2);
    }
}
```

```cpp
static void camera_loop(int cam_id, int width, int height, int fps) {
    cv::VideoCapture cap(cam_id);
    cap.set(cv::CAP_PROP_FRAME_WIDTH, width);
    cap.set(cv::CAP_PROP_FRAME_HEIGHT, height);

    if (!cap.isOpened()) {
        std::lock_guard<std::mutex> lk(g_mtx);
        g_meta = Meta();
        return;
    }

    const int interval_ms = (int)(1000.0 / std::max(1, fps));

    while (g_running.load()) {
        auto t0 = std::chrono::steady_clock::now();

        cv::Mat frame;
        if (!cap.read(frame) || frame.empty()) continue;

        int fc = ++g_frame_count;

        Config cfg;
        {
            std::lock_guard<std::mutex> lk(g_mtx);
            cfg = g_cfg;
            g_latest_raw_bgr = frame.clone();
        }

        cv::Mat out;
        Meta meta;
        apply_processing(frame, out, meta, cfg, fc);

        {
            std::lock_guard<std::mutex> lk(g_mtx);
            // 如果本帧没重算主色，就沿用上一帧的主色
            if (meta.dominant_hex.empty()) {
                meta.dominant_hex = g_meta.dominant_hex;
                meta.dominant_rgb = g_meta.dominant_rgb;
            }
            g_meta = meta;

            std::vector<int> params = {cv::IMWRITE_JPEG_QUALITY, 80};
            g_latest_jpeg.clear();
            cv::imencode(".jpg", out, g_latest_jpeg, params);
        }

        auto t1 = std::chrono::steady_clock::now();
        auto used = std::chrono::duration_cast<std::chrono::milliseconds>(t1 - t0).count();
        int sleep_ms = interval_ms - (int)used;
        if (sleep_ms > 0) std::this_thread::sleep_for(std::chrono::milliseconds(sleep_ms));
    }
    cap.release();
}

static const char* kIndexHtml = R"HTML(
<!doctype html>
<html>
<head>
<meta charset="utf-8"/>
<meta name="viewport" content="width=device-width, initial-scale=1"/>
<title>RPI Vision Console (C++/OpenCV)</title>
<style>
body{font-family:system-ui,Segoe UI,Arial;margin:16px;background:#0b0f14;color:#e8eef6}
.card{background:#121a24;border:1px solid #1f2a3a;border-radius:16px;padding:14px;margin-
bottom:12px}
.row{display:flex;gap:12px;flex-wrap:wrap}
.col{flex:1;min-width:280px}
label{display:block;margin:8px 0 4px;font-size:13px;color:#b9c5d6}
input,select,button{width:100%;padding:10px;border-radius:12px;border:1px solid
#223145;background:#0c121b;color:#e8eef6}
button{cursor:pointer}
small{color:#9fb1c8}
.badge{display:inline-block;padding:6px 10px;border-
radius:999px;background:#0c121b;border:1px solid #223145}
#swatch{height:36px;border-radius:12px;border:1px solid #223145}
img{width:100%;border-radius:16px;border:1px solid #223145;cursor:crosshair}
hr{border:none;border-top:1px solid #223145;margin:12px 0}
</style>
</head>
<body>
  <h2>🤖 树莓派视觉控制台（C++/OpenCV，无Python）</h2>

  <div class="row">
    <div class="col card">
      <div class="row" style="align-items:center;justify-content:space-between">
        <div class="badge" id="status">状态：--</div>
        <div class="badge" id="roi">ROI：--</div>
      </div>
      <div class="row" style="margin-top:10px;align-items:center">
        <div style="flex:1">
          <div class="badge">DOM: <span id="domhex">--</span></div>
        </div>
        <div style="width:140px">
          <div id="swatch"></div>
        </div>
      </div>
      <hr/>
      <img id="stream" src="/stream.mjpg" alt="stream"/>
      <small>提示：点击画面可“取色/框选ROI”（由下方模式决定）</small>
    </div>

