

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

А. С. Момот, О. А. Повшенко

РОБОТОТЕХНІКА ТА ІНТЕЛЕКТУАЛЬНІ СИСТЕМИ ЛАБОРАТОРНИЙ ПРАКТИКУМ

*Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для студентів, які навчаються за спеціальністю
G7 «Автоматизація, комп'ютерно-інтегровані технології та робототехніка»,
освітньою програмою «Комп'ютерно-інтегровані технології та робототехніка»*

Київ
КПІ ім. Ігоря Сікорського
2025

Рецензент *Прізвище, ініціали, науковий ступінь, вчене звання, посада, місце роботи*

Відповідальний редактор *Прізвище, ініціали, науковий ступінь, вчене звання*

*Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № X від DD.MM.YYYY р.)
за поданням Вченої ради приладобудівного факультету (протокол № X від DD.MM.YYYY р.)*

Електронне мережне навчальне видання

*Момот Андрій Сергійович, доктор філософії, доцент
Повшенко Олександр Анатолійович, доктор філософії*

РОБОТОТЕХНІКА ТА ІНТЕЛЕКТУАЛЬНІ СИСТЕМИ ЛАБОРАТОРНИЙ ПРАКТИКУМ

Робототехніка та інтелектуальні системи: лабораторний практикум. [Електронний ресурс]: навч. посіб. для студ. спеціальності G7 «Автоматизація, комп'ютерно-інтегровані технології та робототехніка», освітньої програми «Комп'ютерно-інтегровані технології та робототехніка» / А. С. Момот, О. А. Повшенко; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 3,94 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2025. – 120 с.

Навчальний посібник містить мету і завдання лабораторних практикумів, їх зміст та обсяг. Розглянуто основні положення, послідовність та порядок виконання роботи, наведено завдання для індивідуального опрацювання та самостійної роботи.

© А. С. Момот, О. А. Повшенко 2025
© КПІ ім. Ігоря Сікорського, 2025

ЗМІСТ

1. Лабораторна робота №1. Частина 1. Створення графічного інтерфейсу для дистанційного керування мікроконтролером.....	4
2. Лабораторна робота №1. Частина 2. Дистанційне керування роботом.....	40
3. Лабораторна робота №2. Алгоритм автономної навігації робота.....	70
4. Лабораторна робота №3. Частина 1. Керування роботизованим маніпулятором на прикладі механізму Pan-Tilt.....	84
5. Лабораторна робота №3. Частина 2. Автоматичне детектування та відстеження об'єктів мобільним роботом.....	103

Лабораторна робота №1. Частина 1

СТВОРЕННЯ ГРАФІЧНОГО ІНТЕРФЕЙСУ ДЛЯ ДИСТАНЦІЙНОГО КЕРУВАННЯ МІКРОКОНТРОЛЕРОМ

Мета: Ознайомлення з основами дистанційного керування апаратними компонентами за допомогою Wi-Fi з'єднання, створення графічного інтерфейсу користувача (GUI) на PyQt та організація взаємодії між мікроконтролерами ESP32-CAM і мікроконтролером на платі Arduino Uno.

Теоретичні відомості

Система, розглянута в даній лабораторній роботі, призначена для дистанційного керування світлодіодом, розташованим на платі Arduino Uno, за допомогою комп'ютера через Wi-Fi-з'єднання. Вона демонструє принципи взаємодії апаратних і програмних компонентів у контексті розробки роботизованих систем із віддаленим управлінням, забезпечуючи передачу команд у реальному часі.

Система складається з трьох основних компонентів: модуля ESP32-CAM, плати Arduino Uno та програмного забезпечення на базі Python із графічним інтерфейсом (GUI), створеним за допомогою бібліотеки PyQt. Структурна схема, що демонструє взаємодію всіх компонентів, показана на рис. 1.1.

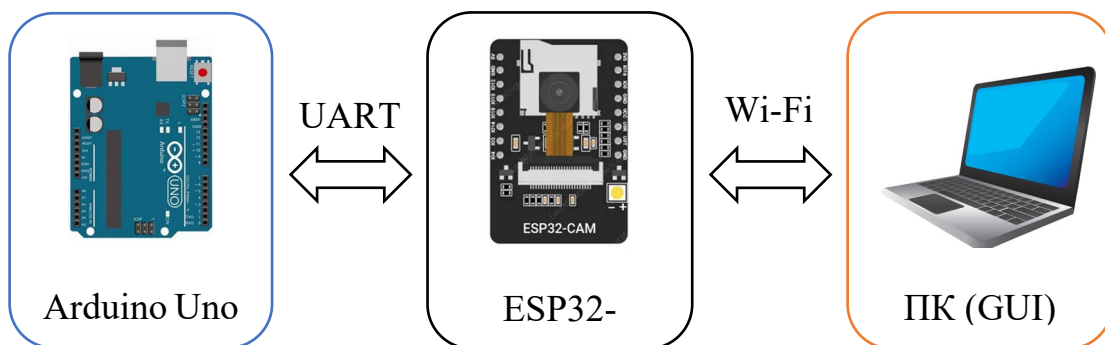


Рис. 1.1. Структурна схема системи дистанційного керування мікроконтролером на платі Arduino Uno

ESP32-CAM виконує функцію комунікаційного вузла, забезпечуючи підключення до Wi-Fi мережі та обмін даними через протокол WebSocket. Цей модуль отримує команди від комп'ютера та передає їх до Arduino Uno через послідовний інтерфейс UART. Плата Arduino Uno, оснащена вбудованим світлодіодом, відповідає за виконання отриманих команд, змінюючи стан світлодіода (увімкнення або вимкнення). Програмне забезпечення на комп'ютері реалізує графічний інтерфейс із кнопками для надсилання команд "увімкнути" або "вимкнути" світлодіод, а також відображає статус системи.

Принцип роботи такої системи полягає в послідовній передачі даних між компонентами. ESP32-CAM ініціалізує підключення до Wi-Fi мережі та створює WebSocket-сервер для взаємодії з комп'ютером. Користувач через графічний інтерфейс надсилає команду, яка передається через WebSocket до ESP32-CAM. Отримавши команду, ESP32-CAM надсилає її через UART до Arduino Uno, яка виконує відповідну дію зі світлодіодом і повертає підтвердження через той самий інтерфейс. ESP32-CAM пересилає це підтвердження назад до комп'ютера, де воно відображається в інтерфейсі користувача.

Таким чином, система інтегрує апаратні компоненти (ESP32-CAM, Arduino Uno) та програмне забезпечення (PyQt, WebSocket-сервер) для реалізації дистанційного керування. Вона ілюструє принципи мережевої взаємодії, послідовного зв'язку та створення інтерфейсу користувача, що є основою для подальших лабораторних робіт і розробки систем віддаленого керування роботами. Тепер розглянемо кожен компонент детальніше.

Модуль ESP32-CAM – це компактна плата на базі мікроконтролера ESP32-S, розроблена компанією Espressif Systems. Вона поєднує в собі потужний процесор, бездротові інтерфейси (Wi-Fi та Bluetooth), а також додаткові компоненти, такі як камера OV2640 і слот для картки microSD. У цій лабораторній роботі ESP32-CAM використовується як посередник для передачі команд від комп'ютера до плати Arduino через Wi-Fi-з'єднання та послідовний інтерфейс UART, забезпечуючи зв'язок між програмним забезпеченням і апаратними компонентами системи дистанційного керування.

Мікроконтролер ESP32-S має двоядерний 32-бітний процесор Tensilica LX6 із частотою до 240 МГц та RISC-архітектурою. Обсяг оперативної пам'яті становить 520 КБ SRAM, а також є 4 МБ зовнішньої PSRAM для обробки великих обсягів даних. Для зберігання програм використовується 4 МБ флеш-пам'яті. Модуль підтримує бездротові протоколи Wi-Fi (802.11 b/g/n) та Bluetooth 4.2 (BR/EDR і BLE), що дозволяє забезпечувати стабільне мережеве з'єднання. ESP32-CAM оснащена апаратними інтерфейсами, включаючи два послідовні порти UART, SPI, I2C, а також 10 GPIO-пінів для підключення зовнішніх пристроїв. Однак, потрібно врахувати, що більшість із цих виводів задіяні для підключення камери і карти пам'яті, а тому не доступні для використання. Схема виводів плати показана на рис. 1.2.



Рис. 1.2. Схема виводів плати ESP32-CAM

Додатково плата має вбудовану камеру OV2640 із роздільною здатністю до 2 МП, яка підтримує формати JPEG, BMP та відтінки сірого, а також слот для картки microSD для збереження даних.

Живлення плати здійснюється напругою 3.3 В, також є вхід зі стабілізатором для використання напруги 5 В. Для зручного програмування плати часто використовують адаптер на основі USB-UART перетворювачів CH340 або FT232RL.

ESP32-CAM є універсальним рішенням для широкого спектра застосувань, особливо в навчальних і дослідницьких проєктах. Завдяки підтримці Wi-Fi та Bluetooth, плата ідеально підходить для створення IoT-пристроїв, таких як системи розумного будинку, де вона може керувати освітленням, датчиками чи іншими пристроями через бездротову мережу. У робототехніці ESP32-CAM використовується для дистанційного керування роботами, передачі відеопотоку з камери або обробки даних з датчиків. Наявність камери робить плату придатною для проєктів із комп'ютерним зором, наприклад, розпізнавання об'єктів або відеоспостереження. Плата також використовується в проєктах із прототипування, де потрібна швидка розробка систем із низьким енергоспоживанням і можливістю бездротового зв'язку.

Arduino Uno – це мікроконтролерна плата, розроблена для спрощення створення проєктів з електроніки та робототехніки. Вона базується на мікроконтролері ATmega328P і є однією з найпоширеніших плат у сімействі Arduino. У контексті даної лабораторної роботи Arduino Uno використовується для обробки команд, отриманих через UART від модуля ESP32-Cam, і керування вбудованим світлодіодом, розташованим на піні 13. В цьому випадку плата слугує виконавчим елементом системи, безпосередньо керуючи індикаторами, датчиками та механізмами роботів.

