M55M1 手势控制智能风扇

NuMicro® 32 位系列微控制器示例代码介绍

文件信息

应用简述	本示例程序在M55M1开发平台上实现以机器视觉进行图像识别，可以判断用户的手势及人体位置，进行风扇的智能控制，达到远程控制风扇启动并追踪人体的功能。
BSP 版本	M55M1_Series_BSP_CMSIS_V3.01.004
开发平台	NuMaker-GestureAI-M55M1 V1.0

The information described in this document is the exclusive intellectual property of
Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1. 概述

本示例程序将展示如何在 NuMaker-GestureAI-M55M1 开发平台上，结合机器视觉算法进行图像识别，以检测用户手势与人体位置信息，并据此实现风扇的智能控制功能。系统通过实时图像分析判断手势指令，控制风扇启动、停止及运转模式，同时利用人体位置追踪机制，自动调整风扇方向以跟随目标位置。此方法可在无需实体接触或传统遥控器的情况下，达成远程直观式控制，并提升使用便利性与互动体验，适合作为结合嵌入式 AI 与智能家电应用的参考示例。

1.1 NuMaker-GestureAI-M55M1 开发平台

M55M1 系列微控制器采用 Arm® Cortex®-M55 核心，最高运行频率达 220 MHz，内置 Arm® Helium™ DSP 技术以增强数字信号处理能力，并整合 Arm® Ethos™-U55 微型神经网络处理器，在数字信号处理及机器学习推理方面表现优异，同时可于边缘装置加速机器学习的推理。

本示例使用 NuMaker-GestureAI-M55M1 模块，如图 1-1 所示，其整合了摄像镜头以及 I²C、UART、USB 等通信界面，本模块充分发挥了 M55M1 核心的高性能处理能力与专属 AI 特性，协助开发者在边缘装置上实现复杂的人工智能应用。为了简化 AI 模型部署，模块配置了 Micro SD 卡槽用于储存 AI 模型文件；此外，模块上的摄像镜头，让 AI 应用能无缝实现图像采集、视频运算及视觉识别功能。