    <div class="col">
      <div class="card">
        <h3 style="margin:0 0 8px">🎯 交互模式</h3>
        <label>点击画面动作</label>
        <select id="clickMode">
          <option value="None">None</option>
          <option value="PickColor">点击取色</option>
          <option value="SetROI">两次点击框选ROI</option>
        </select>

        <label>HSV 容差（用于取色/调色板）</label>
        <div class="row">
          <div class="col">
            <input id="hTol" type="number" value="10" min="0" max="60"/>
            <small>H tol</small>
          </div>
          <div class="col">
            <input id="sTol" type="number" value="80" min="0" max="255"/>
            <small>S tol</small>
          </div>
          <div class="col">
            <input id="vTol" type="number" value="80" min="0" max="255"/>
            <small>V tol</small>
          </div>
        </div>
      </div>

      <div class="card">
        <h3 style="margin:0 0 8px">🎨 颜色目标</h3>
        <label>预置</label>
        <select id="preset">
          <option value="None">None</option>
          <option value="Red">Red</option>
          <option value="Yellow">Yellow</option>
          <option value="Green">Green</option>
        </select>
        <button onclick="applyPreset()">应用预置</button>

        <label>调色板</label>
        <input id="picker" type="color" value="#ff0000"/>
        <button onclick="applyPalette()">应用调色板为目标</button>
        <button onclick="clearTarget()">清除目标（关闭报警）</button>
      </div>

      <div class="card">
        <h3 style="margin:0 0 8px">🧪 算法开关</h3>
        <label><input id="enAlarm" type="checkbox"/> 启用颜色报警</label>
        <label>报警像素阈值</label>
        <input id="pixTh" type="number" value="3000" min="0" max="50000"/>

        <label><input id="maskPreview" type="checkbox"/> 显示mask预览（替换输出）</label>
        <label><input id="drawRoi" type="checkbox" checked/> 画ROI框</label>
        <label><input id="domOn" type="checkbox" checked/> 计算ROI主色</label>

        <hr/>
        <label><input id="enEdges" type="checkbox"/> 边缘检测</label>
```

```
        <label>边缘显示方式</label>
        <select id="edgesMode">
          <option value="overlay">overlay</option>
          <option value="only">only</option>
        </select>
        <div class="row">
          <div class="col"><input id="c1" type="number" value="80" min="0" max="500"/>
<small>Canny t1</small></div>
          <div class="col"><input id="c2" type="number" value="160" min="0" max="500"/>
<small>Canny t2</small></div>
          <div class="col"><input id="ea" type="number" value="0.6" step="0.05" min="0"
max="1"/><small>alpha</small></div>
        </div>

        <hr/>
        <label><input id="enBlur" type="checkbox"/> 模糊</label>
        <label>模糊核(奇数)</label>
        <input id="bk" type="number" value="5" min="1" max="31"/>

        <hr/>
        <button onclick="applyAlgo()">下发算法配置</button>
      </div>
    </div>
  </div>

<script>
async function postJson(url, obj){
  const r = await fetch(url, {method:'POST', headers:{'Content-Type':'application/json'},
body:JSON.stringify(obj)});
  return await r.json().catch(()=> ({}));
}

async function applyPreset(){
  const p = document.getElementById('preset').value;
  await postJson('/config', {preset:p});
}

async function applyPalette(){
  const hex = document.getElementById('picker').value;
  const hTol = parseInt(document.getElementById('hTol').value||10);
  const sTol = parseInt(document.getElementById('sTol').value||80);
  const vTol = parseInt(document.getElementById('vTol').value||80);
  await postJson('/config', {palette_hex:hex, hTol, sTol, vTol});
}

async function clearTarget(){
  await postJson('/config', {clear_target:true});
}

async function applyAlgo(){
  const cfg = {
    enable_color_alarm: document.getElementById('enAlarm').checked,
    alarm_pixel_threshold: parseInt(document.getElementById('pixTh').value||3000),
    show_mask_preview: document.getElementById('maskPreview').checked,
    draw_roi: document.getElementById('drawRoi').checked,
    compute_dominant: document.getElementById('domOn').checked,