Arduino Uno оснащена 8-бітним мікроконтролером ATmega328P із тактовою частотою 16 МГц та CISC-архітектурою. Плата має 32 КБ флеш-пам'яті для зберігання програм, 2 КБ SRAM для оперативних даних і 1 КБ EEPROM для постійного зберігання інформації. Вона містить 14 цифрових входів/виходів, з яких 6 можуть використовуватися для ШІМ (широтно-імпульсної модуляції), а також 6 аналогових входів для підключення датчиків. Arduino Uno підтримує послідовний інтерфейс UART через піни 0 (RX) і 1 (TX), який використовується в цій лабораторній роботі для зв'язку з ESP32-CAM. Живлення плати здійснюється через USB (5 В) або зовнішнє джерело живлення (7–12 В). Плата також має апаратну підтримку інтерфейсів SPI та I2C для взаємодії з іншими пристроями. Схема виводів показана на рис. 1.3.

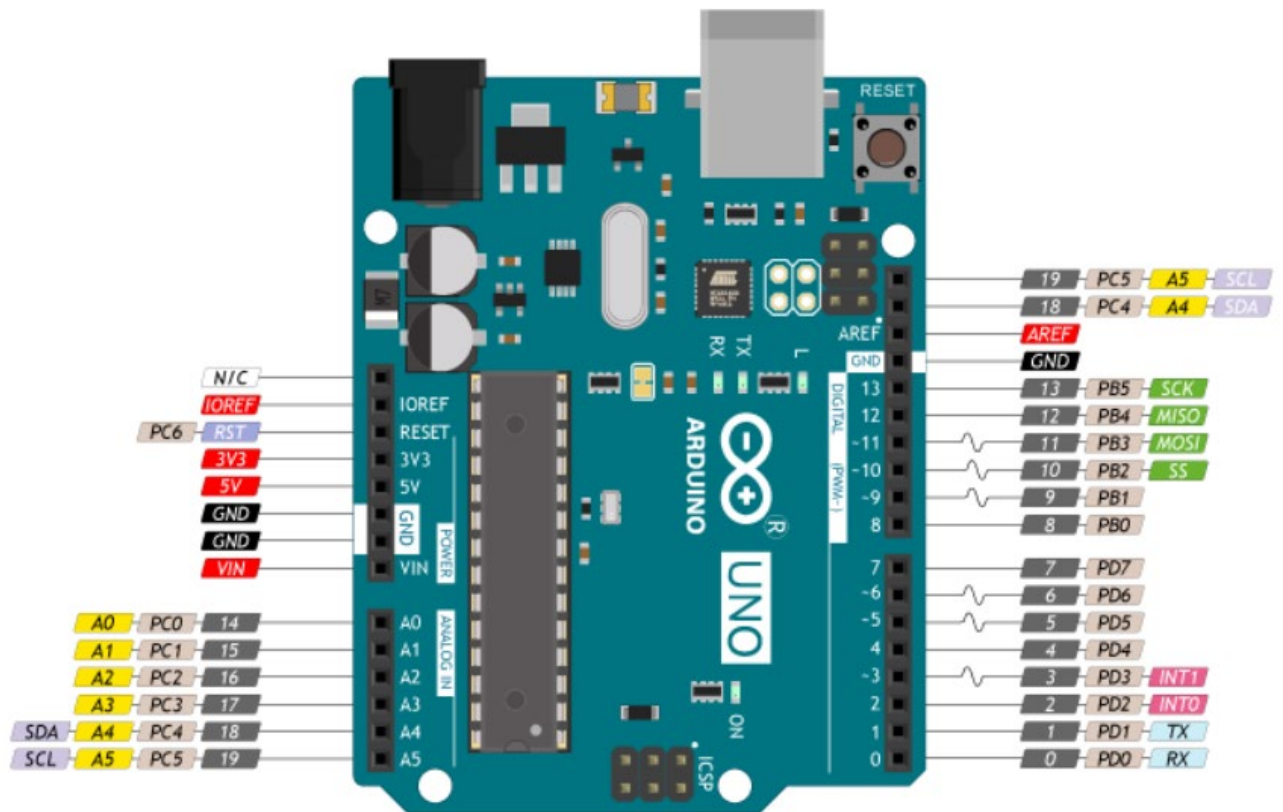


Рис. 1.3. Схема виводів плати Arduino Uno

Arduino Uno широко застосовується в навчальних і дослідницьких проєктах завдяки своїй простоті та універсальності. У робототехніці плата використовується для керування моторами, датчиками та виконавчими пристроями, такими як світлодіоди чи реле. У проєктах IoT Arduino Uno може слугувати основою для обробки даних від сенсорів перед їх передачею до зовнішніх модулів. Arduino Uno також використовується в прототипуванні електронних систем, наприклад, для створення автоматизованих пристроїв, освітлювальних систем або простих контролерів. Завдяки великій кількості сумісних модулів і бібліотек, плата є гнучким рішенням для швидкої розробки експериментальних проєктів.

Основними недоліками є низька швидкодія та розрядність, обмежена периферія і високе енергоспоживання. Наприклад, Arduino Uno має лише один апаратний UART, який одночасно використовується і для підключення зовнішніх пристроїв, і для програмування самого мікроконтролера на платі.

Для створення двостороннього зв'язку між графічним інтерфейсом на комп'ютері та платою Arduino Uno ми будемо використовувати WebSocket-сервер, реалізований на платі ESP32-CAM. Він приймає команди (наприклад, "on" або "off") від GUI-клієнта, пересилає їх до Arduino Uno через UART та повертає логи про стан системи до графічного інтерфейсу.

Загалом, **WebSocket** – це протокол прикладного рівня, який працює поверх протоколу TCP і призначений для двостороннього зв'язку між клієнтом (у цьому випадку – графічним інтерфейсом на ПК) і сервером (ESP32- Cam). На відміну від HTTP, який використовує модель "запит-відповідь" і закриває з'єднання після кожного обміну, WebSocket встановлює постійне з'єднання після початкового "рукоштовування" (Handshake) через HTTP. Це дозволяє клієнту та серверу надсилати повідомлення один одному в будь-який момент без необхідності повторного встановлення з'єднання. Схема такої взаємодії показана на рис. 1.4.



Рис. 1.4. Схема WebSocket-з'єднання

У цій лабораторній роботі WebSocket-сервер на ESP32-CAM створюється за допомогою бібліотеки *ESPAsyncWebServer*, яка забезпечує асинхронну обробку запитів. Це зменшує навантаження на ресурси мікроконтролера, дозволяючи одночасно обробляти кілька клієнтів і передавати дані в реальному часі. Основною особливістю WebSocket є підтримка двосторонньої передачі

текстових або бінарних даних із низькими накладними витратами, що забезпечує швидкість і ефективність.

На відміну від HTTP-сервера, який працює за принципом одноразового обміну (клієнт надсилає запит, сервер відповідає, а з'єднання закривається), WebSocket підтримує постійне з'єднання, що усуває затримки, пов'язані з повторними запитами. HTTP-сервер є більш ресурсоемним через необхідність встановлення нових TCP-з'єднань для кожного запиту, тоді як WebSocket використовує одне з'єднання для всіх обмінів. Крім того, HTTP більше підходить для статичних веб-сторінок або одноразового завантаження даних, тоді як WebSocket оптимальний для застосувань, що потребують частого оновлення даних, таких як чати, ігри чи системи реального часу. У даній лабораторній роботі HTTP-сервер був би менш ефективним через затримки та неможливість миттєвого надсилання логів від ESP32-CAM до клієнта. HTTP-запити були б повільнішими через необхідність повторного встановлення з'єднання і не дозволили б ефективно реалізувати двосторонній зв'язок для отримання логування стану системи в реальному часі.

Протокол WebSocket має широкий спектр застосувань, особливо в проєктах IoT і робототехніки. У IoT він використовується для дистанційного керування пристроями, такими як розумні розетки, системи освітлення чи датчики, де потрібна швидка передача команд і оновлення статусу. У робототехніці WebSocket застосовується для передачі керуючих сигналів до роботів або для потокової передачі даних із сенсорів. Технологія також використовується в системах моніторингу в реальному часі, наприклад, для відображення даних із датчиків температури чи вологості.

Потрібно також пам'ятати, що реалізація WebSocket-сервера потребує додаткових обчислювальних ресурсів порівняно з простими протоколами, що може бути обмеженням для деяких мікроконтролерів. Налаштування WebSocket складніше, ніж HTTP, оскільки вимагає коректної реалізації "рукописання" та обробки подій. Крім того, WebSocket менш сумісний із деякими мережевими проксі-серверами, що може ускладнювати роботу в певних мережах.

У лабораторній роботі для створення графічного інтерфейсу користувача (GUI) використовуються бібліотека PyQt і програма Qt Designer. PyQt забезпечує програмну основу для реалізації функціоналу інтерфейсу, включаючи обробку подій і асинхронний зв'язок із WebSocket-сервером. Qt Designer, у свою чергу, є середовищем для візуального проєктування інтерфейсу, що значно спрощує створення його графічної частини. Разом ці інструменти забезпечують зручний і ефективний спосіб взаємодії користувача з апаратною системою.

PyQt – це бібліотека для мови Python, яка є набором прив'язок до фреймворку Qt, розробленого компанією Qt Company. Вона дозволяє створювати кросплатформні графічні інтерфейси для застосунків, які працюють на Windows, Linux і macOS. PyQt базується на C++ бібліотеці Qt, але адаптована для використання з Python, що робить її доступною для розробників, які працюють у цьому середовищі. У лабораторній роботі ми будемо використовувати PyQt для створення вікна з кнопками для надсилання команд на включення і вимкнення світлодіоду, а також для відображення статусу підключення та логів. Бібліотека підтримує інтеграцію з асинхронними бібліотеками, що дозволяє обробляти WebSocket-з'єднання в реальному часі. PyQt пропонує широкий набір віджетів (кнопки, текстові поля, мітки тощо), які можна налаштувати для створення інтерактивних інтерфейсів.

У порівнянні з іншими бібліотеками GUI, такими як Tkinter, PyQt пропонує більш сучасний вигляд інтерфейсів і ширші можливості для стилізації. У цій лабораторній роботі, окрім безпосередньої реалізації графічного інтерфейсу, PyQt забезпечує стабільну обробку асинхронних операцій. Це є важливим для взаємодії з WebSocket.

