Зміст

1 Вступ

Ця робота присвячена розробці системи автоматичного позиціонування пристроїв на базі мікроконтролера із застосуванням алгоритмів обробки зображень. Основна мета роботи – створення ефективної прошивки для обробки зображень у реальному часі, що забезпечує визначення положення об'єктів (зокрема яскравих точок, таких як Сонце, або ключових точок контурів підозрілих об'єктів у небі) і керування позиціонуванням пристрою за допомогою сервоприводів. Робота виконується в умовах обмежених обчислювальних ресурсів, що вимагає використання оптимізованих алгоритмів мовою C на операційній системі FreeRTOS.

Ключові завдання роботи:

- Розробка прошивки мікроконтролера для виконання поставлених задач.
- Реалізація алгоритмів обробки зображень для автономного виявлення об'єктів - цілей.
- Інтеграція системи з сервоприводами для автоматичного наведення камери.
- Оптимізація алгоритмів для роботи на платформі з обмеженими ресурсами.

Робота має практичне значення для створення автономних систем, таких як трекери сонячної енергії або системи відстеження підозрілих об'єктів у небі, і демонструє можливість застосування складних алгоритмів комп'ютерного зору на недорогих мікроконтролерах.

2 Опис проекту

2.1 Мета

Метою проєкту є розробка автономної системи позиціонування та наведення на основі мікроконтролера ESP32-CAM, яка здатна в реальному часі виявляти та відстежувати об'єкти за допомогою обробки зображень і керування сервоприводами. Система має забезпечити ефективне виконання задач описаних у попередньому розділі.

Проєкт спрямований на створення компактного, доступного та енергоефективного рішення для задач комп'ютерного зору та автоматичного наведення, придатного для використання в портативних пристроях, таких як системи відеоспостереження, трекери об'єктів або турельної установки.

2.2 Технічна база

Система автоматичного позиціонування пристроїв спирається на апаратну та програмну базу, ретельно підібрану для забезпечення ефективної обробки зображень, бездротового зв'язку та керування сервоприводами в умовах обмежених ресурсів. Основним апаратним компонентом є мікроконтролер ESP32-CAM, який поєднує обчислювальні можливості, камеру та комунікаційні модулі. Програмна частина базується на фреймворку ESP-IDF і операційній системі FreeRTOS, що забезпечують гнучке керування задачами та апаратними ресурсами.

2.2.1 Вибір мікроконтролера

Мікроконтролер ESP32-CAM обрано завдяки оптимальному поєднанню високої продуктивності, компактних розмірів (27 × 39 мм, вага 10 г) і доступної вартості. Модуль оснащений 32-бітним двоядерним процесором ESP32-S із частотою до 240 МГц (продуктивність до 600 DMIPS), камерою OV2640 із роздільною здатністю 2 Мп, 520 КБ вбудованої оперативної пам'яті та 4 МБ зовнішньої PSRAM. Наявність модулів Wi-Fi і Bluetooth, а також підтримка інтерфейсів UART, SPI, I2C, PWM, ADC, DAC і слота для microSD-карти забезпечують широкі можливості для обробки зображень, бездротового зв'язку та зберігання даних. Компактність і універсальність роблять ESP32-CAM ідеальним для портативних пристроїв, таких як системи відеоспостереження,

трекери об'єктів або компоненти «розумного будинку». Живлення від 5 В спрощує інтеграцію в різні системи, а підтримка камер OV2640 і OV7670 розширює гнучкість апаратної конфігурації.

2.2.2 Види пам'яті та їх використання

Мікроконтролер ESP32-CAM має кілька типів пам'яті, кожен із яких виконує специфічні функції та має свої особливості доступу. Розуміння їхнього призначення та правильне використання є критично важливим для оптимізації роботи системи, особливо в умовах обмежених ресурсів (520 КБ SRAM, 4 МБ PSRAM).

Основні типи пам'яті, доступні на ESP32:

- DRAM (Data RAM): Використовується для зберігання неконстантних статичних даних (`.data`), нуль-ініціалізованих даних (`.bss`) і динамічної пам'яті (heap). DRAM доступна для побайтового доступу, але не є виконуваною. Максимальний розмір статично виділеної DRAM обмежений 160 КБ через фрагментацію, спричинену ROM. Решта доступна як heap під час виконання. Наприклад, у задачі `photographer` використано 80.36% внутрішньої пам'яті (`INTERNAL`) і 82.81% DMA-сумісної пам'яті, що вказує на інтенсивне використання DRAM для буферів зображень.

- IRAM (Instruction RAM): Призначена для виконуваних інструкцій, доступна лише для операцій із 4-байтним вирівнюванням. Використовується для критичних до часу частин коду, таких як обробники переривань або функції, позначені `IRAM_ATTR`. У проєкті використано 56.88% `EXEC` пам'яті для задачі `photographer`, що включає код обробки зображень.

- DROM (Data ROM): Використовується для константних даних, які зберігаються у флеш-пам'яті та доступні через кеш MMU. Це зменшує використання DRAM, але може викликати затримки через промахи кешу.

- Noinit DRAM: Секція `.noinit`, позначена `__NOINIT_ATTR`, зберігає дані, які не ініціалізуються при старті та зберігають значення після програмного перезапуску. У проєкті не використовується, але може бути корисною для збереження стану між перезавантаженнями.

- RTC Fast/Slow Memory: Призначена для даних і коду, що використовуються в режимі глибокого сну або ULP-копроцесором. У проєкті не задіяна через відсутність таких вимог.

- SPIRAM (External PSRAM): Зовнішня пам'ять об'ємом 4 МБ, доступна для `.bss` або `.noinit` за допомогою `EXT_RAM_BSS_ATTR` або `EXT_RAM_NOINIT_ATTR`. У задачі `photographer` використано лише 1.45% SPIRAM, що вказує на мінімальне використання зовнішньої пам'яті через оптимізацію для внутрішньої SRAM.

Важливість розуміння типів пам'яті полягає в необхідності балансування між швидкістю доступу, обсягом доступної пам'яті та вимогами до виконання. Наприклад, розміщення буферів для DMA-контролерів (камера, SPI) у `DMA`-сумісній DRAM із вирівнюванням по слову є обов'язковим, що відображено у високому використанні `DMA` (82.81%). Неправильне розміщення даних може призвести до помилок або зниження продуктивності. Використання `IRAM` для критичних функцій, таких як `calculate_angles_diff`, зменшує затримки, але обмеженість `IRAM` (200 КБ) вимагає ретельного планування.