    enable_edges: document.getElementById('enEdges').checked,
    edges_mode: document.getElementById('edgesMode').value,
    canny_t1: parseInt(document.getElementById('c1').value||80),
    canny_t2: parseInt(document.getElementById('c2').value||160),
    edges_alpha: parseFloat(document.getElementById('ea').value||0.6),

    enable_blur: document.getElementById('enBlur').checked,
    blur_ksize: parseInt(document.getElementById('bk').value||5),
  };
  await postJson('/config', cfg);
}

document.getElementById('stream').addEventListener('click', async (ev)=>{
  const img = ev.target;
  const rect = img.getBoundingClientRect();
  const x = Math.floor((ev.clientX - rect.left) * (img.naturalWidth / rect.width));
  const y = Math.floor((ev.clientY - rect.top)  * (img.naturalHeight / rect.height));
  const mode = document.getElementById('clickMode').value;
  const hTol = parseInt(document.getElementById('hTol').value||10);
  const sTol = parseInt(document.getElementById('sTol').value||80);
  const vTol = parseInt(document.getElementById('vTol').value||80);
  await postJson('/click', {mode, x, y, hTol, sTol, vTol});
});

async function refreshStatus(){
  const r = await fetch('/status');
  const s = await r.json().catch(()=> ({}));
  document.getElementById('status').textContent = "状态: " + (s.status || "--");
  document.getElementById('roi').textContent = "ROI: " + (s.roi || "--");
  document.getElementById('domhex').textContent = (s.dominant_hex || "--");
  const sw = document.getElementById('swatch');
  const hex = s.dominant_hex || "";
  sw.style.background = hex ? hex : "transparent";
  sw.textContent = hex ? hex : "";
}
setInterval(refreshStatus, 300);
</script>
</body>
</html>
)HTML";

static json meta_to_status_json(const Meta& meta) {
    json j;
    j["alarm"] = meta.alarm;
    j["alarm_pixels"] = meta.alarm_pixels;
    j["dominant_hex"] = meta.dominant_hex;
    if (!meta.dominant_rgb.empty()) j["dominant_rgb"] = meta.dominant_rgb;

    if (meta.has_roi && meta.roi.size()==4) {
        std::ostringstream oss;
        oss << "[" << meta.roi[0] << "," << meta.roi[1] << "," << meta.roi[2] << "," <<
meta.roi[3] << "]";
        j["roi"] = oss.str();
    } else j["roi"] = "未设置";

    j["status"] = meta.alarm ? ("🚨 报警 pixels=" + std::to_string(meta.alarm_pixels)) : "✅
正常";
    return j;
}

int main() {
    // 默认配置
    {
        std::lock_guard<std::mutex> lk(g_mtx);
        g_cfg = Config();
    }

    // 启动相机线程
    std::thread th(camera_loop, 0, 640, 480, 20);

    httplib::Server svr;

    // 首页
    svr.Get("/", [](const httplib::Request&, httplib::Response& res){
        res.set_content(kIndexHtml, "text/html; charset=utf-8");
    });

    // 状态
    svr.Get("/status", [](const httplib::Request&, httplib::Response& res){
        Meta meta;
        {
            std::lock_guard<std::mutex> lk(g_mtx);
            meta = g_meta;
        }
        res.set_content(meta_to_status_json(meta).dump(), "application/json; charset=utf-
8");
    });

    // 配置
    svr.Post("/config", [](const httplib::Request& req, httplib::Response& res){
        json out;
        try {
            json j = json::parse(req.body);

            std::lock_guard<std::mutex> lk(g_mtx);

            if (j.contains("preset")) {
                std::string p = j["preset"].get<std::string>();
                if (p == "None") {
                    g_cfg.hsv_ranges.clear();
                    g_cfg.enable_color_alarm = false;
                } else {
```

