M55M1 手勢控制智慧風扇

NuMicro® 32位元系列微控制器範例程式介紹

文件資訊

應用簡述	本範例程式在 M55M1 開發平台上實作以機器視覺進行影像識別，可以判斷使用者的手勢及人體位置，進行風扇的智慧控制，達到遠距控制風扇啟動並追蹤人體的功能。
BSP 版本	M55M1_Series_BSP_CMSIS_V3.01.004
開發平台	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

本範例程式將展示如何在 NuMaker-GestureAI-M55M1 開發平台上，結合機器視覺演算法進行影像識別，以偵測使用者手勢與人體位置資訊，並據此實現風扇的智慧控制功能。系統透過即時影像分析判斷手勢指令，控制風扇啟動、停止及運轉模式，同時利用人體位置追蹤機制，自動調整風扇方向以跟隨目標位置。此方法可在無需實體接觸或傳統遙控器的情況下，達成遠距離直覺式控制，並提升使用便利性與互動體驗，適合作為結合嵌入式 AI 與智慧家電應用的參考範例。

1.1 NuMaker-GestureAI-M55M1 開發平台

M55M1 系列微控制器採用 Arm® Cortex®-M55 核心，最高運作頻率達 220 MHz，內建 Arm® Helium™ DSP 技術以增強數位訊號處理能力，並整合 Arm® Ethos™-U55 微型神經網路處理器，在數位訊號處理及機器學習推論方面表現優異，同時可於邊緣裝置加速機器學習的推論。

本範例使用 NuMaker-GestureAI-M55M1 模組，如圖 1-1 所示，其整合了攝影鏡頭以及 I²C、UART、USB 等通訊介面，本模組充分發揮了 M55M1 核心的高效能處理能力與專屬 AI 特性，協助開發者在邊緣裝置上實現複雜的人工智慧應用 。為了簡化 AI 模型部署，模組配置了 Micro SD 卡槽用於儲存 AI 模型檔案；此外，模組上的攝影鏡頭，讓 AI 應用能無縫實現影像擷取、視訊運算及視覺辨識功能。