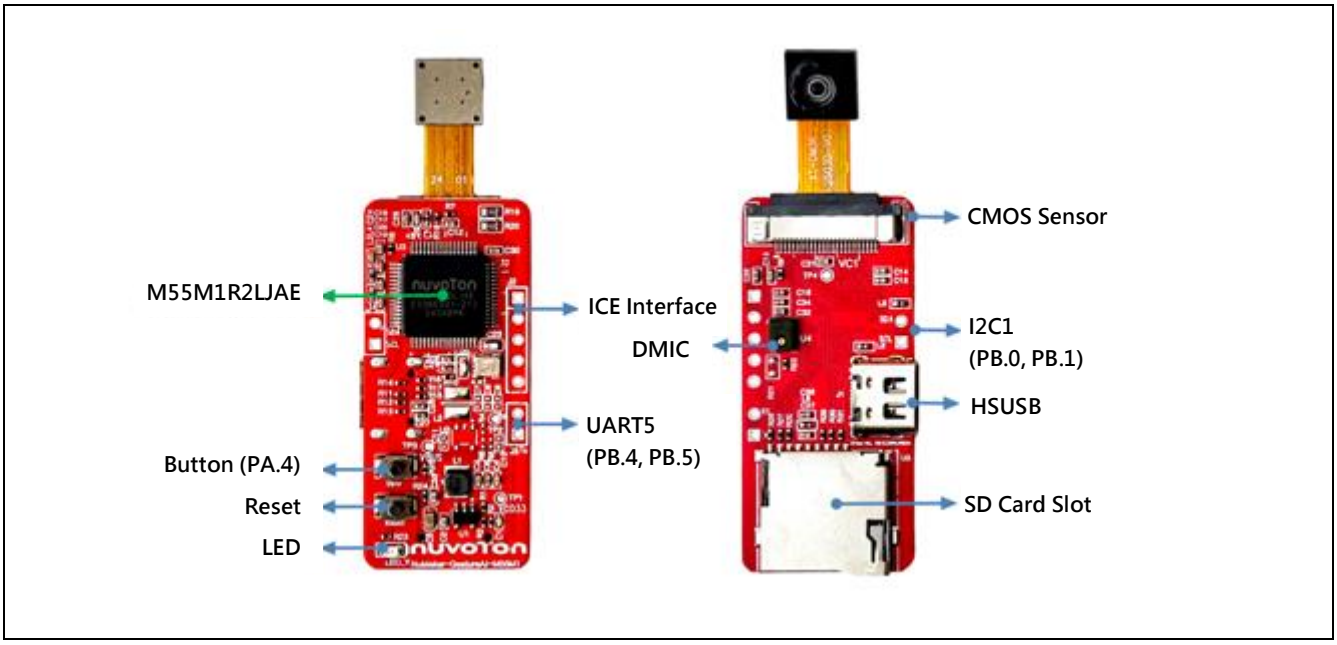


图 1-1 NuMaker-GestureAI-M55M1 模块

1.2 原理

本示例利用 M55M1 系列微控制器，将传统风扇升级为具备「主动感知」的智能家电。通过在边缘端执行高性能对象检测，让用户实现完全脱机、低延迟且保障隐私的非接触式控制。

手势检测系统取代了实体按键，改用直观手势进行互动：比出「OK」调整风速，「拳头」智能关机，或「Peace」启动广角摆头，如图 1-2 所示。最大亮点在于可以利用手势操控的「智能追踪模式」，系统能实时锁定用户坐标并驱动风扇摆头电机跟随，让风扇可以随人而动，达成单芯片整合视觉 AI 与电机控制的智能家电应用。