```
                g_cfg.hsv_ranges = presets_to_ranges(p);
                g_cfg.enable_color_alarm = true;
            }
        }

        if (j.contains("palette_hex")) {
            std::string hex = j["palette_hex"].get<std::string>();
            int hTol = j.value("hTol", 10);
            int sTol = j.value("sTol", 80);
            int vTol = j.value("vTol", 80);

            int r = std::stoi(hex.substr(1,2), nullptr, 16);
            int g = std::stoi(hex.substr(3,2), nullptr, 16);
            int b = std::stoi(hex.substr(5,2), nullptr, 16);
            cv::Mat bgr(1,1,CV_8UC3, cv::Scalar(b,g,r));
            cv::Mat hsv;
            cv::cvtColor(bgr, hsv, cv::COLOR_BGR2HSV);
            auto hv = hsv.at<cv::Vec3b>(0,0);

            g_cfg.hsv_ranges = build_hsv_ranges_wrap(hv[0], hv[1], hv[2], hTol, sTol,
vTol);

            g_cfg.enable_color_alarm = true;
        }

        if (j.value("clear_target", false)) {
            g_cfg.hsv_ranges.clear();
            g_cfg.enable_color_alarm = false;
        }

        auto setb = [&](const char* key, bool& ref){
            if (j.contains(key)) ref = j[key].get<bool>();
        };
        auto seti = [&](const char* key, int& ref){
            if (j.contains(key)) ref = j[key].get<int>();
        };
        auto setd = [&](const char* key, double& ref){
            if (j.contains(key)) ref = j[key].get<double>();
        };
        auto sets = [&](const char* key, std::string& ref){
            if (j.contains(key)) ref = j[key].get<std::string>();
        };

        setb("enable_color_alarm", g_cfg.enable_color_alarm);
        seti("alarm_pixel_threshold", g_cfg.alarm_pixel_threshold);
        setb("show_mask_preview", g_cfg.show_mask_preview);

        setb("enable_blur", g_cfg.enable_blur);
        seti("blur_ksize", g_cfg.blur_ksize);

        setb("enable_edges", g_cfg.enable_edges);
        seti("canny_t1", g_cfg.canny_t1);
        seti("canny_t2", g_cfg.canny_t2);
        sets("edges_mode", g_cfg.edges_mode);
```

```
            setd("edges_alpha", g_cfg.edges_alpha);

            setb("draw_roi", g_cfg.draw_roi);
            setb("compute_dominant", g_cfg.compute_dominant);

            out["ok"] = true;
        } catch (...) {
            out["ok"] = false;
        }
        res.set_content(out.dump(), "application/json; charset=utf-8");
    });

    // click: PickColor / SetROI
    svr.Post("/click", [](const httplib::Request& req, httplib::Response& res){
        json out;
        try {
            json j = json::parse(req.body);
            std::string mode = j.value("mode", "None");
            int x = j.value("x", 0);
            int y = j.value("y", 0);
            int hTol = j.value("hTol", 10);
            int sTol = j.value("sTol", 80);
            int vTol = j.value("vTol", 80);

            std::lock_guard<std::mutex> lk(g_mtx);
            if (g_latest_raw_bgr.empty()) {
                out["ok"] = false; out["msg"] = "no frame";
                res.set_content(out.dump(), "application/json; charset=utf-8");
                return;
            }
            int W = g_latest_raw_bgr.cols, H = g_latest_raw_bgr.rows;
            x = clampi(x, 0, W-1);
            y = clampi(y, 0, H-1);

            if (mode == "PickColor") {
                cv::Vec3b bgr = g_latest_raw_bgr.at<cv::Vec3b>(y,x);
                cv::Mat bgr1(1,1,CV_8UC3, cv::Scalar(bgr[0], bgr[1], bgr[2]));
                cv::Mat hsv;
                cv::cvtColor(bgr1, hsv, cv::COLOR_BGR2HSV);
                auto hv = hsv.at<cv::Vec3b>(0,0);