2.2.3 Вибір операційної системи

Для реалізації багатозадачності використано операційну систему реального часу FreeRTOS. Ця система обрана завдяки її компактності, ефективності та широкій підтримці для вбудованих систем, зокрема для архітектури ESP32. FreeRTOS забезпечує гнучке керування задачами, що дозволяє одночасно виконувати взяття й обробку зображень, керування веб-сервером, Wi-Fi-з'єднанням і сервоприводами в умовах обмежених ресурсів мікроконтролера.

FreeRTOS — це операційна система реального часу, яка підтримує як жорсткий, так і м'який режими роботи. У контексті цього проєкту застосовується м'який режим реального часу, оскільки затримки в обробці задач (наприклад, при захопленні зображень або оновленні веб-інтерфейсу) не призводять до критичних наслідків, але потребують оптимізації для забезпечення стабільної роботи. Основною перевагою FreeRTOS є можливість створення ізольованих задач, які виконуються паралельно завдяки розподілу процесорного часу між двома ядрами ESP32. Це досягається за рахунок планувальника, який розбиває задачі на невеликі фрагменти, що обробляються протягом так званих «тіків» таймера.

У проєкті використано ключові функції FreeRTOS для створення та керування задачами:

- `xTaskCreate`: Створює задачу, визначаючи її функцію, ім'я, розмір стека, параметри, пріоритет і дескриптор. Наприклад, задача `run_photographer` періодично захоплює зображення, тоді як `server_up` керує веб-сервером.

- `vTaskDelay`: Призупиняє задачу на заданий час (у тиках), дозволяючи іншим задачам отримати процесорний час. Це використано для регулювання частоти захоплення кадрів у `photographer.c`.

- `vTaskPrioritySet`: Динамічно змінює пріоритет задачі, що дозволяє, наприклад, підвищувати пріоритет обробки зображень під час активного наведення камери.

Кожна задача в системі працює як нескінченний цикл, що відповідає специфіці FreeRTOS. Наприклад, задача `run_servo_manager` постійно зчитує кути з черги `servo_queue` і оновлює положення сервоприводів. Для синхронізації задач використано черги (`xQueueCreate`, `xQueueSend`, `xQueueReceive`), які забезпечують передачу даних, таких як координати ключових точок або кути відхилення, між задачами обробки зображень і керування сервоприводами. М'ютекси (`xSemaphoreCreateMutex`) застосовуються для захисту спільних ресурсів, наприклад, буфера кадрів камери, від одночасного доступу кількома задачами.

FreeRTOS займає мінімальний обсяг пам'яті: ядро системи потребує 6–12 КБ ПЗУ та близько 0,5 КБ ОЗУ, що ідеально підходить для ESP32-CAM з його 520 КБ вбудованої оперативної пам'яті та 4 МБ зовнішньої PSRAM. Завдяки багатій екосистемі, FreeRTOS підтримує численні архітектури, включаючи ESP32, і надає готові приклади, що спростило інтеграцію в проєкт. Використання FreeRTOS у поєднанні з фреймворком ESP-IDF забезпечило ефективне розподілення обчислювальних ресурсів між задачами, такими як періодичне захоплення зображень, обробка даних за алгоритмами FAST-9 і DBSCAN, а також підтримка Wi-Fi-з'єднання та веб-сервера.

Таким чином, FreeRTOS стала основою для реалізації багатозадачної системи, забезпечуючи стабільність, гнучкість і ефективне використання апаратних можливостей ESP32-CAM. Це дозволило досягти реального часу обробки зображень і керування позиціонуванням із мінімальними затримками.

2.3 Функціональність

Розроблена система підтримує такі функції:

- Підключення до Wi-Fi по пріоритетності заданого списку точок (ssid, password), запуск mDNS, що дозволяє звертатися до пристрою за адресою `http://esp32.local` по спільній WiFi мережі.
- Періодичне створення знімків камерою через окремий FreeRTOS-таск (photographer task).
- Обробка зображень у реальному часі з використанням алгоритмів аналізу яскравості, положення сонця, або FAST-9 з DBSCAN для визначення ключових точок, підозрілих об'єктів у небі.
- Запуск веб-сервера для відображення зображень і обчислених даних у веб-інтерфейсі, з можливістю виділення прямокутної області на зображенні для зуму.
- Керування сервоприводами SG-90 для позиціонування камери за обчисленими кутами.

2.4 Структура проекту

Проект організовано в модульну структуру, що забезпечує чітке розділення функціональності та полегшує підтримку й розширення коду. Основні компоненти проєкту згруповано в директорії `frontend`, `include`, `src` та конфігураційні файли. Нижче наведено детальний опис структури проєкту з урахуванням функціонального призначення кожного модуля.

- frontend: Містить файли для веб-інтерфейсу системи.
  - `index.html`: Основна HTML-сторінка, яка відображає зображення з камери та обчислені дані (наприклад, координати ключових точок або центру маси яскравих областей). Сторінка забезпечує інтерактивну взаємодію, зокрема можливість виділення прямокутної області на зображенні.
  - `index_src.html`: Джерельний файл для генерації `index.html`, що використовується для спрощення оновлення веб-інтерфейсу.

- include: Директорія з заголовковими файлами, які містять оголошення функцій, структур даних і макросів.
  - `img_processing/`:
    - `camera.h`: Оголошення функцій для ініціалізації камери OV2640, роботи з буфером кадрів (`camera_fb_t`), обчислення кутів відхилення та обробки зображень (наприклад, функція `calculate_angles_diff`).
    - `dbscan.h`: Оголошення функцій для кластеризації ключових точок за алгоритмом DBSCAN, включаючи `dbscan_cluster` для групування точок і визначення центрів кластерів.
    - `find_sun.h`: Оголошення функції `mark_sun` для аналізу яскравості та виявлення найяскравіших пікселів (наприклад, Сонця).
    - `photographer.h`: Оголошення функцій для періодичного захоплення зображень (`run_photographer`) та обробки фрагментів кадру (`operate_fragment`).
    - `vision_defs.h`: Оголошення функцій і структур для алгоритму FAST-9 (`fast9`) та виявлення об'єктів, таких як дрони (`find_drone`).
  - `server/`:
    - `index_html.h`: Заголовковий файл із скомпільованою в char масив версією HTML-сторінки для відправки через веб-сервер.
    - `setting_mdns.h`: Оголошення функцій для налаштування mDNS (`set_mdns`) для доступу до пристрою за адресою `http://esp32.local`.
    - `webserver.h`: Оголошення функцій для запуску та роботи веб-сервера (`server_up`).
    - `wifi.h`: Оголошення функцій для підключення до Wi-Fi або створення точки доступу (`local_start_wifi`).
  - `defs.h`: Основний заголовковий файл із загальними визначеннями, макросами (наприклад, `MODE_FIND_SUN`, `MODE_FAST9`) та функціями для ініціалізації апаратного забезпечення (`init_esp_things`) і логування пам'яті (`log_memory`).
  - `my_servos.h`: Оголошення функцій для ініціалізації та керування сервоприводами SG-90 (`init_my_servos`, `run_servo_manager`).
  - `my_vector.h`: Оголошення структури `vector_t` та функцій для роботи з динамічними масивами (наприклад, `vector_create`, `vector_push_back`).