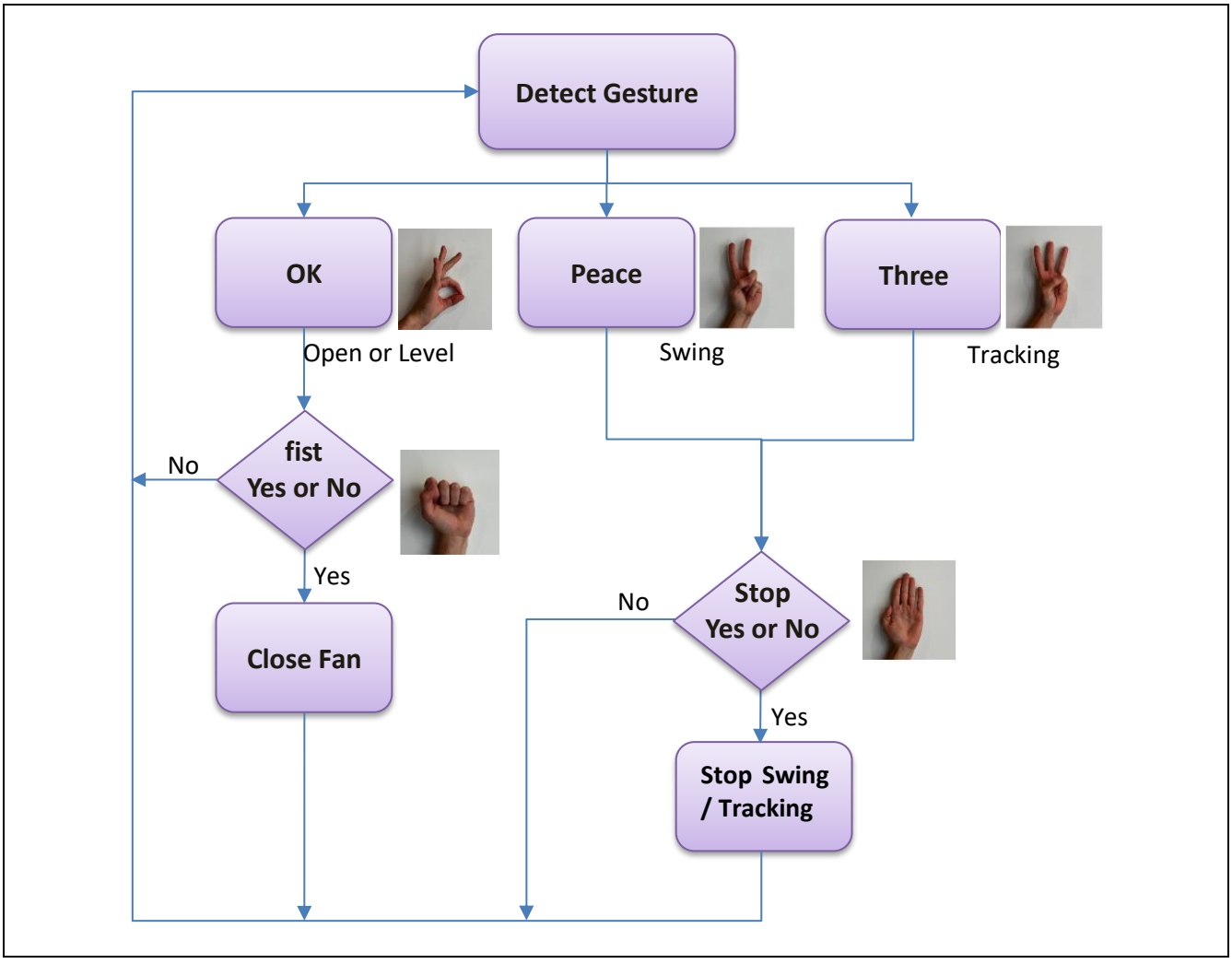


图 1-2 系统示意图

系统上的视觉 AI 使用 YOLOX Nano，是一种轻量化、anchor-free（无锚点）的对象检测模型，专为实时推理与边缘运算装置所设计。其架构采用 decoupled head（解耦式检测头），将分类（classification）与边界框回归（regression）分开处理，以提升训练稳定性与检测准确度。相

较于传统 anchor-based 方法，YOLOX Nano 直接在特征图上预测对象位置与类别机率，减少复杂的 anchor 设置与后处理计算负担。由于模型体积小、运算量低，因此特别适用于 MCU 的 AI 应用场景，在维持实时性的同时，也能提供稳定的对象与手势检测能力。

1.3 执行结果

本示例代码执行编译和刻录的结果如图 1-3 所示，通过 NuMaker-GestureAI-M55M1 模块上的 USB 接口，将模块作为 PC 的外接 USB Camera 使用，并且显示出镜头上检测到的手势。



图 1-3 程序执行结果

2. 代码介绍

本示例由 main() 进行整体初始化，包含时钟源初始化、风扇与步进电机初始状态设置、建立 Fan command queue，并依需求启动 UVC 装置。接着建立 main_task 与 vFanControlTask 两个 FreeRTOS 任务，最后调用 vTaskStartScheduler() 启动调度器，使系统进入多任务执行环境。其中 main_task 会进行手势逻辑处理以及建立第三个 FreeRTOS 任务 inferenceProcessTask 负责 AI 推理，而 vFanControlTask 则负责风扇运转控制与摆头电机控制，通过 Queue 接收指令并实际控制硬件动作。

```
int main()
{
    BaseType_t ret;

    FAN_GPIOInitIdleHigh();

    /* Initialize the UART module to allow printf related functions (if using retarget) */
    BoardInit();

    /* ------------------------------------------------------------------
     * Fan / stepper initialization
     *   - Do NOT start the fan on power-up.