                g_cfg.hsv_ranges = build_hsv_ranges_wrap(hv[0], hv[1], hv[2], hTol, sTol,
vTol);
                g_cfg.enable_color_alarm = true;
                out["ok"] = true;
                out["msg"] = "picked";
            } else if (mode == "SetROI") {
                static bool have_first = false;
                static cv::Point p1;
                if (!have_first) {
                    p1 = cv::Point(x,y);
                    have_first = true;
                    out["ok"] = true;
                    out["msg"] = "roi_p1_set";
                } else {
                    cv::Point p2(x,y);
                    int x1 = std::min(p1.x, p2.x), x2 = std::max(p1.x, p2.x);
                    int y1 = std::min(p1.y, p2.y), y2 = std::max(p1.y, p2.y);
                    if (x2 > x1 && y2 > y1) {
                        g_cfg.has_roi = true;
                        g_cfg.roi = cv::Rect(x1,y1, x2-x1, y2-y1);
                    }
                    have_first = false;
                    out["ok"] = true;
                    out["msg"] = "roi_set";
                }
            } else {
                out["ok"] = true;
                out["msg"] = "noop";
            }
        } catch (...) {
            out["ok"] = false;
        }
        res.set_content(out.dump(), "application/json; charset=utf-8");
    });

    // MJPEG stream
    svr.Get("/stream.mjpg", [](const httplib::Request&, httplib::Response& res){
        res.set_header("Cache-Control", "no-cache");
        res.set_header("Connection", "close");
        res.set_header("Pragma", "no-cache");
        res.set_content_provider(
            "multipart/x-mixed-replace; boundary=frame",
            [](size_t, httplib::DataSink& sink) {
                while (g_running.load()) {
                    std::vector<uchar> jpg;
                    {
                        std::lock_guard<std::mutex> lk(g_mtx);
                        jpg = g_latest_jpeg;
                    }
                    if (!jpg.empty()) {
                        std::ostringstream oss;
                        oss << "--frame\r\n"
                            << "Content-Type: image/jpeg\r\n"
                            << "Content-Length: " << jpg.size() << "\r\n\r\n";
                        auto hdr = oss.str();
                        sink.write(hdr.data(), hdr.size());
                        sink.write(reinterpret_cast<const char*>(jpg.data()), jpg.size());
                        sink.write("\r\n", 2);
                    }
                    std::this_thread::sleep_for(std::chrono::milliseconds(50));
                }
                return true;
            }
        );
    });
```

```
    const int port = 7860;
    std::printf("Open http://<raspberrypi-ip>:%d\n", port);
    svr.listen("0.0.0.0", port);

    g_running.store(false);
    if (th.joinable()) th.join();
    return 0;
}
```

---

5) 写 CMakeLists.txt

在同目录创建：

```
cd ~/Project/CProject/rpi_vision_cpp
nano CMakeLists.txt
```

粘贴：

```
cmake_minimum_required(VERSION 3.16)
project(rpi_vision_console)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

find_package(OpenCV REQUIRED)

add_executable(rpi_vision main.cpp)

target_include_directories(rpi_vision PRIVATE
  ${OpenCV_INCLUDE_DIRS}
  ${CMAKE_SOURCE_DIR}/third_party
)