Qt Designer – це програмний інструмент із пакету Qt, який дозволяє створювати графічні інтерфейси шляхом перетягування елементів (drag-and-drop) у візуальному редакторі. Qt Designer значно спрощує процес розробки, дозволяючи розробникам зосередитися на функціональності, а не на написанні коду для розміщення елементів інтерфейсу. Qt Designer генерує файли у форматі .ui (XML), які легко інтегруються з PyQt.

Хід виконання роботи

1. Написати код програми для ESP32-CAM для реалізації зв'язку з Arduino Uno по UART та з ПК по Wi-Fi через WebSocket-сервер.
 - 1.1. Запустити середовище програмування Arduino IDE.
 - 1.2. Відкрити менеджер плат, обравши в меню *Tools* → *Board* → *Boards Manager*.
 - 1.3. У полі пошуку ввести «ESP32» та встановити пакет *esp32 by Espressif Systems* (рис. 1.5). Якщо пакет вже був попередньо встановлений – пропустити даний пункт.

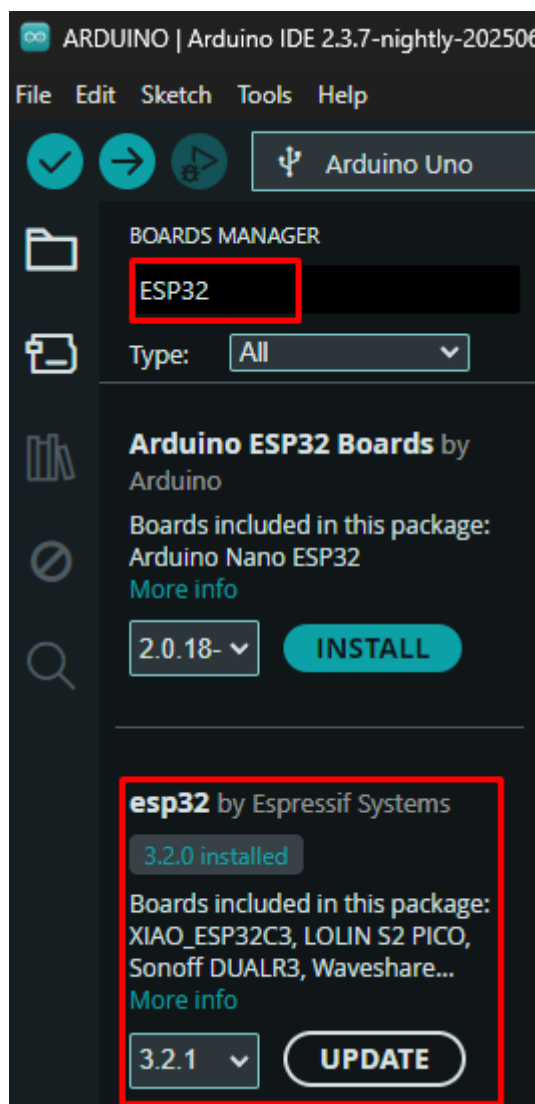


Рис. 1.5. Встановлення підтримки плат на основі ESP32

1.4. Створити новий проект, обравши в меню *File* → *New Sketch*.

1.5. У вікні, яке з'явилося, обрати плату *AI Thinker ESP32-CAM* на відповідному COM-порті. Якщо плати немає в переліку, потрібно знайти і обрати її, як показано на рис. 1.6. Для визначення правильного номера COM-порту необхідно тимчасово підключити плату до USB-порту ПК через адаптер.

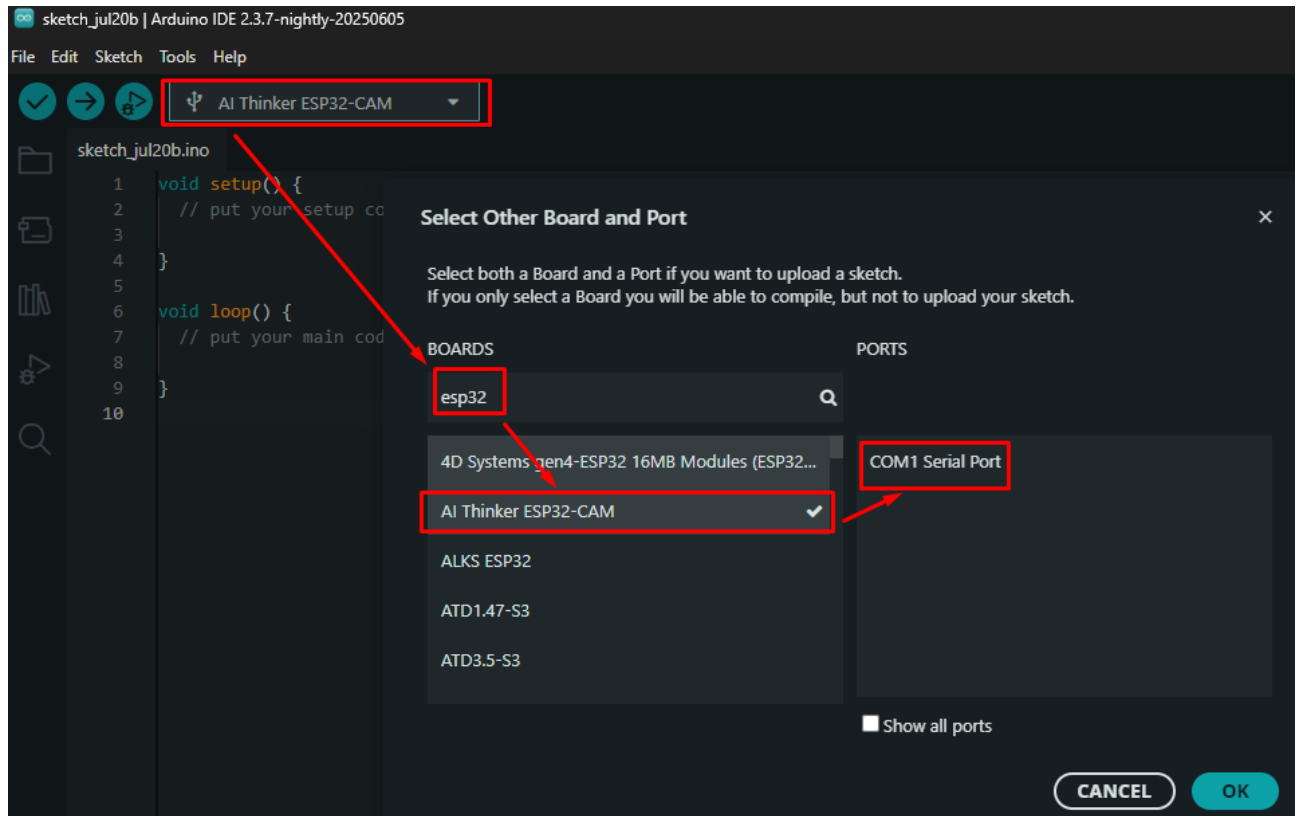


Рис. 1.6. Вибір плати ESP32-CAM

1.6. До папки з проектом (для її коректного створення скетч потрібно зберегти під довільним ім'ям) скопіювати файли *AsyncTCP-main.zip* та *ESPAsyncWebServer-main.zip*, які містять необхідні для роботи асинхронного веб-сервера бібліотеки.

1.7. По черзі підключити дані бібліотеки до проекту, обравши меню *Sketch* → *Include Library* → *Add .ZIP Library*.

1.8. Підключити бібліотеки в коді:

```
#include <WiFi.h>
#include <ESPAsyncWebServer.h>
```

Перша бібліотека *WiFi.h* забезпечує підключення ESP32 до Wi-Fi-мережі. Друга – *ESPAsyncWebServer.h* – дозволяє реалізувати асинхронний веб-сервер, зокрема підтримку WebSocket-з'єднання. Асинхронна модель означає, що ESP32 може обробляти кілька клієнтів без блокування основного циклу *loop()*.

1.9. Оголосити апаратні UART виводи:

```
#define RXD1 15  
#define TXD1 14
```

Ці макроси вказують, які виводи ESP32-Cam використовує для UART-зв'язку з Arduino. Пін 15 приймає дані (RXD1), а пін 14 їх передає (TXD1).

1.10. Оголосити глобальні змінні:

```
char buffer[20];  
int bufferIndex = 0;  
bool ledState = false;
```

buffer – масив, призначений для тимчасового зберігання вхідного рядка символів із UART (від Arduino). Розмір 20 символів достатній для коротких повідомлень.

bufferIndex – лічильник символів у буфері. Зберігає кількість символів, які вже отримано. Вказує, куди буде записуватися наступний байт.

ledState – булева змінна, зарезервована для зберігання стану світлодіода (на даному етапі не використовується, але знадобиться згодом).

1.11. Оголосити налаштування мережі Wi-Fi:

```
const char* ssid = "Name";  
const char* password = "Password";
```

У цих змінних зберігаються назва мережі Wi-Fi (SSID) та пароль. Значення використовуються для з'єднання ESP32 з маршрутизатором.

1.12. Створити веб-сервери:

```
AsyncWebServer server(80);  
AsyncWebSocket ws("/ws");
```

server(80) – створює HTTP-сервер, який слухає порт 80 (стандартний порт для HTTP-з'єднання).

ws("/ws") – створює об'єкт WebSocket, який буде доступний за адресою *ws://<IP-адреса ESP32-Cam>/ws*.

HTTP-сервер необхідний, тому що в архітектурі ESPAsyncWebServer (і взагалі у WebSocket-протоколі) WebSocket завжди стартує як розширення протоколу HTTP, і саме HTTP-сервер відповідає за початкове "рукоштовування" (Handshake), необхідне для встановлення WebSocket-з'єднання.

- 1.13. Написати функцію для передачі текстових команд з ESP32-CAM до Arduino Uno через UART (інтерфейс *Serial1*):

```
void sendToArduino(const char* msg) {  
    Serial1.println(msg);  
    Serial.print("Sent to Arduino -> ");  
    Serial.println(msg);  
}
```

Команда `println(msg)` відправляє повідомлення, яке завершується символом нового рядка (`\n`), що є стандартним способом завершення повідомлень в Arduino. Додатково, ця команда дублюється у консоль (Serial) Arduino IDE, що корисно для відлагодження.