  - `README`: За замовчуванням розміщається у багатьох дефолтних директоріях проектів Platformio. Тут не використовується.

- src: Директорія з вихідними файлами реалізації функціональності.
  - `img_processing/`:
    * `vision/`:
      · `dbscan.c`: Реалізація алгоритму DBSCAN для кластеризації ключових точок, включаючи функції `dbscan_cluster`, `get_neighbors`, `expand_cluster` і `calculate_cluster_centers`.
      · `fast9.c`: Реалізація алгоритму FAST-9 для виявлення ключових точок, включаючи `fast9Detect`, `fast9ComputeScores` і `fastNonmaxSuppression`.
      · `vision.c`: Реалізація функцій для обробки зображень, зокрема `find_drone` для виявлення об'єктів із використанням FAST-9 і DBSCAN.
    * `camera.c`: Реалізація ініціалізації камери, обробки кадрів (`camera_fb_create`, `camera_fb_crop`) та обчислення кутів відхилення (`calculate_angles_diff`).
    * `find_sun.c`: Реалізація аналізу яскравості для виявлення найяскравіших пікселів (`mark_sun`, `find_max_brightness_pixels`).
    * `operating_with_fb.c`: Реалізація функцій для роботи з буфером кадрів, зокрема обрізання зображень.
    * `photographer.c`: Реалізація періодичного захоплення зображень і їх обробки (`run_photographer`, `operate_fragment`).
  - `server/`:
    * `setting_mdns.c`: Реалізація mDNS для спрощення доступу до пристрою через локальну мережу (`initialise_mdns`, `mdns_task`).
    * `webserver.c`: Реалізація веб-сервера для відображення зображень і даних, обробки WebSocket-з'єднань (`ws_handler`) та обробки запитів (`get_req_handler`, `rect_handler`).
    * `wifi.c`: Реалізація підключення до Wi-Fi або створення точки доступу (`connect_wifi`, `init_softap`).
  - `esp_things.c`: Реалізація ініціалізації апаратного забезпечення (GPIO, NVS) та функцій для логування пам'яті (`log_memory`).

- `main.c`: Точка входу програми, що ініціалізує всі модулі та запускає основні таски (`run_photographer`, `server_up`, `run_servo_manager`).
- `my_vector.c`: Реалізація динамічних масивів для зберігання координат пікселів (`vector_create`, `vector_reserve`, `vector_push_back`).
- `servos.c`: Реалізація керування сервоприводами через ШІМ-сигнали (`init_my_servos`, `run_servo_manager`).

• Конфігураційні файли:

- `CMakeLists.txt`: Скрипт для збирання проєкту з використанням CMake, що визначає залежності та правила компіляції.
- `Kconfig.projbuild`: Файл конфігурації для налаштування параметрів проєкту (наприклад, SSID і паролі Wi-Fi).
- `sdkconfig.esp32cam`: Конфігураційний файл для ESP-IDF, що визначає апаратні налаштування (наприклад, роздільну здатність камери, режими Wi-Fi).

Така структура забезпечує модульність, полегшує тестування та дозволяє ефективно розподіляти ресурси між різними компонентами системи. Кожен модуль має чітко визначену функціональність, що сприяє масштабованості та зрозумілості коду.

3 Обробка зображень

3.1 Аналіз яскравості

Метод аналізу яскравості використовується для виявлення найяскравіших об'єктів, наприклад, Сонця, у задачах позиціонування камери. Алгоритм визначає координати пікселів із максимальною яскравістю, обчислює їхній центр маси та кути відхилення від центру кадру для керування сервоприводами.

3.1.1 Основні кроки алгоритму

1. Виявлення найяскравіших пікселів: Прохід по всіх пікселях зображення з визначенням яскравості та збереженням координат пікселів із максимальною яскравістю.
2. Обмеження кількості пікселів: Якщо кількість пікселів перевищує ліміт (16), виконується відбір кожної k-ї точки.
3. Обчислення центру маси: Визначається середнє значення координат відібраних пікселів.
4. Визначення кутів відхилення: Обчислюються кути для наведення камери.

3.1.2 Обчислення яскравості

Для формату RGB565 яскравість B пікселя обчислюється як:

$$B = R + G + B, \tag{3.1}$$

де $R = (pixel \gg 11)\&0x1F$, $G = (pixel \gg 5)\&0x3F$, $B = pixel\&0x1F$. У форматі GRAYSCALE яскравість відповідає значенню пікселя.

3.1.3 Обмеження кількості пікселів

Кількість пікселів обмежується до $\mathrm{STD\_BRIGHTEST\_PIXELS\_COUNT} = 16$. Якщо кількість пікселів N більша, обирається кожна k-та точка:

$$k = \lfloor N/16 \rfloor + 1. \tag{3.2}$$

3.1.4 Обчислення центру маси

Центр маси (x_c, y_c) обчислюється як:

$$x_c = \frac{1}{N}\sum_{i=1}^{N} x_i, \quad y_c = \frac{1}{N}\sum_{i=1}^{N} y_i. \tag{3.3}$$

Результат округляється до цілого значення.

3.1.5 Реалізація на ESP32

Алгоритм оптимізований для ESP32 шляхом використання структури `vector_t`, обмеження кількості пікселів і формату GRAYSCALE. Перевірки на помилки забезпечують стабільність.