     *   - Keep PA4 at idle HIGH to avoid an unintended High->Low edge.
     * ------------------------------------------------------------------ */
    FAN_Init();
    g_i32CurrentAngleX100 = kFanCenterAngleX100;
    g_i32TotalSteps = 0;
    g_u8MoveDone = 0;

    FAN_GPIOInitIdleHigh();

    g_fanCmdQueue = xQueueCreate(16, sizeof(FanCmdMsg));

    if (g_fanCmdQueue == nullptr)
    {
        info("FreeRTOS: Failed to create fan command queue\n");
        return -1;
    }

    #if defined(__USE_UVC__)
    UVC_Init();
    NVIC_SetPriority(HSUSBD_IRQn, 5);
    HSUSBD_Start();
    #endif

    /* Create main task. */
    ret = xTaskCreate(main_task, "main task", 2 * 1024, nullptr, MAINLOOP_TASK_PRIO,
nullptr);

    if (ret != pdPASS)
    {
```

```
        info("FreeRTOS: Failed to create main loop task \n");
        return -1;
    }

    ret = xTaskCreate(vFanControlTask, "fan_task", 1024, nullptr, 2, nullptr);

    if (ret != pdPASS)
    {
        info("FreeRTOS: Failed to create fan task \n");
        return -1;
    }

    // Start Scheduler
    vTaskStartScheduler();

    /* This is unreachable without errors. */
    info("program terminating...\n");
}
```