- 1.14. Написати функцію для опрацювання повідомлень, отриманих по WebSocket від ПК:

```
void onWebSocketMessage(AsyncWebSocket *server, AsyncWebSocketClient *client,  
                        AwsFrameInfo *info, String data) {  
    data.trim();  
    if (data == "on" || data == "off") {  
        sendToArduino(data.c_str());  
    }  
}
```

Ця функція викликається щоразу, коли WebSocket-клієнт (в нашому випадку програма на ПК) надсилає повідомлення. Метод `data.trim()` видаляє зайві пробіли й символи нового рядка з початку й кінця рядка. Якщо отримане повідомлення дорівнює "on" або "off", воно передається на Arduino через `sendToArduino()`. Таким чином, ця функція реалізує фільтрацію дозволених команд та безпосередньо відповідає за логіку передачі команд керування.

- 1.15. Написати функцію-обробник подій WebSocket-з'єднання:

```

void onEvent(AsyncWebSocket *server, AsyncWebSocketClient *client,
             AwsEventType type, void *arg, uint8_t *data, size_t len) {
    if (type == WS_EVT_DATA) {
        AwsFrameInfo *info = (AwsFrameInfo*)arg;
        if (info->final && info->index == 0 && info->len == len &&
            info->opcode == WS_TEXT) {
            String msg = "";
            for (size_t i = 0; i < len; i++) msg += (char)data[i];
            onWebSocketMessage(server, client, info, msg);
        }
    }
}

```

Ця функція є головним обробником подій WebSocket-з'єднання. В нашому випадку вона реагує лише на подію *WS_EVT_DATA*, яка сигналізує про надходження нових даних від клієнта. Далі перевіряється, чи:

- це фінальний фрейм (*info->final == true*);
- це перший (і єдиний) фрагмент повідомлення (*info->index == 0*);
- довжина вхідного масиву *data* відповідає заявленій (*info->len == len*);
- повідомлення є текстовим (*info->opcode == WS_TEXT*).

Якщо всі ці умови виконано, виконується цикл по байтах у *data*, в якому формується рядок *msg*. Цей рядок далі обробляється функцією *onWebSocketMessage()*.

1.16. Написати функцію *setup()* для задання початкових налаштувань мікроконтролера:

```

void setup() {
    Serial.begin(115200); // Serial monitor
    Serial1.begin(9600, SERIAL_8N1, RXD1, TXD1); // UART з'єднання з Arduino

    WiFi.begin(ssid, password);
    Serial.print("Connecting to WiFi");
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("\nConnected to WiFi");
    Serial.print("ESP32 IP: ");
    Serial.println(WiFi.localIP());

    ws.onEvent(onEvent);
    server.addHandler(&ws);
    server.begin();
}

```


Функція *setup()* виконується один раз після завантаження мікроконтролера. В ній виконується така послідовність дій:

- 1) Ініціалізуються два UART:
Serial – основний порт для монітора порту;
Serial1 – UART для зв'язку з Arduino.
- 2) Через *WiFi.begin()* ESP32-CAM підключається до бездротової мережі.
- 3) У циклі *while* з перевіркою *WiFi.status()* очікується підключення.
- 4) Після з'єднання виводиться IP-адреса пристрою.
- 5) Додається обробник WebSocket-подій до об'єкта *ws*.
- 6) WebSocket об'єкт додається до HTTP-сервера.
- 7) Сервер запускається методом *server.begin()*.

1.17. Написати основну функцію *loop()*:

```
void loop() {  
  while (Serial1.available()) {  
    char inChar = Serial1.read();  
    if (inChar == '\n') {  
      buffer[bufferIndex] = '\0';  
      Serial.print("Received from Arduino -> ");  
      Serial.println(buffer);  
      bufferIndex = 0;  
      break;  
    } else if (bufferIndex < sizeof(buffer) - 1) {  
      buffer[bufferIndex++] = inChar;  
    }  
  }  
}
```

Ця функція виконується безперервно. Вона перевіряє, чи доступні нові символи у вхідному UART-буфері. Якщо так, вони зчитуються по одному символу.

Якщо символ – новий рядок (*\n*), то вважається, що повідомлення завершено. У цей момент у кінець *buffer* додається термінувальний нуль (*\0*), який перетворює масив символів у рядок. Отриманий рядок виводиться у монітор порту, після чого буфер скидається (індекс обнуляється).

Якщо ж це звичайний символ, і буфер не заповнений, він додається до буфера, а *bufferIndex* збільшується. Таким чином, *loop()* відповідає за приймання повідомлень від Arduino та їх виведення для відлагодження.

- 1.18. Натиснути кнопку *Verify* (👍) для попередньої компіляції та перевірки повного коду програми. Повний код виглядає так:

```
#include <WiFi.h>
#include <ESPAsyncWebServer.h>

#define RXD1 15
#define TXD1 14

char buffer[20];
int bufferIndex = 0;
bool ledState = false;

const char* ssid = "Name";
const char* password = "Password";

AsyncWebServer server(80);
AsyncWebSocket ws("/ws");

void sendToArduino(const char* msg) {
    Serial1.println(msg);
    Serial.print("Sent to Arduino -> ");
    Serial.println(msg);
}

void onWebSocketMessage(AsyncWebSocket *server, AsyncWebSocketClient *client,
    AwsFrameInfo *info, String data) {
    data.trim();
    if (data == "on" || data == "off") {
        sendToArduino(data.c_str());
    }
}

void onEvent(AsyncWebSocket *server, AsyncWebSocketClient *client,
    AwsEventType type, void *arg, uint8_t *data, size_t len) {
    if (type == WS_EVT_DATA) {
        AwsFrameInfo *info = (AwsFrameInfo*)arg;
        if (info->final && info->index == 0 && info->len == len &&
            info->opcode == WS_TEXT) {
            String msg = "";
            for (size_t i = 0; i < len; i++) msg += (char)data[i];
            onWebSocketMessage(server, client, info, msg);
        }
    }
}

void setup() {
    Serial.begin(115200); // Serial monitor
    Serial1.begin(9600, SERIAL_8N1, RXD1, TXD1); // UART з'єднання Arduino

    WiFi.begin(ssid, password);
    Serial.print("Connecting to WiFi");
```


```

while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
Serial.println("\nConnected to WiFi");
Serial.print("ESP32 IP: ");
Serial.println(WiFi.localIP());

ws.onEvent(onEvent);
server.addHandler(&ws);
server.begin();
}

void loop() {
    while (Serial1.available()) {
        char inChar = Serial1.read();
        if (inChar == '\n') {
            buffer[bufferIndex] = '\0';
            Serial.print("Received from Arduino -> ");
            Serial.println(buffer);
            bufferIndex = 0;
            break;
        } else if (bufferIndex < sizeof(buffer) - 1) {
            buffer[bufferIndex++] = inChar;
        }
    }
}
}

```

- 1.19. Підключити плату ESP32-CAM до ПК через USB за допомогою адаптера. Натиснути кнопку *Upload* () для завантаження програми до мікропроцесора. Критерієм правильності буде вивід IP-адреси ESP32-CAM та супутніх повідомлень до консолі *Serial Monitor*.
2. Написати код програми для Arduino Uno для реалізації зв'язку з ESP32-CAM по UART.
 - 2.1. В Arduino IDE створити новий проект, обравши в меню *File* → *New Sketch*.
 - 2.2. Аналогічно до п. 1.5 обрати плату Arduino Uno на відповідному COM-порті.
 - 2.3. Оголосити глобальні змінні:

```

char buffer[20];
int bufferIndex = 0;

```

Призначення даних змінних аналогічне цим самим змінним у коді для ESP32-CAM.

- 2.4. Написати функцію *setup()*:

```

void setup() {
    Serial.begin(9600); // UART

```

```
pinMode(LED_BUILTIN, OUTPUT); // LED
Serial.println("Ready to receive");
Serial.flush();
}
```

Як і для ESP32-CAM, функція *setup()* виконується лише один раз після запуску або перезавантаження мікроконтролера. В ній викликається *Serial.begin(9600)* – ініціалізується UART з швидкістю передачі 9600 біт/с. Цей UART використовується як для зв'язку з ESP32-CAM, так і для програмування мікропроцесора.

Команда *pinMode(LED_BUILTIN, OUTPUT)* встановлює вбудований світлодіод (на більшості Arduino Uno він підключений до порту 13) у режим виходу, тобто мікроконтролер може подавати на нього сигнал.

Serial.println("Ready to receive") надсилає повідомлення до консолі та ESP32-CAM, сигналізуючи про готовність приймати команди.

Serial.flush() чекає завершення передачі всіх даних, щоб впевнитись, що повідомлення дійшло до адресата перед продовженням роботи.

2.5. Написати функцію *loop()*:

```
void loop() {
    if (Serial.available()) {
        char inChar = Serial.read();
        if (inChar == '\n') {
            buffer[bufferIndex] = '\0';

            if (bufferIndex > 0 && buffer[bufferIndex - 1] == '\r') {
                buffer[--bufferIndex] = '\0';
            }

            // Опрацювання команд
            if (strcmp(buffer, "on") == 0) {
                digitalWrite(LED_BUILTIN, HIGH);
            } else if (strcmp(buffer, "off") == 0) {
                digitalWrite(LED_BUILTIN, LOW);
            }

            // Відправка підтвердження
            Serial.print(buffer);
            Serial.println();
            Serial.flush();
            bufferIndex = 0;
        } else if (bufferIndex < sizeof(buffer) - 1) {
            buffer[bufferIndex++] = inChar;
        }
    }
}
```

Функція `loop()` виконується циклічно і безперервно, одразу після завершення `setup()`. У середині `loop()` реалізовано наступний алгоритм прийому і обробки UART-команд:

- 1) Перевірка `if (Serial.available())` з'ясовує, чи є нові символи у вхідному UART-буфері. Якщо є, то один символ зчитується командою `Serial.read()` і зберігається у змінну `inChar`.
- 2) Якщо цей символ є символом нового рядка (`\n`), це означає завершення текстової команди. У такому разі до буфера додається символ завершення рядка (`\0`) для правильного формування C-рядка. Далі виконується перевірка, чи останній символ перед `\n` був поверненням каретки (`\r`). Така ситуація можлива, якщо команда була надіслана з Windows (де кінець рядка має формат `\r\n`). Якщо так, то `\r` замінюється на `\0`, щоб отримати чисту команду.
- 3) Після очищення команди виконується її аналіз за допомогою `strcmp(buffer, "on")` та `strcmp(buffer, "off")`. Якщо команда "on", то подається високий рівень сигналу (*HIGH*) на пін `LED_BUILTIN`, тобто світлодіод вмикається. Якщо команда "off", то на пін подається низький рівень (*LOW*), і світлодіод вимикається.
- 4) Після виконання команди Arduino надсилає назад ту ж саму команду як підтвердження, що вона була оброблена. Цей механізм зворотного зв'язку дозволяє ESP32-CAM переконатися, що команда досягла адресата і була виконана.
- 5) Нарешті, `Serial.flush()` знову забезпечує завершення всіх операцій надсилання перед обнуленням індексу `bufferIndex`, який готує систему до прийому наступної команди.