3.1.6 Псевдокод

```
function FindMaxBrightnessPixels(frame):
    max_brightness = 0
    coords = create_vector()
    for y from 0 to frame.height-1:
        for x from 0 to frame.width-1:
            brightness = frame[y*width + x]  // GRAYSCALE
            if brightness > max_brightness:
                max_brightness = brightness
                clear_vector(coords)
            if brightness == max_brightness:
                push_back(coords, (x, y))

    if coords.size > 16:
        new_coords = create_vector()
        divider = floor(coords.size / 16) + 1
        for i from 0 to coords.size step divider:
            push_back(new_coords, coords[i])
        destroy_vector(coords)
        coords = new_coords

    center_x = sum(coords.x) / coords.size
    center_y = sum(coords.y) / coords.size
    return {coords, center_coord=(center_x, center_y)}
```

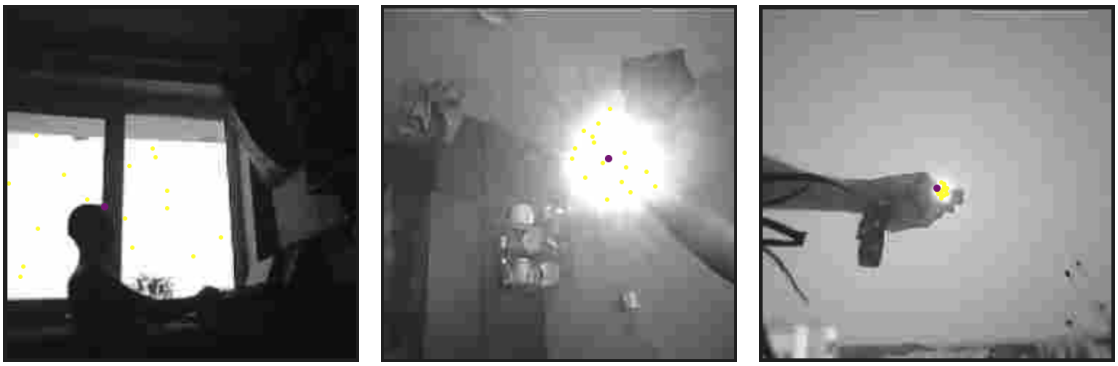

Рис. 3.1. Аналіз яскравості

3.2 Алгоритм FAST-9

Алгоритм FAST-9 (Features from Accelerated Segment Test) є ефективним методом виявлення ключових точок (кутів контурів) на зображенні, що вирізняється високою швидкістю та низькими обчислювальними вимогами, що робить його ідеальним для використання на мікроконтролері ESP32-CAM [rosten2006machine]. Алгоритм базується на евристичному підході, який дозволяє швидко ідентифікувати особливі точки, такі як кути об'єктів, шляхом аналізу пікселів на окружності навколо центральної точки. Його ефективність забезпечується оптимізацією порівнянь і застосуванням дерева рішень, що значно зменшує кількість обчислень.

3.2.1 Принцип роботи

FAST-9 аналізує 16 пікселів, розташованих на окружності радіусом 3 пікселі навколо центральної точки C. Для кожного пікселя p на окружності обчислюється його яскравість I_p і порівнюється з яскравістю центральної точки I_C за заданим порогом t. Піксель класифікується за такими умовами:

$$C_p = \begin{cases} I_p > I_C + t & \text{(світліший)}, \\ I_p < I_C - t & \text{(темніший)}, \\ I_C - t \leq I_p \leq I_C + t & \text{(подібний)}. \end{cases} \tag{3.4}$$

Точка C вважається ключовою, якщо на окружності є щонайменше 9 послідовних пікселів, які одночасно світліші або темніші за C. Це дозволяє ефективно виявляти кути, які мають значні контрасти у порівнянні з оточенням.

3.2.2 Оптимізація за допомогою дерева рішень

Для зменшення обчислювальної складності алгоритм використовує евристичний підхід, який починається з перевірки лише чотирьох пікселів (з но-

мерами 1, 5, 9, 13). Якщо щонайменше три з них відповідають умові (світліші або темніші за C), виконується повна перевірка всіх 16 пікселів. Цей початковий тест значно зменшує кількість точок, що потребують подальшого аналізу. Удосконалена версія FAST-9, запропонована в [rosten2006machine], використовує дерево рішень, яке оптимізує порядок порівнянь пікселів. Дерево рішень будується за допомогою машинного навчання, яке визначає оптимальну послідовність перевірки пікселів на основі зміни ентропії.

Ентропія множества точок обчислюється як:

$$H = (c + \overline{c}) \log_2(c + \overline{c}) - c \log_2 c - \overline{c} \log_2 \overline{c}, \tag{3.5}$$

де c — кількість особливих точок, $\overline{c}$ — кількість неособливих точок у множині. Зміна ентропії після перевірки пікселя p визначається як:

$$\Delta H = H - (H_{\text{dark}} + H_{\text{equal}} + H_{\text{bright}}), \tag{3.6}$$

де H_{dark}, H_{equal}, H_{bright} — ентропії підмножин точок, класифікованих як темніші, подібні та світліші відповідно. Піксель, який забезпечує максимальну зміну ентропії, обирається для перевірки на кожному кроці, що дозволяє зменшити середню кількість порівнянь до приблизно 2 на піксель [rosten2006machine]. Це досягається шляхом жадібного вибору пікселів, які надають найбільше інформації для класифікації точки як особливої чи неособливої.

Дерево рішень реалізується у коді через набір умовних операторів if-else, що генеруються на основі навчальних даних. Наприклад, у функції `fast9Detect` порівняння пікселів виконується в оптимальному порядку, визначеному деревом рішень, що значно прискорює обробку.

3.2.3 Подавлення немаксимумів

Для уникнення виявлення кількох сусідніх ключових точок використовується процедура подавлення немаксимумів. Сила ключової точки оцінюється метрикою V:

$$V = \max\left(\sum_{x \in S_{\text{bright}}} |I_x - I_C| - t, \sum_{x \in S_{\text{dark}}} |I_C - I_x| - t \right), \tag{3.7}$$

де S_{bright} і S_{dark} — множини пікселів, світліших і темніших за центральну точку відповідно, а t — порогове значення яскравості. Точки з нижчим значенням V відкидаються, що дозволяє залишити лише найвиразніші ключові точки. У реалізації (функція `fastNonmaxSuppression`) порівнюються сусідні точки в межах одного рядка та сусідніх рядків, щоб обрати точку з найвищим V.

3.2.4 Адаптація для ESP32

Алгоритм FAST-9 адаптовано для ESP32-CAM шляхом переписування коду з бібліотеки OpenCV мовою C без використання C++ STL й залежностей, що зменшує вимоги до пам'яті та забезпечує сумісність із обмеженими ресурсами мікроконтролера (520 КБ ОЗУ, 4 МБ PSRAM). Використовується формат GRAYSCALE для спрощення обчислень, оскільки він потребує лише одного каналу яскравості замість трьох у RGB. Поріг t адаптується динамічно залежно від умов освітлення, що дозволяє стабільно виявляти ключові точки в різних сценаріях. Для управління пам'яттю використано власну структуру `vector_t`, яка замінює стандартні контейнери STL, забезпечуючи ефективне зберігання координат ключових точок.