main_task() 负责建立 YOLOX Nano AI 模型、推理任务与图像传感器，并通过多重 Frame Buffer 管理图像数据。在主循环中，程序持续采集图像并送交 inferenceProcessTask 执行 AI 推理，待取得识别结果后调用 *HandleGestureAfterInference()* 进行手势判断与防抖处理，进而控制风扇转速、风扇关闭、电机摆头、目标追踪等功能。此外，程序也会在执行过程中，进行检测框绘制与 UVC 视频输出，形成完整的「图像采集 → AI 识别 → 手势控制 → 画面输出」应用流程。

```
static void main_task(void *pvParameters)
{
    BaseType_t ret;

    info("main task running \n");
    /* Model object creation and initialisation. */
    arm::app::YoloXnanoNu model;

    if (!model.Init(arm::app::tensorArena,
                    sizeof(arm::app::tensorArena),
                    arm::app::yoloxnanonu::GetModelPointer(),
                    arm::app::yoloxnanonu::GetModelLen()))
    {
        printf_err("Failed to initialise model\n");
        vTaskDelete(nullptr);
        return;
    }
…
    ret = xTaskCreate(inferenceProcessTask, "inference task", 1 * 1024, &taskParam,
INFERENCE_TASK_PRIO, nullptr);

…
    TfLiteTensor *inputTensor   = model.GetInputTensor(0);
    TfLiteTensor *outputTensor = model.GetOutputTensor(0);
…
```

```
#if defined (__USE_CCAP__)
    //Setup image sensor
    ImageSensor_Init();
    ImageSensor_Config(eIMAGE_FMT_RGB565, frameBuffer.w, frameBuffer.h, true);
#endif

    while (1)
    {
        infFramebuf = get_inf_framebuf();

        if (infFramebuf)
        {
            /* Detector post-processing*/
            inferenceJob->responseQueue = inferenceResponseQueue;
            inferenceJob->pPostProc = &postProcess;
            inferenceJob->modelCols = inputImgCols;
            inferenceJob->mode1Rows = inputImgRows;
            inferenceJob->srcImgWidth = infFramebuf->frameImage.w;
            inferenceJob->srcImgHeight = infFramebuf->frameImage.h;
            inferenceJob->results = &infFramebuf->results; //&results;

            xQueueReceive(inferenceResponseQueue, &inferenceJob, portMAX_DELAY);
        }

        fullFramebuf = get_full_framebuf();

        if (fullFramebuf)
        {
            //resize full image to input sensor
            image_t resizeImg;

            roi.x = 0;
            roi.y = 0;
            roi.w = fullFramebuf->frameImage.w;
            roi.h = fullFramebuf->frameImage.h;

            resizeImg.w = inputImgCols;
            resizeImg.h = inputImgRows;
            resizeImg.data = (uint8_t *)inputTensor->data.data; //direct resize to input
sensor buffer
            resizeImg.pixfmt = PIXFORMAT_RGB888;

            imlib_nvt_scale(&fullFramebuf->frameImage, &resizeImg, &roi);

            /* If the data is signed. */
            if (model.IsDataSigned())
            {
                arm::app::image::ConvertImgToInt8(inputTensor->data.data, inputTensor-
>bytes);
            }

            //trigger inference
            inferenceJob->responseQueue = inferenceResponseQueue;
            inferenceJob->pPostProc = &postProcess;
            inferenceJob->modelCols = inputImgCols;
            inferenceJob->mode1Rows = inputImgRows;
```

```
            inferenceJob->srcImgWidth = fullFramebuf->frameImage.w;
            inferenceJob->srcImgHeight = fullFramebuf->frameImage.h;
            inferenceJob->results = &fullFramebuf->results;

            xQueueSend(inferenceProcessQueue, &inferenceJob, portMAX_DELAY);
            fullFramebuf->eState = eFRAMEBUF_INF;
        }

        if (infFramebuf)
        {

            /* Smart gesture fan control: run AFTER YOLO results are ready. */
            HandleGestureAfterInference(infFramebuf->results);

            // draw bbox and render
            DrawImageDetectionBoxes(infFramebuf->results, &infFramebuf->frameImage,
labels);
            TrackObject(infFramebuf->results);

#if defined (__USE_UVC__)
            if (UVC_IsConnect())
            {
                Rotate180InPlace_2Bpp((uint8_t*)infFramebuf->frameImage.data,
                                      infFramebuf->frameImage.w,
                                      infFramebuf->frameImage.h);

                SCB_CleanDCache_by_Addr((uint32_t*)infFramebuf->frameImage.data,
                                        infFramebuf->frameImage.w * infFramebuf-
>frameImage.h * 2U);
                __DSB();
                __ISB();

                UVC_SendImage((uint32_t)infFramebuf->frameImage.data,
                              infFramebuf->frameImage.w * infFramebuf->frameImage.h * 2U,
                              uvcStatus.StillImage);
            }
#endif
            PresentInferenceResult(infFramebuf->results, labels);
            infFramebuf->eState = eFRAMEBUF_EMPTY;
        }

        emptyFramebuf = get_empty_framebuf();

        if (emptyFramebuf)
        {

#if defined (__USE_CCAP__)
            //capture frame from CCAP
            ImageSensor_Capture((uint32_t)(emptyFramebuf->frameImage.data));
#endif
            emptyFramebuf->results.clear();
            emptyFramebuf->eState = eFRAMEBUF_FULL;
        }

        vTaskDelay(1);
    }
```

```
}
```