2.6. Натиснути кнопку *Verify* (🔍) для попередньої компіляції та перевірки повного коду програми. Повний код виглядає так:

```
char buffer[20];
int bufferIndex = 0;

void setup() {
  Serial.begin(9600);
  pinMode(LED_BUILTIN, OUTPUT);
}
```

```

    Serial.println("Ready to receive");
    Serial.flush();
}


void loop() {
    if (Serial.available()) {
        char inChar = Serial.read();
        if (inChar == '\n') {
            buffer[bufferIndex] = '\0';

            if (bufferIndex > 0 && buffer[bufferIndex - 1] == '\r') {
                buffer[--bufferIndex] = '\0';
            }

            // Опрацювання команд
            if (strcmp(buffer, "on") == 0) {
                digitalWrite(LED_BUILTIN, HIGH);
            } else if (strcmp(buffer, "off") == 0) {
                digitalWrite(LED_BUILTIN, LOW);
            }

            // Відправка підтвердження
            Serial.print(buffer);
            Serial.println();
            Serial.flush();
            bufferIndex = 0;
        } else if (bufferIndex < sizeof(buffer) - 1) {
            buffer[bufferIndex++] = inChar;
        }
    }
}
}

```

2.7. Підключити плату Arduino Uno до ПК через USB. Натиснути кнопку *Upload* () для завантаження програми до мікропроцесора. Критерієм правильності буде вивід рядка «Ready to receive» до консолі *Serial Monitor*. Відключити Arduino Uno від ПК.

3. Створити графічний інтерфейс (GUI) програми на ПК.

3.1. Запустити Qt Designer.

3.2. Створити новий інтерфейс, обравши в розділі *templates/forms* шаблон «*Widget*» та натиснувши на кнопку *Create* (рис. 1.7).

3.3. В блоці *Object Inspector* клікнути лівою кнопкою миші на віджет *Form*. Після цього в блоці *Property Editor* змінити значення властивості *windowTitle* на «*ESP32 WebSocket Control*» (рис. 1.8).

3.4. Зберегти створений файл інтерфейсу (.ui) під іменем «*led_control*», натиснувши *Ctrl + S* або натиснувши  на палітрі інструментів.

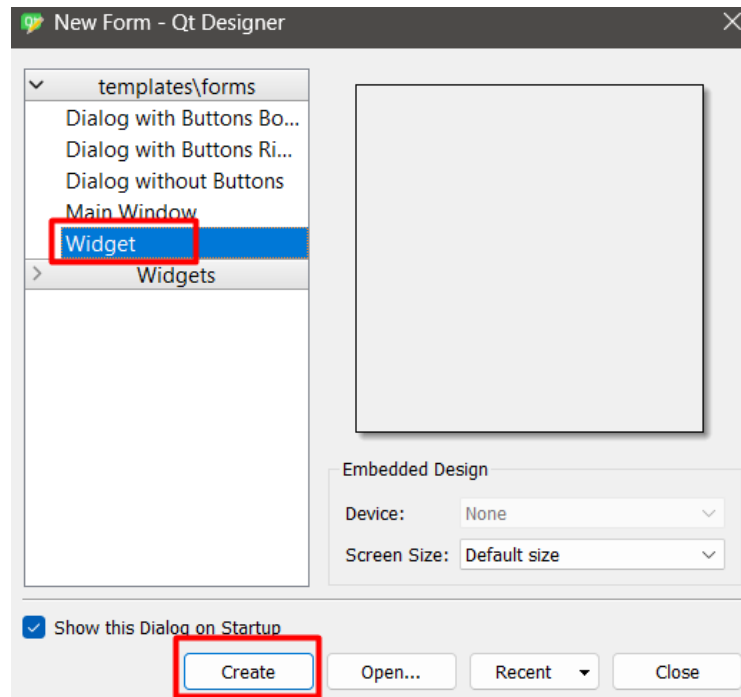


Рис. 1.7. Створення нового графічного інтерфейсу

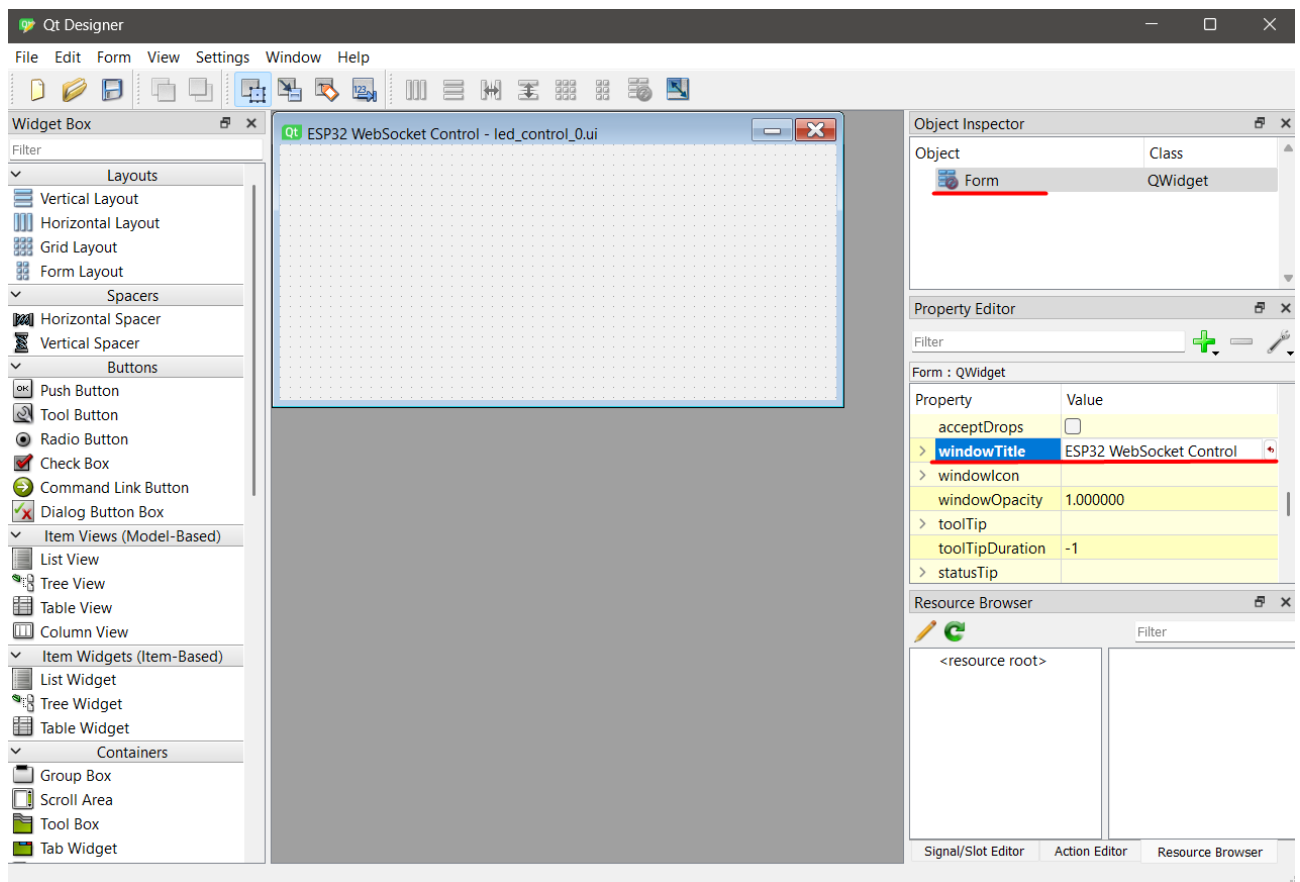


Рис. 1.8. Зміна назви вікна програми

3.5. З блоку *Widget Box* перетягнути у створену форму віджети *Label* (текстова мітка), *PushButton* (кнопка, 2 шт) та *Horizontal Spacer*. Розмістити їх довільно (рис. 1.9).