Ефективність FAST-9 на ESP32 зумовлена кількома факторами:

- Мінімальна кількість порівнянь: Завдяки дереву рішень середня кількість порівнянь зменшується до ≈ 2 на піксель, що критично важливо для мікроконтролера з обмеженою обчислювальною потужністю (240 МГц, 600 DMIPS).
- Оптимізована структура даних: Використання `vector_t` і статичних масивів для зберігання координат пікселів зменшує витрати пам'яті.
- Простота алгоритму: FAST-9 не потребує складних обчислень, таких як градієнти чи матриці, на відміну від інших алгоритмів (наприклад, SIFT або SURF), що робить його придатним для вбудованих систем.
- Паралельна обробка: Інтеграція з FreeRTOS дозволяє виконувати FAST-9 у окремій задачі, паралельно з іншими операціями, такими як захоплення зображень або керування веб-сервером.

3.2.5 Роль дерева рішень

Дерево рішень є ключовим елементом оптимізації FAST-9, оскільки воно мінімізує кількість порівнянь, необхідних для класифікації точки. У традиційному підході FAST потрібно перевіряти до 16 пікселів для кожної точки, що може бути надто затратно для ESP32. Дерево рішень, побудоване на основі зміни ентропії, визначає оптимальну послідовність порівнянь, дозволяючи швидко відкинути неособливі точки. Наприклад, якщо перші кілька пікселів не відповідають умові, точка класифікується як неособлива без перевірки решти пікселів. Це зменшує обчислювальне навантаження, що є критично важливим для мікроконтролерів із обмеженими ресурсами.

У реалізації (функція `fast9Detect`) дерево рішень реалізовано через вкладені умовні оператори, які відображають логіку, отриману з навчальних даних. Кожен вузол дерева відповідає порівнянню певного пікселя на окружності, а листки вказують на результат класифікації (особлива чи неособлива точка). Такий підхід дозволяє обробляти зображення в реальному часі навіть на слабких апаратних платформах, таких як ESP32-CAM.

3.2.6 Псевдокод

```
function FAST9Detect(image, width, height, threshold):
    corners = []
    pixel_offsets = computeOffsets(radius=3, stride=width)
    for y from margin to height-margin:
        for x from margin to width-margin:
            center = image[y*width + x]
            cb = center + threshold
            c_b = center - threshold
            if countPixels(image, pixel_offsets, [0,4,8,12], cb, c_b) >= 3:
                if countContinuousPixels(image, pixel_offsets, cb, c_b) >= 9:
                    corners.append((x, y))
    scores = computeScores(image, corners, pixel_offsets, threshold)
    corners = nonMaxSuppression(corners, scores)
    return corners

function computeScores(image, corners, pixel_offsets, threshold):
    scores = []
    for each corner in corners:
        score = fast9CornerScore(image, corner, pixel_offsets, threshold)
        scores.append(score)
    return scores

function nonMaxSuppression(corners, scores):
    result = []
    for each corner in corners:
        if corner has higher score than neighbors:
            result.append(corner)
    return result
```

Рис. 3.2. Демонстрація роботи FAST9 на ноутбуці

Рис. 3.3. FAST9 на практиці, виведено на веб сторінку

3.2.7 Висновки

Алгоритм FAST-9 є оптимальним вибором для обробки зображень на ESP32-CAM завдяки своїй швидкості, низьким вимогам до ресурсів і ефективному використанню дерева рішень. Це дозволяє виявляти ключові точки в реальному часі, що є критично важливим для задач автоматичного позиціонування, таких як відстеження об'єктів або наведення камери.

3.3 Алгоритм DBSCAN

Алгоритм DBSCAN (Density-Based Spatial Clustering of Applications with Noise) застосовується для кластеризації ключових точок, отриманих за допомогою FAST-9, з метою ідентифікації груп об'єктів, таких як дрони, у задачах автоматичного позиціонування [ester1996density]. Завдяки своїй здатності обробляти дані з шумом і виявляти кластери довільної форми, DBSCAN є оптимальним вибором для аналізу зображень у реальних умовах, де присутні викиди та неоднорідна щільність точок. Його ефективність на платформі ESP32-CAM забезпечується адаптацією до обмежених обчислювальних ресурсів.

3.3.1 Принцип роботи

DBSCAN базується на концепції щільності даних, групуючи точки, які розташовані близько одна до одної (в межах радіуса ϵ) і мають достатню кількість сусідів (не менше min_points). Алгоритм класифікує точки на три категорії:

- Основні (кореневі): точки, які мають щонайменше min_points сусідів у своїй ϵ-околу.
- Прикордонні: точки, які належать до ϵ-околу основної точки, але самі мають менше min_points сусідів.
- Шум (викиди): точки, які не є ані основними, ані прикордонними, і не належать до жодного кластера.

Кластери формуються шляхом об'єднання основних точок, які є тісно згруповані одна від одної, тобто між ними існує ланцюжок основних точок, кожна з яких належить до ϵ-околу попередньої.

Інтуїтивно DBSCAN можна уявити як процес групування людей на вечірці: особи, що стоять близько (в межах ϵ) і мають достатньо сусідів (min_points), утворюють ядро групи. Ті, хто стоїть поруч із цими ядрами, але не мають достатньої кількості сусідів, приєднуються до групи як прикордонні учасники. Одиночки, які не мають зв'язку з групами, позначаються як шум.

3.3.2 Основні кроки

Алгоритм працює ітеративно, обробляючи точки даних у наступному порядку:

1. Ініціалізація: усі точки позначаються як некласифіковані.
2. Для кожної некласифікованої точки P:

- Знаходження всіх сусідів у радіусі ϵ (функція `regionQuery`).
- Якщо кількість сусідів менша за min_points, точка позначається як шум.
- Якщо кількість сусідів $\geq$ min_points, створюється новий кластер, а точка P стає основною. Викликається процедура `expandCluster` для додавання всіх тісно згрупованих точок до кластера.

3. Після формування всіх кластерів обчислюються центри кластерів як середнє арифметичне координат їхніх точок.
4. Для задачі виявлення дронів обирається центр кластера з найменшою y-координатою (найвище розташований об'єкт).

3.3.3 Математична основа

DBSCAN використовує метрику відстані для визначення ϵ-околу. Для двох точок (x_1, y_1) і (x_2, y_2) евклідова відстань обчислюється як:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}. \tag{3.8}$$

Точка x є основною, якщо кількість точок у її ϵ-околу $E(x) = \{y \mid \rho(x, y) \leq \epsilon\}$ не менша за min_points:

$$|E(x)| \geq \text{min\_points}. \tag{3.9}$$

Центр кластера для множини точок $\{(x_i, y_i)\}_{i=1}^{N}$ обчислюється як:

$$x_c = \frac{1}{N}\sum_{i=1}^{N} x_i, \quad y_c = \frac{1}{N}\sum_{i=1}^{N} y_i. \tag{3.10}$$

Поняття тісної згрупованості формалізує зв'язок між точками: точка p тісно згрупована з точки q, якщо існує ланцюжок основних точок $p_1, p_2, \ldots, p_n$, де $p_1 = q$, $p_n = p$, і кожна наступна точка належить до ϵ-околу попередньої.