inferenceProcessTask() 为独立的 AI 推理任务，会持续等待 main_task 送入的 inference job。当收到任务后调用 InferenceProcess::RunJob() 执行模型推理与后处理，完成后再将结果通过 response queue 回传给主流程。此设计可避免推理运算阻塞主任务，并让图像采集与手势控制能并行运行。

```
void inferenceProcessTask(void *pvParameters)
{
    struct ProcessTaskParams params = *reinterpret_cast<struct ProcessTaskParams
*>(pvParameters);

    InferenceProcess::InferenceProcess inferenceProcess(params.model);

    for (;;)
    {
        xInferenceJob *xJob;

        xQueueReceive(params.queueHandle, &xJob, portMAX_DELAY);

        inferenceProcess.RunJob(
            xJob->pPostProc,
            xJob->modelCols,
            xJob->mode1Rows,
            xJob->srcImgWidth,
            xJob->srcImgHeight,
            xJob->results
        );

        xQueueSend(xJob->responseQueue, &xJob, portMAX_DELAY);
    }

}
```

vFanControlTask() 为风扇与步进电机控制任务，负责系统开机后的步进电机自动更正，先移动至右端点再回到前方基准位置，以建立角度参考。任务通过 g_fanCmdQueue 接收来自手势或应用层的风扇指令，并以固定时间间隔产生 GPIO PA.4 控制脉波切换风扇等级，同时维护实际风扇状态。任务亦支持 FORCE_OFF 优先指令，可清除待处理脉波并计算所需脉波数将风扇关闭，确保风扇控制与步进电机动作可在背景持续运行且不影响 AI 任务。

```
void vFanControlTask(void *pvParameters)
{
    (void)pvParameters;

    /*
     * Fan pulse task
     *   - Receives fan commands from gesture logic (main_task)
     *   - Generates PA4 pulses with a minimum inter-pulse gap
     *
```

```
     * IMPORTANT: This task MUST NOT start the fan on power-up.
     */

    /* Ensure PA.4 stays idle HIGH (avoid unintended falling edge at boot). */
    FAN_GPIOInitIdleHigh();

    /* ---------------------------------------------------------------
     * Stepper boot calibration (restore original behavior)
     *   Step1: CW 1030 steps (go to right end)
     *   Step2: reset angle to 0, then CCW 560 steps (go to front)
     *
     * NOTE: We DO NOT start the fan here anymore.
     * --------------------------------------------------------------- */
    static bool s_calStarted = false;

    if (!s_calStarted)
    {
        g_stepperCalibrated = false;
        g_u8AutoTestState = 1;
        g_u8MoveDone = 0;
        STEPPER_MoveSteps(1030, 1);
        s_calStarted = true;
        info("[INIT] stepper auto-calibration start\n");
    }

    /* Fan speed state:
     *   0=OFF, 1=Speed1, 2=Speed2, 3=Speed3, 4=Speed4, 5=Speed5
     * NOTE: This variable MUST represent the *actual physical* level.
     *       Therefore we only update it AFTER a pulse is generated.
     */
    static uint8_t s_fanLevel = 0;

    uint8_t pendingPulses = 0;
    TickType_t nextPulseTick = 0;

    const TickType_t kPulseGapTicks = pdMS_TO_TICKS(350);
    const uint8_t kMaxPendingPulses = 20;

    info("Fan pulse task started (gesture-controlled)\n  ");

// --- Boot auto force-off (one-shot) ------------------------------------------
// Assumption: fan controller powers up at Speed1.
// States: 0(off) + 5 speeds => 6 states total.
// To go from 1 -> 0 needs 5 pulses: 1->2->3->4->5->0
    static bool s_bootForceOffDone = false;
    if (!s_bootForceOffDone)
    {
        // Sync SW level to known HW boot state.
        s_fanLevel = 1U;

        // Schedule pulses immediately; gap is enforced by kPulseGapTicks in the task
loop.
        pendingPulses = 5U;
        nextPulseTick = 0;

        s_bootForceOffDone = true;
```

```
        info("[INIT] boot auto force-off: 5 pulses scheduled\n");
    }

    for (;;)
    {
…

        /* Stepper calibration state machine: advance when move completes. */
        if (g_u8MoveDone)
        {
            g_u8MoveDone = 0;

            if (g_u8AutoTestState == 1)
            {
                /* Reached right end. Define this as angle 0.00 deg. */
                g_i32TotalSteps = 0;
                g_i32CurrentAngleX100 = 0;

                g_u8AutoTestState = 2;
                STEPPER_MoveSteps(560, -1);
                info("[INIT] stepper auto-calibration step2 (to front)\n");
            }
            else if (g_u8AutoTestState == 2)
            {
                g_u8AutoTestState = 0;
                g_stepperCalibrated = true;
                info("[INIT] stepper auto-calibration complete\n");
            }
        }

        bool forceOffRequested = false;

        /* Drain all pending commands (non-blocking). */
        FanCmdMsg msg;

        while ((g_fanCmdQueue != nullptr) && (xQueueReceive(g_fanCmdQueue, &msg, 0) ==
pdTRUE))
        {
            if (msg.cmd == FAN_CMD_FORCE_OFF)
            {
                forceOffRequested = true;
                /* Keep draining to clear the queue. */
            }
            else if (msg.cmd == FAN_CMD_NEXT_LEVEL)
            {
                /* If FORCE_OFF is requested in the same burst, ignore NEXT_LEVEL. */
                if (!forceOffRequested)
                {
                    if (pendingPulses < kMaxPendingPulses)
                    {
                        pendingPulses++;
                    }
                }
            }
        }
```

```
        if (forceOffRequested)
        {
            /*
             * Priority command: FORCE_OFF
             *   - discard any queued NEXT_LEVEL pulses
             *   - compute pulses needed from the *actual* current level
             */
            pendingPulses = 0;

            const uint8_t pulsesNeeded = (s_fanLevel == 0U) ? 0U : (uint8_t)(6U -
s_fanLevel);

            pendingPulses = pulsesNeeded;

            /* If other tasks send during this moment, they will be handled next loop. */
        }

        const TickType_t now = xTaskGetTickCount();

        if ((pendingPulses > 0U) && (nextPulseTick == 0 || now >= nextPulseTick))
        {
            generate_pulse();
            pendingPulses--;

            /* Update actual fan level AFTER pulse. */
            s_fanLevel++;

            if (s_fanLevel >= 6U)
            {
                s_fanLevel = 0U;
            }

            nextPulseTick = now + kPulseGapTicks;
        }

        vTaskDelay(pdMS_TO_TICKS(20));
    }
}
```