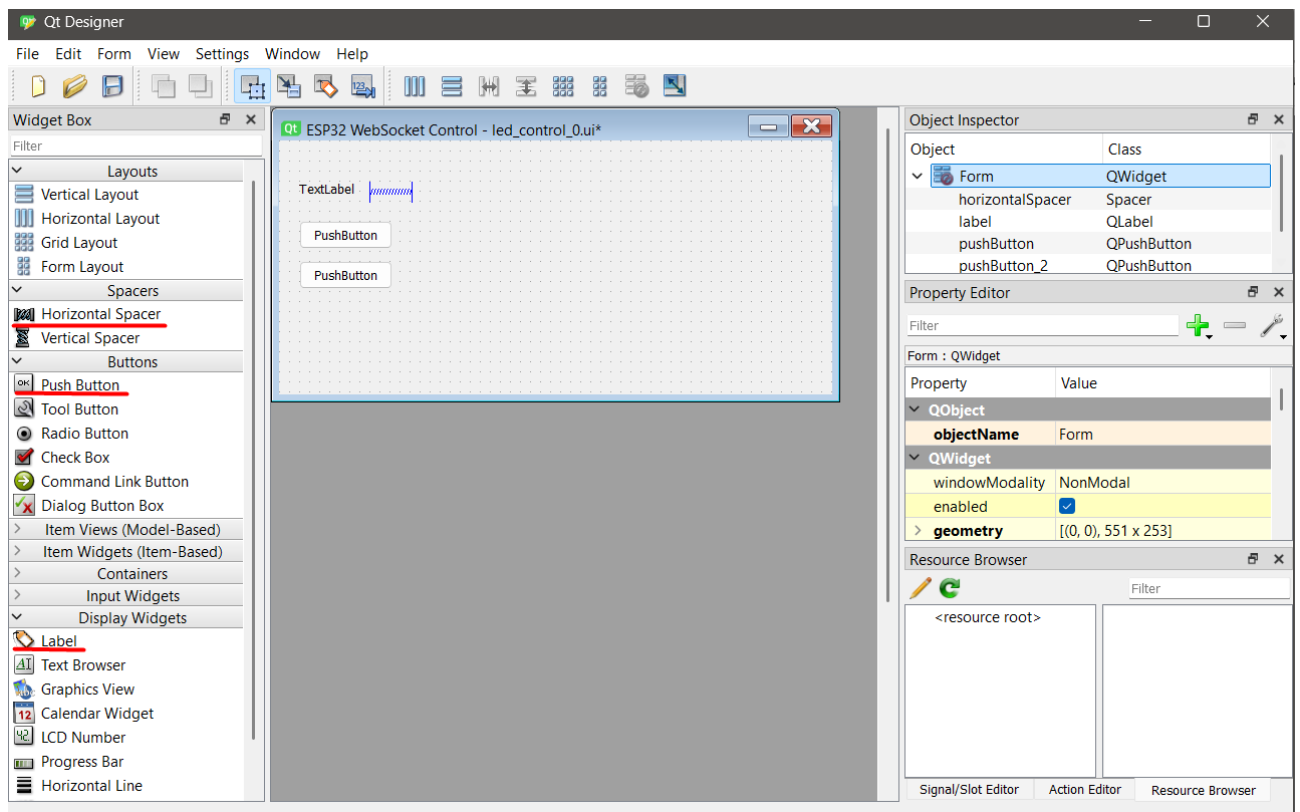


Рис. 1.9. Створення віджетів графічного інтерфейсу

3.6. Виділити одночасно віджети *Label* та *Horizontal Spacer*, затиснувши ліву кнопку миші та обвівши їх рамкою. Після цього натиснути кнопку *Lay Out Horizontally* для групування даних об'єктів за горизонталлю (рис. 1.10).

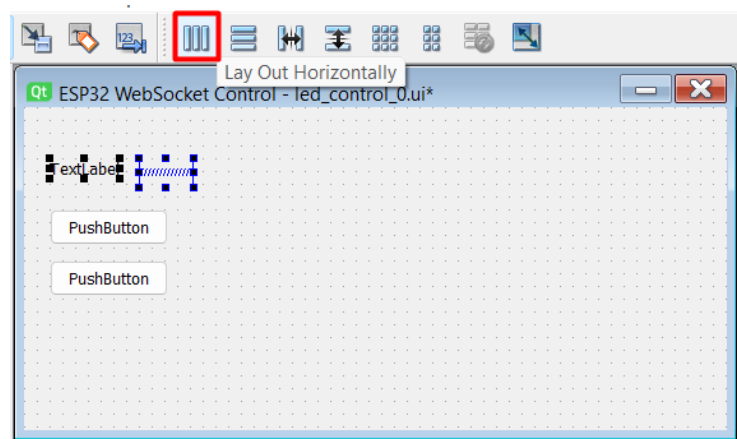



Рис. 1.10. Групування віджетів

3.7. В блоці *Object Inspector* клікнути лівою кнопкою миші на віджет *Form* або просто клікнути лівою кнопкою миші по пустому місцю на полотні інтерфейсу програми. Після цього натиснути кнопку  *Lay Out Vertically*. Тепер всі віджети будуть автоматично змінювати свої розміри зі зміною розміру вікна (рис. 1.11). Таким чином, комбінуючи різні способи групування, можна створювати різноманітні блоки інтерфейсу програми з прив'язкою один до одного, один відносно одного або іншими типами розміщення. Це дозволяє керувати розташуванням і розмірами віджетів під час масштабування вікна програми. Віджети типу *Horizontal Spacer* або *Vertical Spacer* дозволяють «притискати» віджети (або групи віджетів) до певних країв вікна програми або інших віджетів, щоб зберігати їхню позицію під час масштабування.

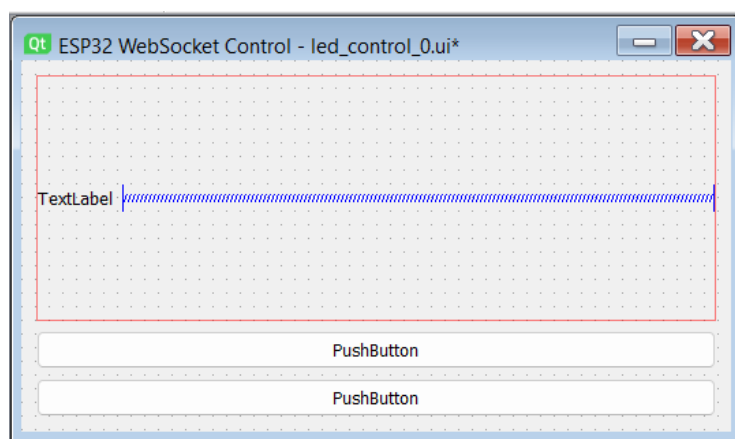


Рис. 1.11. Глобальне групування віджетів за вертикаллю

3.8. В блоці *Property Editor* встановити налаштування властивостей віджетів так, як вказано у таблиці 1.1.

Таблиця 1.1. Перелік віджетів графічного інтерфейсу та їх властивостей

Тип віджету	Назва (objectName)	Додаткові властивості
Label	status_label	<i>minimumSize</i> : width=150, height=50 <i>maximumSize</i> : width=150 <i>text</i> : "Не підключено"
PushButton	pushButton_on	<i>text</i> : "Включити світлодіод"
PushButton	pushButton_off	<i>text</i> : "Вимкнути світлодіод"

3.9. Загальний вигляд інтерфейсу після внесених змін має приблизно відповідати такому, який показано на рис. 1.12.

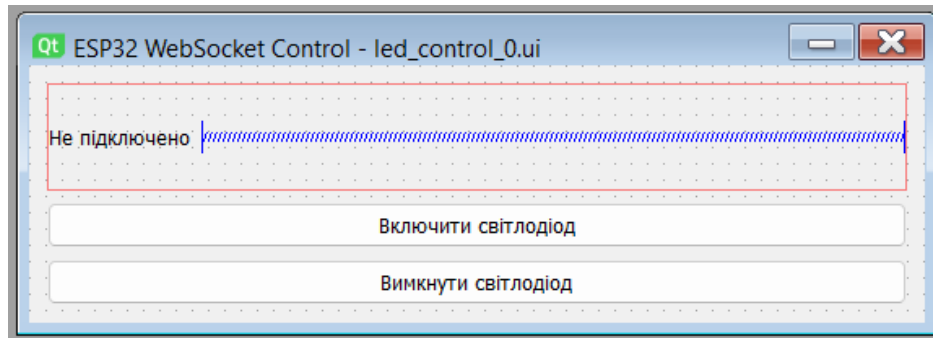


Рис. 1.12. Загальний вигляд GUI після налаштувань віджетів

***Примітка:** в даній лабораторній роботі наводяться конкретні вимоги до графічного інтерфейсу. В усіх наступних роботах буде надаватись лише перелік віджетів та їхніх основних властивостей. Конкретне розміщення, розміри та інші параметри віджетів у подальшому можна обирати самостійно. Це дасть змогу кожному створити свій унікальний дизайн.*

- 3.10. Зберегти файл з інтерфейсом. Запам'ятати його місце збереження.
4. Написати Python-програму для роботи зі створеним графічним інтерфейсом для керування світлодіодом на платі ESP32 через WebSocket-з'єднання.
- 4.1. Відкрити будь-який редактор Python-коду або середовище розробки (наприклад, PyCharm) і створити новий .py-файл або проект.
- 4.2. Імпортувати необхідні бібліотеки:

```
import sys
import asyncio
import websockets
from PyQt5 import uic, QtGui
from PyQt5.QtWidgets import QApplication, QWidget
from qasync import QEventLoop, asyncSlot
```

В цьому коді:

`sys` – потрібна для ініціалізації GUI-програми;

asyncio – асинхронний фреймворк, що забезпечує неблокуюче виконання операцій;

websockets – бібліотека для роботи з WebSocket-протоколом;

PyQt5 – набір модулів для створення графічного інтерфейсу;

uic – модуль для завантаження *.ui*-файлів (створених у Qt Designer);

qasync – інструмент для інтеграції *PyQt* і *asyncio*.

4.3. Створити константу, яка містить адресу WebSocket-сервера на ESP32. Цю адресу потрібно замінити на актуальну IP-адресу пристрою в мережі:

```
ESP32_WS_URL = "ws://192.168.31.81/ws" # Замінити на свою адресу
```

4.4. Створити клас для головного вікна програми:

```
class LEDControl(QWidget):
    def __init__(self):
        super().__init__()
        uic.loadUi("led_control.ui", self)
```

Цей клас наслідує *QWidget*, що дозволяє створювати й відображати графічний інтерфейс. Після виклику конструктора батьківського класу відбувається завантаження користувацького інтерфейсу з файлу *led_control.ui*. Цей файл був створений у Qt Designer на попередньому кроці. Для коректного завантаження треба або вказати повний шлях до файлу, або розмістити його у папці з Python-проектом (в одній папці із файлом *.py* з основним кодом програми).

4.5. Завантажити необхідні елементи інтерфейсу:

```
self.status_label = self.findChild(type(self.findChild(QWidget,
"status_label")), "status_label")
self.button_on = self.findChild(type(self.findChild(QWidget,
"pushButton_on")), "pushButton_on")
self.button_off = self.findChild(type(self.findChild(QWidget,
"pushButton_off")), "pushButton_off")
```

Тут за допомогою методу *findChild()* відбувається пошук і збереження посилань на елементи GUI за їхніми іменами. Це дозволяє програмно змінювати їхній вміст або обробляти події (наприклад, натискання на кнопку).

4.6. Прив'язати до кнопок функцію-обробник натискань:

```
self.button_on.clicked.connect(lambda: self.send_command("on"))
self.button_off.clicked.connect(lambda: self.send_command("off"))
```

Тепер у разі натискання кнопки *button_on* буде викликана асинхронна функція *send_command("on")*, яка відправить відповідну команду через WebSocket.

4.7. Оголосити інші необхідні змінні:

```
self.ws = None
self.is_running = True
asyncio.ensure_future(self.connect_ws())
```

Змінна *self.ws* буде містити об'єкт WebSocket-з'єднання. *self.is_running* використовується як логічний прапорець для контролю виконання фонових завдань. А за допомогою *asyncio.ensure_future()* запускається фонове з'єднання з ESP32-CAM.