3.3.4 Реалізація на ESP32

Реалізація DBSCAN для ESP32-CAM адаптована до обмежених ресурсів мікроконтролера (520 КБ ОЗУ, 4 МБ PSRAM, 240 МГц). Для управління пам'яттю використано структуру `vector_t`, яка забезпечує динамічне зберігання координат точок і сусідів. Параметри $\epsilon = 20$ пікселів і min_points $= 3$ підібрано емпірично для ефективного виявлення дронів на зображеннях. Ці значення дозволяють знаходити компактні групи ключових точок, отриманих від FAST-9, і відфільтровувати шум, спричинений неоднорідним фоном або освітленням.

Щоб зменшити обчислювальну складність, відстань між точками обчислюється лише для обмеженої кількості кандидатів, а матриця відстаней не зберігається в пам'яті, що знижує вимоги до ОЗУ. У функції `dbscan_cluster` (файл `dbscan.c`) ітеративно обробляються точки, а сусіди додаються до черги для розширення кластера, що нагадує обхід графа в ширину. Для оптимізації використано статичні масиви для тимчасового зберігання сусідів, а динамічне виділення пам'яті мінімізовано.

Ефективність DBSCAN на ESP32 пояснюється такими факторами:

- Стійкість до шуму: Алгоритм автоматично відфільтровує викиди, що важливо для обробки зображень із неоднорідним фоном.
- Гнучкість форми кластерів: На відміну від k-means, DBSCAN не припускає сферичної форми кластерів, що дозволяє виявляти об'єкти складної конфігурації, такі як дрони.
- Оптимізована складність: У типових сценаріях складність становить $O(N \log N)$ за умови використання ефективних структур даних, що є прийнятним для невеликих наборів ключових точок (сотні точок після FAST-9).

3.3.5 Вибір параметрів

Вибір параметрів ϵ і min_points є критичним для роботи DBSCAN. У проєкті використано евристичний підхід:

- min_points = 3 обрано як мінімальну кількість точок для формування кластера, що відповідає типовим групам ключових точок на дронах. Значення 3 забезпечує баланс між стійкістю до шуму та чутливістю до невеликих кластерів.
- $\epsilon = 20$ пікселів початково визначено між сусідніми ключовими точками, отриманими від FAST-9. Це значення динамічне, в процесі може мінятись. Впливає на кількість кластерів точок, останніх мало б бути 2 - 3.

Для точнішого підбору параметрів можна обчислити середню відстань до min_points найближчих сусідів для кожної точки, відсортувати ці відстані та обрати ϵ у точці перегину графіка. Однак у проєкті використано фіксовані значення для спрощення реалізації та забезпечення реального часу роботи.

3.3.6 Обмеження та особливості

DBSCAN має кілька обмежень, які враховано під час реалізації:

- Чутливість до параметрів: Неправильний вибір min_points або калібрування ϵ може призвести до об'єднання різних кластерів або надмірного позначення точок як шуму. У проєкті це компенсується попередньою обробкою зображень FAST-9, що зменшує кількість нерелевантних точок.
- Прокляття розмірності: У двовимірному просторі зображень ($n = 2$) DBSCAN працює ефективно, але при збільшенні розмірності даних потрібна експоненціальна кількість точок, що неактуально для цього проєкту.
- Недетермінованість граничних точок: Прикордонні точки можуть приєднуватися до різних кластерів залежно від порядку обходу. У реалізації використано правило приєднання до першого знайденого кластера, що мінімізує вплив цього ефекту.

3.3.7 Псевдокод

```
function DBSCAN(points, epsilon, min_points):
    cluster_ids = array(size=points.size, value=UNCLASSIFIED)
    cluster_id = 0
    for i from 0 to points.size-1:
        if cluster_ids[i] == UNCLASSIFIED:
            neighbors = regionQuery(i, epsilon)
            if neighbors.size < min_points:
                cluster_ids[i] = NOISE
            else:
                cluster_ids[i] = cluster_id
                expandCluster(i, cluster_id, epsilon, min_points, neighbors)
                cluster_id += 1
    centers = computeClusterCenters(cluster_ids, points)
    highest_center = selectHighestCenter(centers)
    return centers, highest_center

function expandCluster(point, cluster_id, epsilon, min_points, neighbors):
    queue = neighbors
    while queue is not empty:
```

```
            q = queue.pop()
            if cluster_ids[q] == UNCLASSIFIED:
                cluster_ids[q] = cluster_id
                q_neighbors = regionQuery(q, epsilon)
                if q_neighbors.size >= min_points:
                    queue = queue + q_neighbors
            if cluster_ids[q] == NOISE:
                cluster_ids[q] = cluster_id

function regionQuery(point, epsilon):
    neighbors = []
    for each p in points:
        if distance(point, p) <= epsilon:
            neighbors.append(p)
    return neighbors
```

Рис. 3.4. Демонстрація роботи FAST9 DBSCAN поверх низькоякісного зображення на веб сторінці

3.3.8 Висновки

DBSCAN є потужним інструментом для кластеризації ключових точок у задачах комп'ютерного зору на ESP32-CAM, завдяки своїй стійкості до шуму та здатності обробляти кластери довільної форми. Адаптація алгоритму до обмежених ресурсів мікроконтролера та оптимальний вибір параметрів забезпечують його ефективність у реальному часі для виявлення об'єктів, таких як дрони, у складних умовах.

4 Позиціонування та наведення

В результаті обробки зображень отримується координата об'єкта взятому на зображенні. Для забезпечення точного наведення камери на об'єкт у системі реалізовано обчислення кутів відхилення цієї координати від центру кадру та керування сервоприводами. Ці процеси інтегровано з алгоритмами обробки зображень (FAST-9 і DBSCAN) для визначення положення об'єкта в кадрі та коригування орієнтації камери в реальному часі. Використання FreeRTOS і апаратних можливостей ESP32-CAM забезпечує ефективну координацію між обчисленнями та керуванням апаратними компонентами.

4.1 Обчислення кутів відхилення

Для наведення камери на об'єкт (наприклад, дрон) обчислюються кути відхилення центра маси об'єкта від центра кадру. Центр маси (x_c, y_c) визначається як середнє координат точок кластера, отриманого від DBSCAN. Відхилення обчислюються за формулами:

$$\Delta x = \frac{W}{2} - x_c, \quad \Delta y = \frac{H}{2} - y_c, \tag{4.1}$$

де W і H — ширина та висота кадру відповідно. Ці значення відображають зміщення центра об'єкта відносно центра зображення в пікселях.