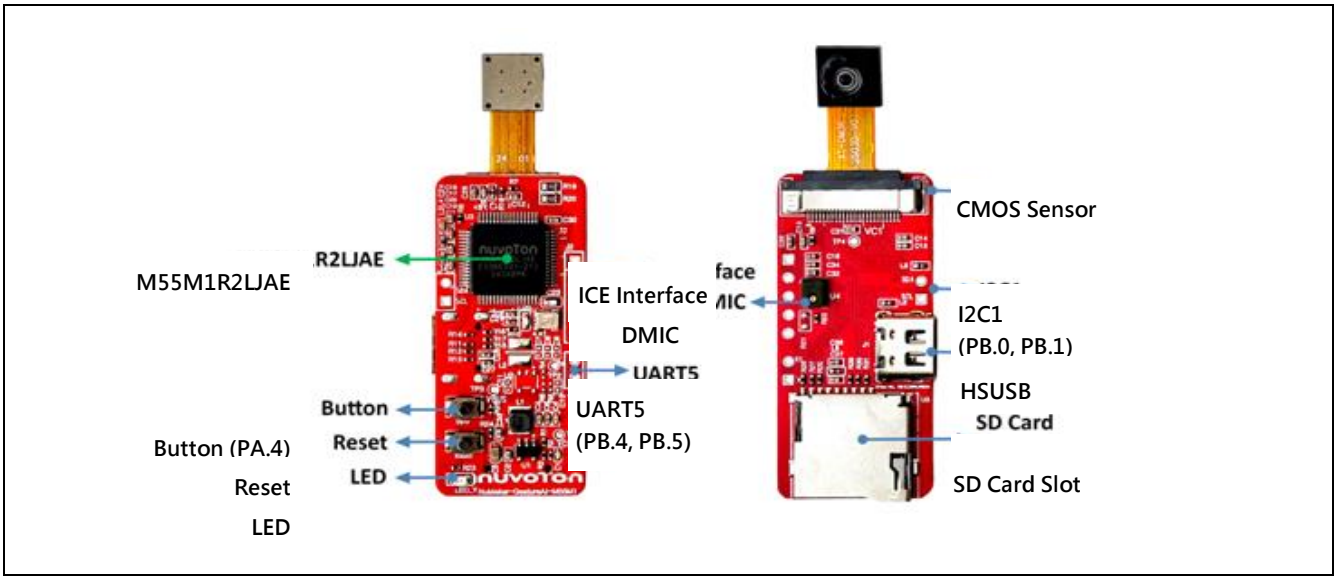


圖 1-1 NuMaker-GestureAI-M55M1 模組

1.2 原理

本範例利用 M55M1 系列微控制器，將傳統風扇升級為具備「主動感知」的智慧家電。透過在邊緣端執行高效能的物件偵測，讓使用者實現完全離線、低延遲且保障隱私的非接觸式控制。

手勢偵測系統取代了實體按鍵，改用直覺手勢互動：比出「OK」調整風速，「拳頭」 智慧關機，或 「Peace」啟動廣角擺頭，如圖 1-2 所示。最大亮點在於可以利用手勢操控的「智慧追蹤模式」，系統能即時鎖定使用者座標並驅動風扇擺頭馬達跟隨，讓風扇可以隨人而動，達成單晶片整合視覺 AI 與馬達控制的智慧家電應用。