target_link_libraries(rpi_vision PRIVATE
  ${OpenCV_LIBS}
  pthread
)
```

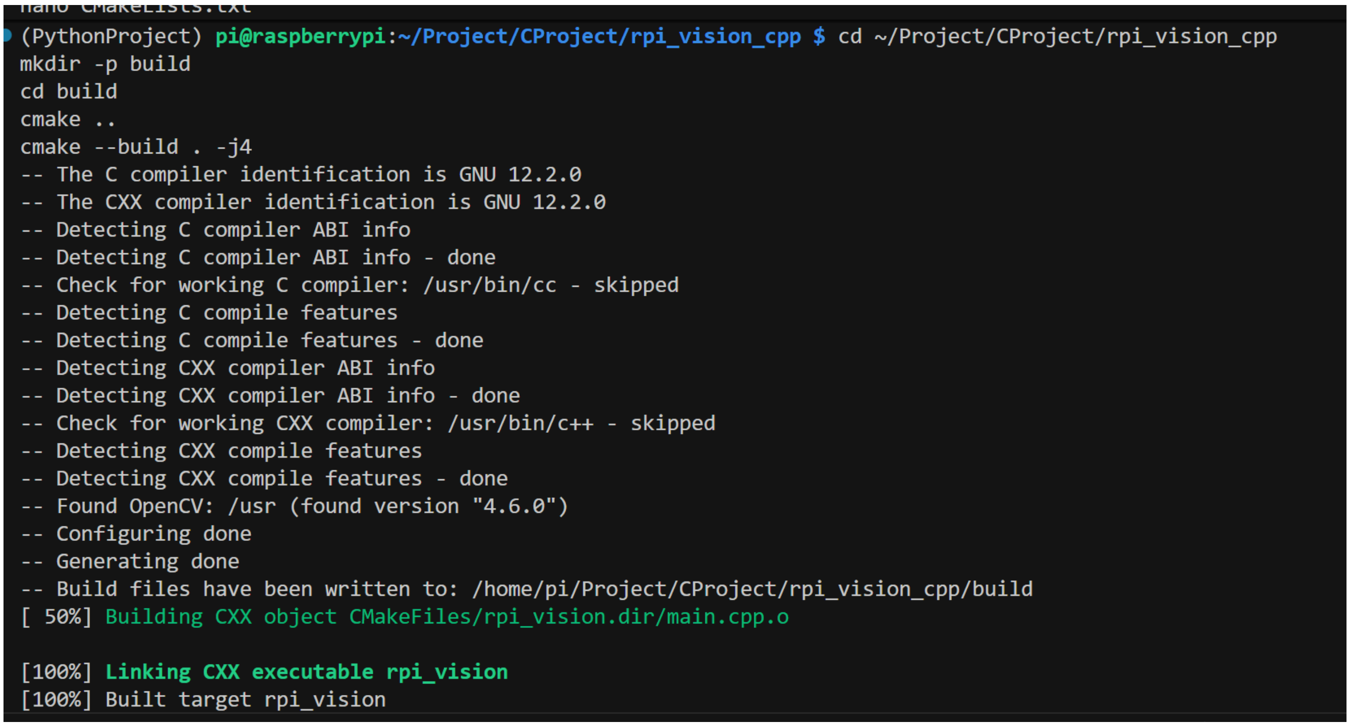
---

6) 编译与运行

6.1 编译

```
cd ~/Project/CProject/rpi_vision_cpp
mkdir -p build
cd build
cmake ..
cmake --build . -j4
```

成功后会生成：



```
nano CMakeLists.txt
(PythonProject) pi@raspberrypi:~/Project/CProject/rpi_vision_cpp $ cd ~/Project/CProject/rpi_vision_cpp
mkdir -p build
cd build
cmake ..
cmake --build . -j4
-- The C compiler identification is GNU 12.2.0
-- The CXX compiler identification is GNU 12.2.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found OpenCV: /usr (found version "4.6.0")
-- Configuring done
-- Generating done
-- Build files have been written to: /home/pi/Project/CProject/rpi_vision_cpp/build
[ 50%] Building CXX object CMakeFiles/rpi_vision.dir/main.cpp.o

[100%] Linking CXX executable rpi_vision
[100%] Built target rpi_vision
```

`~/Project/CProject/rpi_vision_cpp/build/rpi_vision`

6.2 运行