HandleGestureAfterInference() 负责将 YOLOX Nano 的对象检测结果转换为实际控制指令。函数首先从每一帧结果中挑选最佳手势候选，并通过跨帧 debounce 与 cooldown 机制进行稳定性过滤，避免短暂误判触发动作。当手势被确认后，依据类别分别触发不同控制行为，例如风扇转速切换与关闭、步进电机摆头启动/停止、以及人物追踪等，再通过 SendFanCmd() 将风扇控制命令送入 g_fanCmdQueue，由 vFanControlTask 实现稳定的手势驱动控制逻辑。

```
static void HandleGestureAfterInference(const
std::vector<arm::app::object_detection::DetectionResult> &results)
{
    double score = 0.0;
    int area = 0;

    /* 1) Pick a raw gesture candidate from this frame (or none). */
    const int candidateCls = FindBestGestureCandidate(results, score, area);
```

```
    /* 2) Debounce across frames: only fire on stable gesture-change edge. */
    const int firedCls = GestureFilterUpdate(candidateCls);

    if (firedCls < 0)
    {
        return;
    }

    const TickType_t now = xTaskGetTickCount();

    /* 3) Additional cooldown (protect against brief flicker changing stable state). */
    if (firedCls >= 0 && firedCls < 12)
    {
        const TickType_t cooldown = GestureCooldownTicks(firedCls);
        const TickType_t last = g_lastGestureFireTick[firedCls];

        if (last != 0 && (now - last) < cooldown)
        {
            return;
        }

        g_lastGestureFireTick[firedCls] = now;
    }

    switch (firedCls)
    {
        case kClsOk:
            /* Fan: cycle speed/off by one step. */
            SendFanCmd(FAN_CMD_NEXT_LEVEL);
            info("[GESTURE] ok -> fan next level (score=%.2f area=%d)\n", score, area);
            break;

        case kClsFist:
            /* Fan: force off by sending remaining pulses. */
            SendFanCmd(FAN_CMD_FORCE_OFF);
            info("[GESTURE] fist -> fan force off (score=%.2f area=%d)\n", score, area);
            break;

        case kClsPeace:
            /* Motor: start swing. */
            g_swingMode = true;
            STEPPER_Abort();
            info("[GESTURE] peace -> start swing (score=%.2f area=%d)\n", score, area);
            break;

        case kClsStop:
            /*
             * "stop" should stop swing & tracking:
             *   - stop swing
             *   - stop tracking
             */
            g_swingMode = false;
            g_trackingMode = false;
            g_cyclePersons = false;
            g_cycleIndex = 0;
```

```
            g_nextPersonSwitchTick = 0;
            STEPPER_Abort();
            info("[GESTURE] stop -> stop swing & tracking (score=%.2f area=%d)\n", score,
area);
            break;

        case kClsThree:
            /* Enter tracking mode. */
            g_trackingMode = true;
            g_swingMode = false;
            g_cyclePersons = false;
            g_cycleIndex = 0;
            g_nextPersonSwitchTick = 0;
            STEPPER_Abort();
            info("[GESTURE] three -> enter tracking mode (score=%.2f area=%d)\n", score,
area);
            break;

        case kClsPalm:
            g_trackingMode = true;
            /* In tracking mode: toggle cycling among detected persons. */
            if (g_trackingMode)
            {
                g_cyclePersons = !g_cyclePersons;
                g_cycleIndex = 0;
                g_nextPersonSwitchTick = now + pdMS_TO_TICKS(5000);
                info("[GESTURE] palm -> cycle persons %s (score=%.2f area=%d)\n",
                     g_cyclePersons ? "ON" : "OFF", score, area);
            }

            break;

        default:
            /* Other gesture classes are ignored in this spec. */
            break;
    }
}
```

3. 软件与硬件需求

3.1 软件需求

- BSP 版本
  - M55M1_Series_BSP_CMSIS_V3.01.004
- IDE 版本
  - Keil uVision 5.43
  - VS Code V1.112.0
    - CMSIS Solution extension
    - Arm Tools Environment Manager
    - Arm Compiler (ARMClang 6.x)