Для переведення піксельних відхилень у кути використовується поле зору (FOV) камери OV2640, яке становить приблизно 52° по горизонталі та вертикалі. Кути відхилення FOV_x і FOV_y обчислюються як:

$$FOV_x = \arctan\left(\frac{\Delta x}{W/2 \cdot \tan 26^\circ}\right) \cdot \frac{180}{\pi}, \tag{4.2}$$

$$FOV_y = \arctan\left(\frac{\Delta y}{H/2 \cdot \tan 26^\circ}\right) \cdot \frac{180}{\pi}. \tag{4.3}$$

Ці формули враховують геометрію камери, де $\tan 26^\circ$ відповідає половині кута огляду (52°/2). Отримані кути FOV_x і FOV_y представляють необхідні зміни положення камери по горизонтальній і вертикальній осях для центрування об'єкта в кадрі.

Обчислення виконуються у функції `calculate_angles_diff` (файл `camera.c`), яка приймає координати центра маси від DBSCAN і повертає структуру `angles_diff_t` із значеннями кутів. Для забезпечення стабільності кути нормалізуються до діапазону 0–180°, що відповідає діапазону руху сервоприводів SG-90. Динамічна адаптація порогу для FAST-9 і параметрів DBSCAN дозволяє зменшити вплив шумів на точність обчислень.

4.2 Керування сервоприводами

Керування сервоприводами SG-90 реалізовано за допомогою широтно-імпульсної модуляції (ШІМ) через модуль `LEDC` мікроконтролера ESP32. Використовуються два канали:

- `LEDC_CHANNEL_1` (GPIO 14) для горизонтального руху (pan).
- `LEDC_CHANNEL_2` (GPIO 15) для вертикального руху (tilt).

Сервоприводи SG-90 підтримують діапазон кутів від 0° до 180°, що відповідає ширині імпульсу від 0,5 мс до 2,5 мс при частоті ШІМ 50 Гц. Переведення кутів у значення обов'язкового циклу (duty cycle) виконується функцією `angle2duty`:

$$\text{pulse\_width} = 0.5 + \left(\frac{\text{angle}}{180}\right) \cdot 2.0, \tag{4.4}$$

$$\text{duty} = \left(\frac{\text{pulse\_width}}{20.0}\right) \cdot 4096, \tag{4.5}$$

де `pulse_width` — ширина імпульсу в мілісекундах, а 4096 — роздільна здатність ШІМ при 12-бітному таймері.

Керування сервоприводами організовано через окрему задачу FreeRTOS `servo_manager_task`, яка отримує кути відхилення (FOV_x, FOV_y) через чергу `servo_queue` (розмір черги — 5 елементів типу `angles_diff_t`). Функція `my_servos_change_angles` оновлює кути сервоприводів, додаючи отримані відхилення до поточних значень `curr_pan_angle` і `curr_tilt_angle`, із перевіркою на межі діапазону (0–180°). Для уникнення різких рухів застосовується дільник `ANGLE_DIVIDER` = 1.5, який зменшує кутові зміни, забезпечуючи плавне позиціонування.

Ініціалізація сервоприводів виконується функцією `init_my_servos`, яка конфігурує таймер `LEDC_TIMER_1` із частотою 50 Гц і роздільною здатністю 12 біт, а також налаштовує канали ШІМ для GPIO 14 і 15. Початкові кути встановлюються на 90° (pan) і 80° (tilt), що відповідає нейтральному положенню камери. Деініціалізація (`deinit_my_servos`) зупиняє ШІМ-сигнали для безпечного вимкнення.

4.3 Інтеграція з FreeRTOS

Використання FreeRTOS дозволяє ефективно координувати задачі обробки зображень і керування сервоприводами. Задача `servo_manager_task` працює на ядрі 0 із пріоритетом 7, що забезпечує швидку реакцію на нові кути відхилення. Черга `servo_queue` із синхронізацією через `xQueueSend` і `xQueueReceive` забезпечує надійну передачу даних між задачами обробки зображень (`run_photographer`) і керування сервоприводами. Затримка `vTaskDelay(pdMS_TO_TICKS(10))` у циклі задачі дозволяє уникнути надмірного завантаження процесора, зберігаючи ресурси для інших задач.

5 Основні виклики та шляхи їх подолання

Під час розробки системи наведення виникла низка викликів, зумовлених апаратними та програмними обмеженнями платформи.

- Апаратні обмеження ESP32-CAM. Мікроконтролер має лише 520 КБ ОЗУ та 4 МБ PSRAM, що ускладнює обробку зображень високої роздільної здатності. Для оптимізації навантаження було застосовано перетворення кадрів у формат `GRAYSCALE`

  Логування пам'яті (`log_memory`) у проєкті показало ефективне використання ресурсів: 7.14% 32-бітної та 8-бітної пам'яті (`32BIT`, `8BIT`) і 6.47% загальної пам'яті (`DEFAULT`). Це свідчить про успішну оптимізацію, але високе використання 80.36% `INTERNAL` (80.36%) і 82.81% `DMA` вказує на потребу в подальшій оптимізації для обробки більших зображень або складніших сценаріїв.

- Складність синхронізації у FreeRTOS. Робота з багатопотоковістю на C вимагала ретельного керування пам'яттю. Початкові помилки в роботі з м'ютексами спричиняли взаємні блокування (deadlocks), які вдавалось усувати, уникнувши призначення спільного використання однакових таймерів, ШІМ-каналів для різних виводів.

- Нестабільність алгоритмів обробки зображень. Через апаратні обмеження точність роботи алгоритмів FAST-9 (детекція ключових точок) та DBSCAN (кластеризація) була низькою. Для часткової компенсації цього недоліку було впроваджено динамічну адаптацію їхніх параметрів (поріг t для FAST-9, ϵ та `min_points` для DBSCAN), однак це рішення потребує тестування за різних умов освітлення.

- Ускладнене налагодження (дебагінг). Локалізація помилок у багатопоточному середовищі була ускладнена через їхню відкладену та непередбачувану появу. Користувався стандартними системними логами `ESP_LOGI`, `ESP_LOGE` та саморобним моніторингом всіх видів використовуваної пам'яті (`log_memory`), що дозволили виявляти переповнення черг чи помилки в обчисленнях.

- Невеликий FPS спричиняє затримки при відстеженні рухомих об'єктів. Для забезпечення плавності руху було впроваджено поділ кута повороту (`ANGLE_DIVIDER`).