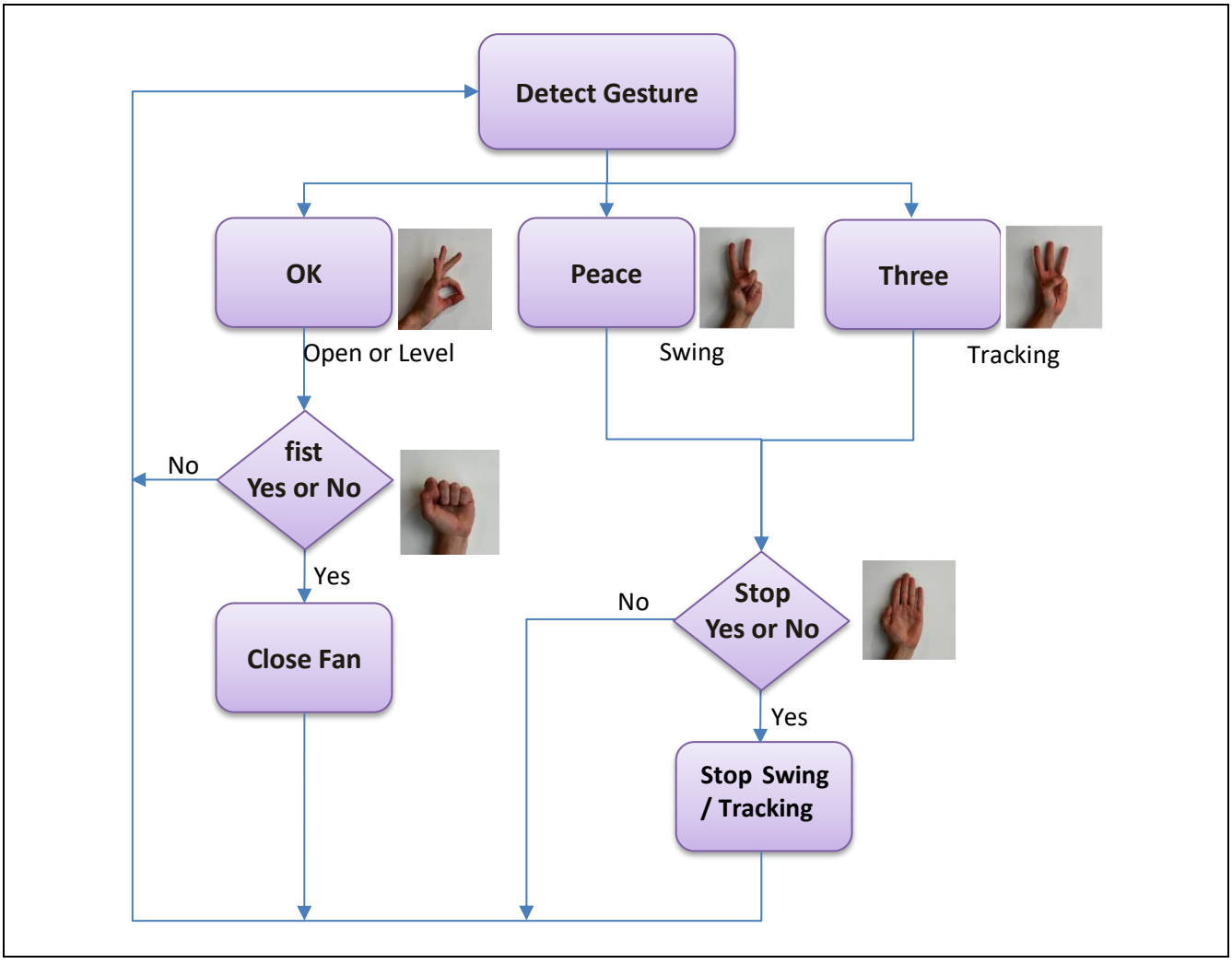


圖 1-2 系統示意圖

系統上的視覺 AI 是使用 YOLOX Nano，是一種輕量化、anchor-free（無錨點）的物件偵測模型，專為即時推論與邊緣運算裝置所設計。其架構採用 decoupled head（解耦式檢測頭），將分類（classification）與邊界框回歸（regression）分開進行處理，以提升訓練穩定性與偵測準

確度。相較於傳統 anchor-based 方法，YOLOX Nano 直接在特徵圖上預測物件位置與類別機率，減少複雜的 anchor 設定與後處理計算負擔。由於模型體積小、運算量低，因此特別適用於 MCU 的 AI 應用場景，在維持即時性的同時，也能提供穩定的物件與手勢偵測能力。

1.3 執行結果

本範例程式執行編譯和燒錄的結果如圖 1-3 所示，透過 NuMaker-GestureAI-M55M1 模組上的 USB 介面，將模組作為 PC 的外接 USB Camera 使用，並且顯示出鏡頭上偵測到的手勢。



圖 1-3 程式執行結果

2. 程式介紹

本範例由 main() 進行整體初始化，包含時鐘源初始化、風扇與步進馬達初始狀態設定、建立 Fan command queue，並依需求啟動 UVC 裝置。接著建立 main_task 與 vFanControlTask 兩個 FreeRTOS 任務，最後呼叫 vTaskStartScheduler() 來啟動排程器，使系統進入多工執行環境。其中 main_task 會進行手勢邏輯處理以及建立第三個 FreeRTOS 任務 inferenceProcessTask 負責 AI 推論，而 vFanControlTask 則負責風扇運轉控制與擺頭馬達控制，透過 Queue 接收指令並實際控制硬體動作。

```
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() 負責建立 YOLOX Nano AI 模型、推論任務與影像感測器，並透過多重 Frame Buffer 管理影像資料。在主迴圈中，程式持續擷取影像並送交 inferenceProcessTask 執行 AI 推論，待取得辨識結果後呼叫 *HandleGestureAfterInference()* 進行手勢判斷與防抖處理，進而控制風扇轉速、風扇關閉、馬達擺頭、目標追蹤等功能。此外，程式也會在執行過程中，進行偵測框繪製與 UVC 視訊輸出，形成完整的「影像擷取 → AI 辨識 → 手勢控制 → 畫面輸出」應用流程。

```
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() 為獨立的 AI 推論任務，會持續等待 main_task 送入的 inference job。當收到任務後呼叫 InferenceProcess::RunJob() 執行模型推論與後處理，完成後再將結果透過 response queue 回傳給主流程。此設計可避免推論運算阻塞主任務，並讓影像擷取與手勢控制能並行運作。

```
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() 為風扇與步進馬達控制任務，負責系統開機後的步進馬達自動校正，先移動至右端點再回到前方基準位置，以建立角度參考。任務透過 g_fanCmdQueue 接收來自手勢或應用層的風扇指令，並以固定時間間隔產生 GPIO PA.4 控制脈波切換風扇等級，同時維護實際風扇狀態。任務亦支援 FORCE_OFF 優先指令，可清除待處理脈波並計算所需脈波數將風扇關閉，確保風扇控制與步進馬達動作可在背景持續運行且不影響 AI 任務。