4.8. Створити метод для забезпечення WebSocket-з'єднання:

```
async def connect_ws(self):
    while self.is_running:
        try:
            self.ws = await websockets.connect(ESP32_WS_URL)
            self.status_label.setText("Connected to ESP32")
            async for message in self.ws:
                pass
        except Exception as e:
            self.status_label.setText(f"Error: {e}")
            self.ws = None
            await asyncio.sleep(1)
```

Цей метод є асинхронним і працює в циклі *while*, який виконується доти, доки програма активна. Спочатку виконується спроба встановлення WebSocket-з'єднання з ESP32-CAM. У разі успіху оновлюється мітка статусу *status_label*: «Connected to ESP32». Після встановлення з'єднання виконується цикл *async for message in self.ws*, який постійно чекає на повідомлення від ESP32-CAM.

У разі виникнення будь-якої помилки (наприклад, відсутнє з'єднання або ESP32 вимкнено), на мітці статусу виводиться повідомлення про помилку. Потім програма робить паузу на 1 секунду і намагається повторити підключення.

4.9. Створити метод для надсилання команд до ESP32-CAM через WebSocket:

```

@asyncSlot()
async def send_command(self, command):
    if self.ws and self.ws.open:
        try:
            await self.ws.send(command)
            self.status_label.setText(f"LED state: {command}")
        except Exception as e:
            self.status_label.setText(f"Error while sending: {e}")
    else:
        self.status_label.setText("WebSocket isn't connected!")

```

Цей асинхронний метод позначений декоратором `@asyncSlot()`, щоб його можна було використовувати в графічному інтерфейсі. Якщо WebSocket-з'єднання існує (`self.ws`) і воно активне (`self.ws.open`), команда надсилається через `await self.ws.send(command)`. Якщо ж з'єднання неактивне або відбулася помилка при надсиланні, на мітці `status_label` відображається повідомлення про помилку.

4.10. Створити метод для коректного завершення програми:

```

def closeEvent(self, event):
    self.is_running = False
    if self.ws:
        asyncio.ensure_future(self.ws.close())
    event.accept()

```

Цей метод викликається автоматично при закритті вікна програми. Він встановлює прапорець `self.is_running` у `False`, що зупиняє цикл спроб повторного з'єднання. Якщо WebSocket-з'єднання існує, воно закриється асинхронно. Закриття вікна підтверджується методом `event.accept()`.

4.11. Створити головну функцію для запуску програми:

```

def main():
    app = QApplication(sys.argv)
    loop = QEventLoop(app)
    asyncio.set_event_loop(loop)

    window = LEDControl()
    window.show()

    with loop:
        loop.run_forever()

if __name__ == "__main__":
    main()

```

Функція *main()* запускає всю програму. Створюється об'єкт PyQt-додатку, запускається асинхронний цикл *QEventLoop*, а також встановлюється його інтеграція з *asyncio*. Потім створюється об'єкт класу *LEDControl*, який відображається на екрані. Основний цикл подій запускається командою *loop.run_forever()*.

Блок *if __name__ == "__main__"* гарантує, що головна функція *main()* буде викликана лише в разі, якщо файл запущено напряму, а не імпортовано з іншого модуля.

4.12. Провести тестовий запуск програми. Повний код програми:

```
import sys
import asyncio
import websockets
from PyQt5 import uic, QtGui
from PyQt5.QtWidgets import QApplication, QWidget
from qasync import QEventLoop, asyncSlot

ESP32_WS_URL = "ws://192.168.31.81/ws" # Заміни на свою адресу

class LEDControl(QWidget):
    def __init__(self):
        super().__init__()
        uic.loadUi("led_control.ui", self)

        self.status_label = self.findChild(type(self.findChild(QWidget,
"status_label")), "status_label")
        self.button_on = self.findChild(type(self.findChild(QWidget,
"pushButton_on")), "pushButton_on")
        self.button_off = self.findChild(type(self.findChild(QWidget,
"pushButton_off")), "pushButton_off")

        self.button_on.clicked.connect(lambda: self.send_command("on"))
        self.button_off.clicked.connect(lambda: self.send_command("off"))

        self.ws = None
        self.is_running = True
        asyncio.ensure_future(self.connect_ws())

    async def connect_ws(self):
        while self.is_running:
            try:
                self.ws = await websockets.connect(ESP32_WS_URL)
                self.status_label.setText("Connected to ESP32")
                async for message in self.ws:
                    pass
            except Exception as e:
                self.status_label.setText(f"Error: {e}")
                self.ws = None
                await asyncio.sleep(1)

    @asyncSlot()
    async def send_command(self, command):
        if self.ws and self.ws.open:
```

```

try:
    await self.ws.send(command)
    self.status_label.setText(f"LED state: {command}")
except Exception as e:
    self.status_label.setText(f"Error while sending: {e}")
else:
    self.status_label.setText("WebSocket isn't connected!")

def closeEvent(self, event):
    self.is_running = False
    if self.ws:
        asyncio.ensure_future(self.ws.close())
    event.accept()

def main():
    app = QApplication(sys.argv)
    loop = QEventLoop(app)
    asyncio.set_event_loop(loop)

    window = LEDControl()
    window.show()

    with loop:
        loop.run_forever()

if __name__ == "__main__":
    main()

```

У разі успішного запуску має відкритись головне вікно програми. Після цього його можна тимчасово закрити.

- Зібрати схему, показану на рис. 1.13, та перевірити правильність роботи системи. Для цього підключити всі компоненти до живлення, запустити GUI та перевірити результат встановлення з'єднання з ESP32-CAM. Для перевірки правильності передачі і виконання команд спробувати натиснути кнопки «Включити світлодіод» та «Вимкнути світлодіод», при цьому візуально відслідковуючи результат на платі Arduino UNO.

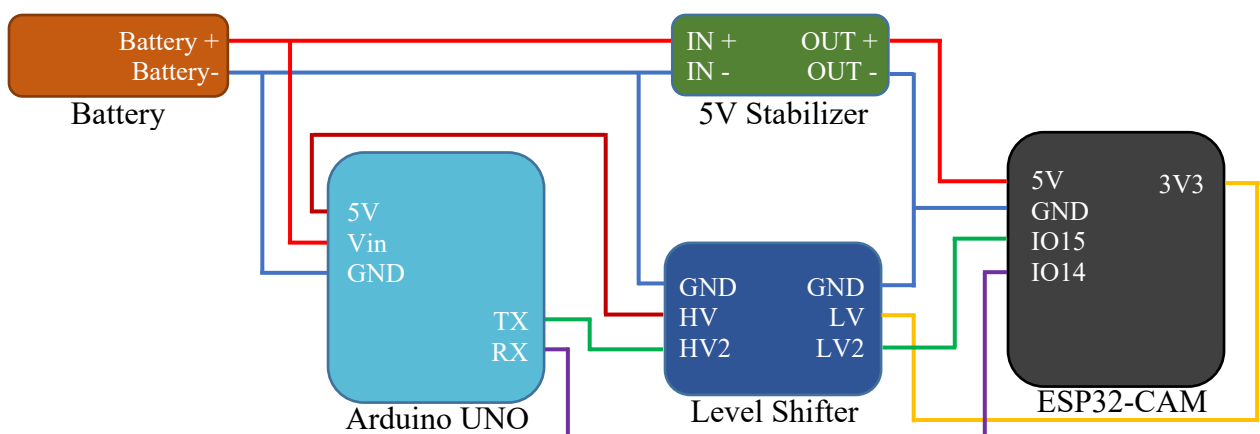


Рис. 1.13. Схема підключення компонентів

6. Оновити код для ESP32-CAM так, щоб передавати інформацію щодо логування не лише на серійний монітор, а й через WebSocket на GUI.

6.1. Додати нову функцію для надсилання логів:

```
void sendLog(String msg) {  
    Serial.println(msg);           // Стандартний UART лог  
    ws.textAll(msg);               // Надсилає у всі WebSocket-клієнти  
}
```

Ця функція реалізує централізовану систему логування. Вона приймає текстове повідомлення *msg* (типу *String*) і виконує дві основні дії: по-перше, виводить це повідомлення у серійний монітор через *Serial.println*, по-друге – надсилає те саме повідомлення через WebSocket на підключений ПК. Для цього використовується метод *ws.textAll(msg)*, який розсилає текст усім активним WebSocket-з'єднанням.

6.2. Модифікувати функцію *sendToArduino()*:

```
void sendToArduino(const char* msg) {  
    Serial1.println(msg);  
    sendLog(String("Sent to Arduino -> ") + msg);  
}
```

У новій реалізації функції відбулася заміна логування. Раніше повідомлення про надсилання команди на Arduino виводилось безпосередньо через *Serial.print()*. Тепер це делеговано функції *sendLog()*, яка не лише виведе повідомлення на Serial Monitor, а й передасть його на ПК через WebSocket. Таким чином, користувач може бачити, які саме команди були відправлені, навіть не маючи доступу до порту UART. Формування повідомлення здійснюється шляхом об'єднання фрази "Sent to Arduino -> " з переданим текстом *msg*. Усі перетворення виконуються в об'єкті типу *String*.

6.3. Змінити обробку отриманих UART-повідомлень у *loop()*:

```
if (inChar == '\n') {  
    buffer[bufferIndex] = '\0';  
    sendLog(String("Received from Arduino -> ") + buffer);  
    bufferIndex = 0;  
    break;  
}
```

Після того як повне повідомлення від Arduino було зчитано, раніше воно виводилося лише на серійний монітор. Тепер використовується вже

згадана універсальна функція `sendLog()`. Формування повідомлення знову виконується через конкатенацію `String("Received from Arduino -> ") + buffer`, де `buffer` — це символьний масив, який містить текст, надісланий Arduino (наприклад, підтвердження "on" або "off").