Завдяки оптимізації коду, ретельному налаштуванню параметрів та використанню спародійованої структури даних `vector_t` більшість зазначених

проблем вдалося частково вирішити. Проте подальше підвищення надійності та продуктивності системи вимагає всебічного тестування в реальних умовах експлуатації та вдосконалення механізмів управління пам'яттю.

6 Вигляд пристрою

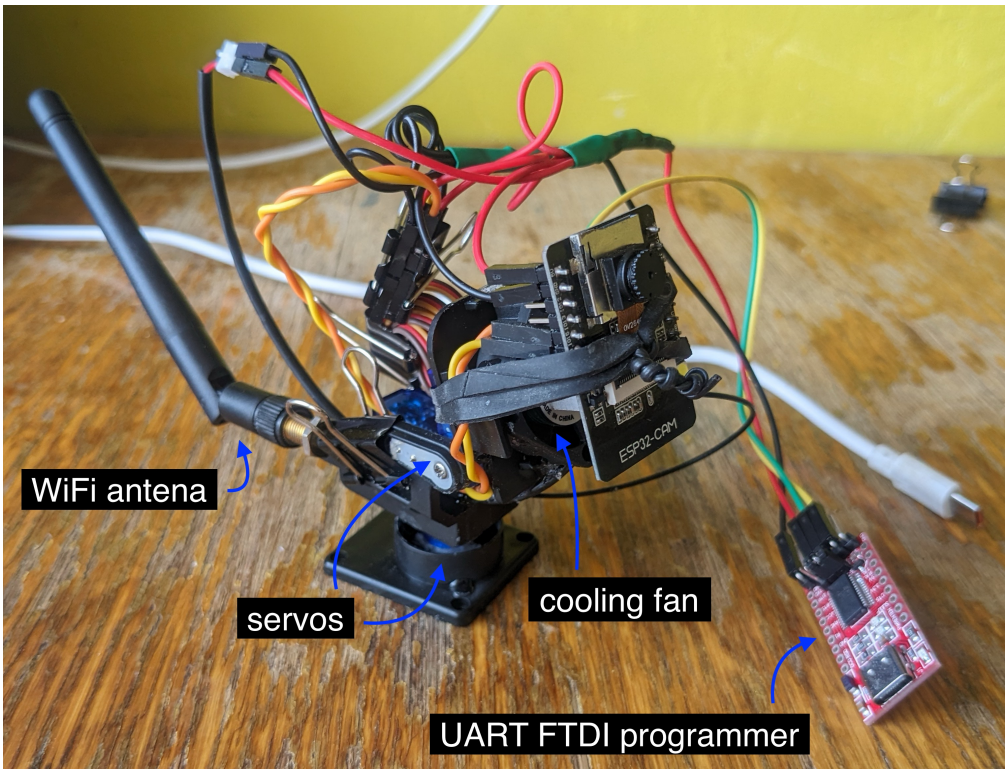

Рис. 6.1. Готовий пристрій

https://github.com/vidkryvashka/esp32-cam-positioning
посилання на Git-репозиторій проєкту.

7 Висновки

Розроблена система автоматичного позиціонування на базі мікроконтролера ESP32-CAM успішно реалізує задачі обробки зображень у реальному часі, виявлення та відстеження об'єктів, а також керування сервоприводами для наведення камери. Використання алгоритмів FAST-9 і DBSCAN дозволило ефективно виявляти ключові точки та кластеризувати їх, забезпечуючи надійне визначення положення об'єктів, таких як дрони, навіть у складних умовах із шумом і неоднорідним освітленням. Аналіз яскравості, адаптований для формату GRAYSCALE, виявився ефективним для задач на кшталт відстеження яскравих об'єктів, таких як Сонце.

Оптимізація алгоритмів для обмежених ресурсів ESP32-CAM (520 КБ ОЗУ, 4 МБ PSRAM, 240 МГц) досягнута завдяки використанню структури `vector_t`, динамічної адаптації параметрів, формату GRAYSCALE і частішим використанням окремих алокацій у зовнішню оперативну пам'ять PSRAM, що зменшило обчислювальне навантаження. Інтеграція з FreeRTOS забезпечила багатозадачність, дозволяючи одночасно виконувати захоплення зображень, їх обробку, керування сервоприводами та підтримку веб-сервера для відображення результатів через Wi-Fi.

Основні досягнення проєкту:

- Реалізація швидкого та стійкого до шуму виявлення об'єктів за допомогою комбінації FAST-9 і DBSCAN.
- Точне наведення камери завдяки обчисленню кутів відхилення та плавному керуванню сервоприводами SG-90.
- Модульна структура проєкту, яка полегшує підтримку та масштабування коду.
- Забезпечення роботи веб-інтерфейсу для інтерактивного відображення даних і керування системою.

Проєкт демонструє практичну цінність для створення компактних і доступних систем комп'ютерного зору, таких як трекери об'єктів, системи відеоспостереження або компоненти автоматизації. Отримані результати підтверджують можливість застосування складних алгоритмів обробки зображень на недорогих мікроконтролерах, відкриваючи перспективи для подальших досліджень у напрямку підвищення точності, швидкості та енергоефективності подібних систем. >

Література

[1] E. Rosten and T. Drummond, “Fusing points and lines for high performance tracking,” in Proc. 10th IEEE Int. Conf. on Computer Vision (ICCV), vol. 2, Beijing, China, 2005, pp. 1508–1515. DOI: 10.1109/ICCV.2005.114.

[2] E. Rosten and T. Drummond, “Machine learning for high-speed corner detection,” in Proc. 9th Eur. Conf. on Computer Vision (ECCV), Graz, Austria, 2006, pp. 430–443. DOI: 10.1007/11744023_34.

[3] E. Rosten, R. Porter, and T. Drummond, “Faster and better: A machine learning approach to corner detection,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 32, no. 1, pp. 105–119, Jan. 2010. DOI: 10.1109/TPAMI.2008.275.

[4] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise,” in Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD-96), Portland, OR, USA, 1996, pp. 226–231.

[5] C. Harris and M. Stephens, “A combined corner and edge detector,” in Proc. 4th Alvey Vision Conf., Manchester, UK, 1988, pp. 147–151. DOI: 10.5244/C.2.23.

[6] J. Shi and C. Tomasi, “Good features to track,” in Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 1994, pp. 593–600. DOI: 10.1109/CVPR.1994.323794.

[7] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” Int. J. Comput. Vis., vol. 60, no. 2, pp. 91–110, Nov. 2004. DOI: 10.1023/B:VISI.0000029664.99615.

[8] H. Bay, T. Tuytelaars, and L. Van Gool, “SURF: Speeded up robust features,” in Proc. European Conf. on Computer Vision (ECCV), Graz, Austria, 2006, pp. 404–417