```
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() 負責將 YOLOX Nano 的物件偵測結果轉換為實際控制指令。函式首先從每一幀結果中挑選最佳手勢候選，並透過跨幀 debounce 與 cooldown 機制進行穩定性過濾，避免短暫誤判觸發動作。當手勢被確認後，依據類別分別觸發不同控制行為，例如風扇轉速切換與關閉、步進馬達擺頭啟動/停止、以及人物追蹤等，再透過 SendFanCmd() 將風扇控制命令送入 g_fanCmdQueue，由 vFanControlTask 實現穩定的手勢驅動控制邏輯。

```
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. 軟體與硬體需求

3.1 軟體需求

- 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 硬體需求

- 電路元件
  - NuMaker-GestureAI-M55M1
- 腳位連接
  - 透過杜邦線連接 NuMaker-GestureAI-M55M1 與風扇控制板(配置於風扇本體內)，如圖 3-1 所示。

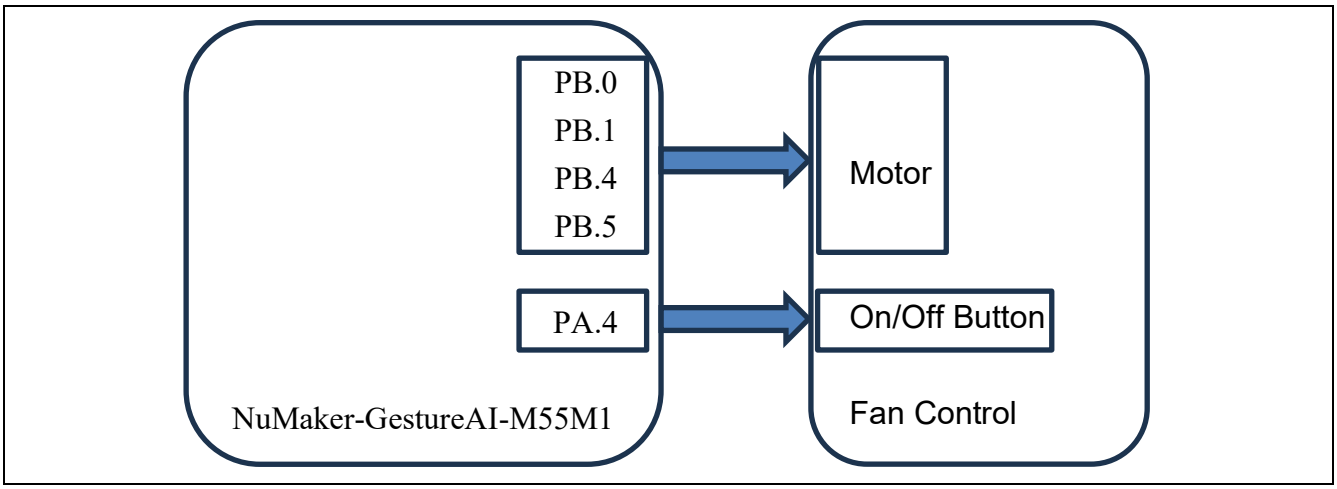


圖 3-1 腳位連接

4. 目錄資訊

目錄結構如下所示。

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

圖 4-1 目錄資訊

5. 範例程式執行

5.1 Keil MDK

1. 在 Windows 檔案總管中進入 **Keil** 專案資料夾，再開啟 *GestureFanControl.uvprojx* 專案檔。
2. 編譯程式碼
3. 下載程式碼
4. 進入偵錯模式
   - 執行程式碼

5.2 VS Code

1. 在 **VS Code** IDE 中開啟 **File** 選單
   - 選取 **Open Folder**
   - 選取專案目錄下的 **VSCode** 資料夾
2. 選擇 **CMSIS**
   - 點選 **Build solution** 編譯專案
   - 點選 **Load & Run application** 下載映像檔至目標晶片
   - 點選 **Load & Debug application** 進入偵錯模式

6. 修訂紀錄

Date	Revision	Description
2026.06.24	1.00	初始發布。

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.