3.2 硬件需求

- 电路组件
  - NuMaker-GestureAI-M55M1
- 引脚连接
  - 通过杜邦线连接 NuMaker-GestureAI-M55M1 与风扇控制板(配置于风扇本体内)，如图 3-1 所示。

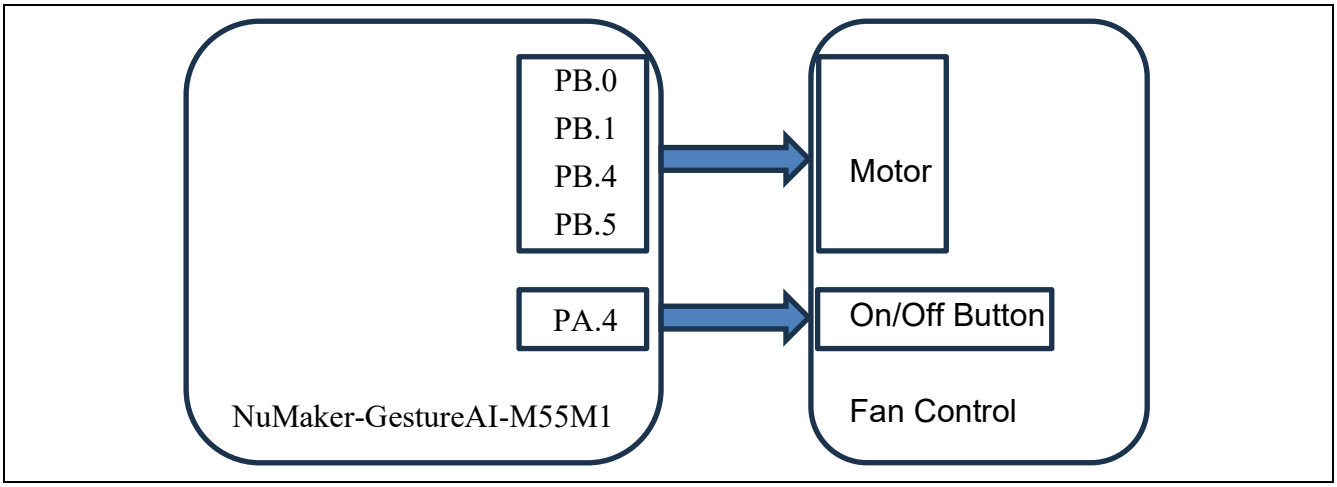


图 3-1 引脚连接

4. 目录信息

目录结构如下所示。

EC_M55M1_Gesture_Smart_Fan_Control_V1.00	
Library	Core software libraries and device support files
CMSIS	Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp.
Device	CMSIS compliant device header file
StdDriver	All peripheral driver header and source files
ThirdParty	
FreeRTOS	Real-time operating system kernel for embedded devices
ml-embedded-evaluation-kit	Machine learning embedded evaluation examples and reference applications
openmv	MicroPython-based machine vision framework for embedded cameras
tflite_micro	TensorFlow Lite for Microcontrollers inference library for tiny ML on MCU
SampleCode	
ExampleCode	
GestureFanControl	Example source code directory
Keil	Source code of Keil project
VSCode	Source code of VS Code project

图 4-1 目录信息

5. 示例程序执行

5.1 Keil MDK

1. 在 Windows 档案总管中进入 **Keil** 工程文件夹，再开启 *GestureFanControl.uvprojx* 工程文件。
2. 编译代码
3. 下载代码
4. 进入仿真模式
   - 执行代码

5.2 VS Code

1. 在 **VS Code** IDE 中开启 **File** 选单
   - 单击 **Open Folder**
   - 选取工程目录下的 **VSCode** 文件夹
2. 选择 **CMSIS**
   - 单击 **Build solution** 编译工程
   - 单击 **Load & Run application** 下载映像文件到目标芯片
   - 单击 **Load & Debug application** 进入仿真模式

6. 修订记录

Date	Revision	Description
2026.06.24	1.00	初始发布。

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.