```
./rpi_vision
```

6.3 获取树莓派 IP

开另一个终端：

```
hostname -I
```

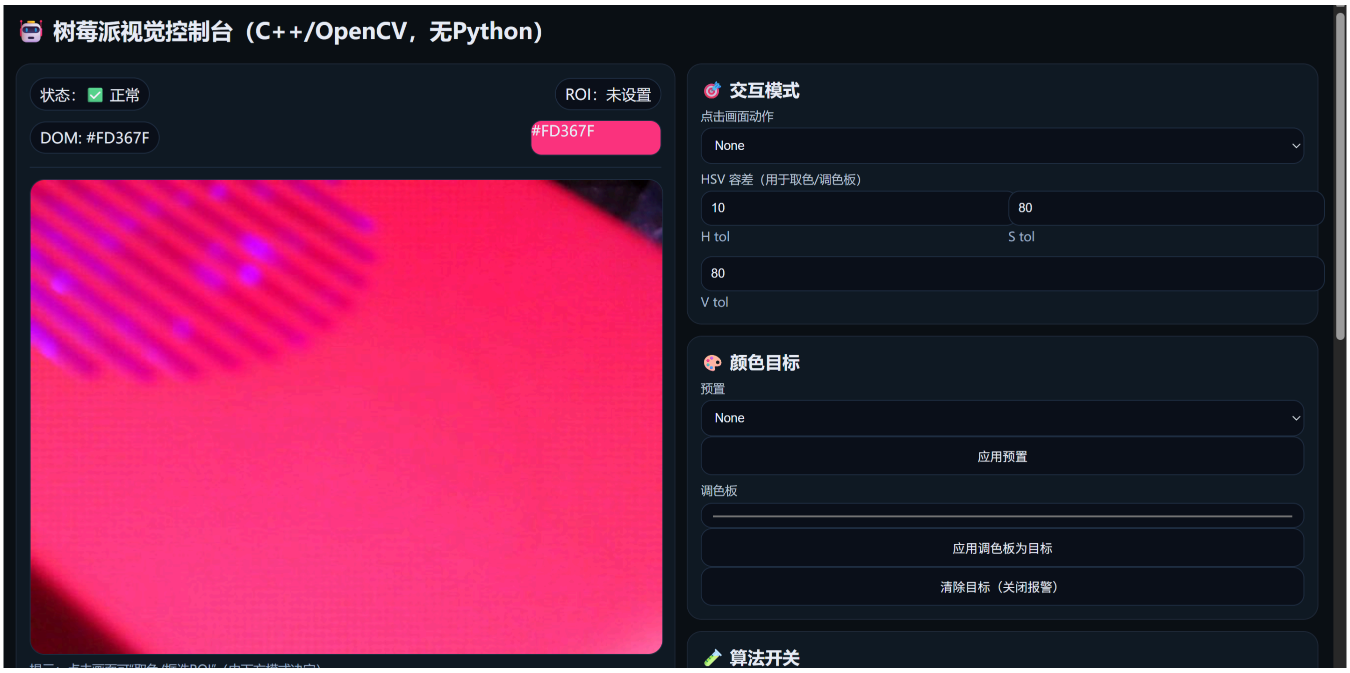
假设显示 `192.168.1.180`，浏览器打开：

- `http://192.168.1.180:7860/`

---

7) 浏览器使用说明

网页里：


树莓派视觉控制台（C++/OpenCV，无Python）

状态：✅ 正常

ROI：未设置

DOM: #FD367F

#FD367F

交互模式

点击画面动作: None

HSV 容差（用于取色/调色板）

H tol: 10

S tol: 80

V tol: 80

颜色目标

预置: None

7.1 实时画面

- 自动显示 `/stream.mjpg`

7.2 点击画面动作（最关键）

- **None**：不处理点击
- **点击取色**：点一下画面→采样像素→生成 HSV 范围→开启报警
- **两次点击框选ROI**：点两下→设 ROI → 主色只在 ROI 内计算，并显示 dominant_hex

7.3 颜色目标

- 预置：Red/Yellow/Green（红色自动处理 hue 环绕）
- 调色板：选颜色 + HSV 容差
- 清除目标：关闭报警

7.4 算法开关

- 报警：像素阈值、mask 预览
- 边缘检测：Canny t1/t2、overlay/only、alpha
- 模糊：核大小（奇数）
- ROI 框/主色计算：开关