6.4. Натиснути кнопку *Verify* (🔍) для попередньої компіляції та перевірки повного коду програми. Повний код виглядає так:

```
#include <WiFi.h>
#include <ESPAsyncWebServer.h>

#define RXD1 15
#define TXD1 14

char buffer[20];
int bufferIndex = 0;
bool ledState = false; // Поточний стан світлодіода: false = off, true = on

const char* ssid = "Mystery";
const char* password = "BZ343MJIPH78T01SL";

AsyncWebServer server(80);
AsyncWebSocket ws("/ws");

void sendLog(String msg) {
    Serial.println(msg); // Стандартний UART лог
    ws.textAll(msg);     // Надсилає у всі WebSocket-клієнти
}

void sendToArduino(const char* msg) {
    Serial1.println(msg);
    sendLog(String("Sent to Arduino -> ") + msg);
}

void onWebSocketMessage(AsyncWebSocket *server, AsyncWebSocketClient *client,
                        AwsFrameInfo *info, String data) {
    data.trim();
    if (data == "on" || data == "off") {
        sendToArduino(data.c_str());
    }
}

void onEvent(AsyncWebSocket *server, AsyncWebSocketClient *client,
             AwsEventType type, void *arg, uint8_t *data, size_t len) {
    if (type == WS_EVT_DATA) {
        AwsFrameInfo *info = (AwsFrameInfo*)arg;
        if (info->final && info->index == 0 && info->len == len &&
            info->opcode == WS_TEXT) {
            String msg = "";
            for (size_t i = 0; i < len; i++) msg += (char)data[i];
            onWebSocketMessage(server, client, info, msg);
        }
    }
}
```

```


void setup() {
    Serial.begin(115200); // Serial monitor
    Serial1.begin(9600, SERIAL_8N1, RXD1, TXD1); // UART communication with
    Arduino

    WiFi.begin(ssid, password);
    Serial.print("Connecting to WiFi");
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("\nConnected to WiFi");
    Serial.print("ESP32 IP: ");
    Serial.println(WiFi.localIP());

    ws.onEvent(onEvent);
    server.addHandler(&ws);
    server.begin();
}

void loop() {
    while (Serial1.available()) {
        char inChar = Serial1.read();
        if (inChar == '\n') {
            buffer[bufferIndex] = '\0';
            sendLog(String("Received from Arduino -> ") + buffer);
            bufferIndex = 0;
            break;
        } else if (bufferIndex < sizeof(buffer) - 1) {
            buffer[bufferIndex++] = inChar;
        }
    }
}

```

- 6.5. Вимкнути живлення загальної схеми. Від'єднати лінію Tx ESP32-CAM від RX Arduino Uno.
- 6.6. Підключити плату ESP32-CAM до ПК через USB за допомогою адаптера. Натиснути кнопку Upload () для завантаження програми до мікропроцесора. У випадку коректного завантаження, відключити ESP32-CAM від ПК та відновити цілісність схеми (рис. 1.13).
7. Внести зміни до GUI для додавання можливості відображення логів. Додатково створити графічний індикатор, який буде відображати поточний статус світлодіода у вигляді графічної іконки.
- 7.1. Відкрити Qt Designer та додати до інтерфейсу віджети з параметрами, вказаними у таблиці 1.2. Розмістити віджети довільно або за зразком, показаним на рис. 1.14. Зберегти внесені зміни.

Таблиця 1.2. Опис доданих віджетів графічного інтерфейсу

Тип віджету	Назва (objectName)	Додаткові властивості
Label	led_indicator	<i>minimumSize</i> : width=32, height=32 <i>iconSize</i> : width=24, height=24 <i>text</i> : "" (порожній)
PlainTextEdit	log_view	<i>wordWrap</i> : False <i>readOnly</i> : true <i>horizontalScrollBarPolicy</i> : AsNeeded <i>font</i> : Monospace <i>scrollBarsEnabled</i> : true

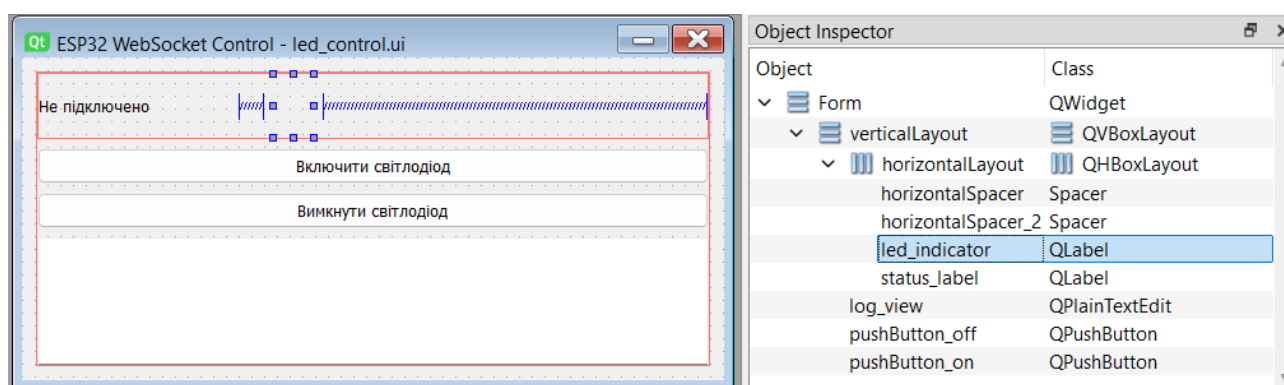


Рис. 1.14. Зразок оновленого інтерфейсу

7.2. Додати до папки з Python-кодом дві іконки: `led_on.png` (зображення яскравої лампочки) та `led_off.png` (тьмяне зображення лампочки).

7.3. Оновити код програми з GUI. У конструкторі класу *LEDControl* додати два нових елементи:

```
self.led_indicator = self.findChild(type(self.findChild(QWidget,
"led_indicator")), "led_indicator")
self.log_view = self.findChild(type(self.findChild(QWidget,
"log_view")), "log_view")
```

Перший елемент – *led_indicator* – призначений для візуального відображення стану світлодіода у вигляді зображення. Це *QLabel*, у який буде вставлятися зображення відповідної піктограми (наприклад, `led_on.png` або `led_off.png`), що символізує увімкнений чи вимкнений стан світлодіода.

Другий елемент – *log_view* – слугує для виведення журналу повідомлень, які надходять від ESP32-CAM. Його типом має бути *QPlainTextEdit*, що дозволяє виводити текстову інформацію в багаторядковому вигляді.

7.4. Додати новий метод *set_led_image()*:

```
def set_led_image(self, state):  
    pixmap = QtGui.QPixmap(f"led_{state}.png")  
    self.led_indicator.setPixmap(pixmap)
```

Цей метод оновлює зображення світлодіода на індикаторі залежно від стану, який задається параметром *state*. Значенням *state* має бути рядок "on" або "off". Метод завантажує відповідний PNG-файл (наприклад, *led_on.png*) як об'єкт *QPixmap* і встановлює його у віджет *led_indicator* за допомогою *setPixmap()*. Таким чином, кожне натискання кнопки або зміна стану світлодіода супроводжується візуальною індикацією.

7.5. Додати новий метод *append_log()*:

```
def append_log(self, message):  
    self.log_view.appendPlainText(message.strip())  
self.log_view.verticalScrollBar().setValue(self.log_view.verticalScrollBar()  
.maximum())
```

Цей метод реалізує виведення текстових повідомлень у полі журналу (елемент *log_view*). Кожне нове повідомлення додається в кінець поля за допомогою *appendPlainText()*, а прокрутка автоматично переміщується до останнього рядка, щоб нове повідомлення було одразу видно користувачу. Вхідний параметр *message* може містити повідомлення, отримане від ESP32-CAM через WebSocket (наприклад, підтвердження "on" або "off"), або будь-яке інше діагностичне повідомлення.

7.6. Змінити частину методу *connect_ws()*:

```
async def connect_ws(self):  
    while self.is_running:  
        try:  
            self.ws = await websockets.connect(ESP32_WS_URL)  
            self.status_label.setText("Connected to ESP32")  
            async for message in self.ws:  
                self.append_log(message)
```

Стара версія методу *connect_ws()* не виконувала жодної обробки отриманих WebSocket-повідомлень. Тепер обробка реальна. Це означає,

що кожне повідомлення, яке надходить через WebSocket з ESP32, автоматично передається у `append_log()` і з'являється в журналі подій. Таким чином, користувач може в режимі реального часу бачити відповіді від ESP32-CAM, включно з тими, які генерує Arduino (наприклад, підтвердження про зміну стану світлодіода).

7.7. Доповнити метод `send_command()`:

```
@asyncSlot()
async def send_command(self, command):
    if self.ws and self.ws.open:
        try:
            await self.ws.send(command)
            self.status_label.setText(f"LED state: {command}")
            self.set_led_image("on" if command == "on" else "off")
        except Exception as e:
            self.status_label.setText(f"Error while sending: {e}")
    else:
        self.status_label.setText("WebSocket isn't connected")
```

Тепер виклик `self.set_led_image()` забезпечує оновлення зображення стану світлодіода одразу після натискання кнопки. Наприклад, при натисканні кнопки "Включити", буде викликано `set_led_image("on")`, що замінить піктограму на зображення увімкненого світлодіода.

7.8. Провести тестовий запуск програми. Повний код програми:

```
import sys
import asyncio
import websockets
from PyQt5 import uic, QtGui
from PyQt5.QtWidgets import QApplication, QWidget
from qasync import QEventLoop, asyncSlot

ESP32_WS_URL = "ws://192.168.31.81/ws" # Заміни на свій

class LEDControl(QWidget):
    def __init__(self):
        super().__init__()
        uic.loadUi("led_control.ui", self)

        self.status_label = self.findChild(type(self.findChild(QWidget, "status_label")), "status_label")
        self.button_on = self.findChild(type(self.findChild(QWidget, "pushButton_on")), "pushButton_on")
        self.button_off = self.findChild(type(self.findChild(QWidget, "pushButton_off")), "pushButton_off")
        self.led_indicator = self.findChild(type(self.findChild(QWidget, "led_indicator")), "led_indicator")
        self.log_view = self.findChild(type(self.findChild(QWidget, "log_view")), "log_view")

        # Початковий стан лампочки — вимкнена
        self.set_led_image("off")
```

```

        self.button_on.clicked.connect(lambda: self.send_command("on"))
        self.button_off.clicked.connect(lambda: self.send_command("off"))

        self.ws = None
        self.is_running = True
        asyncio.ensure_future(self.connect_ws())

    def set_led_image(self, state):
        pixmap = QtGui.QPixmap(f"led_{state}.png")
        self.led_indicator.setPixmap(pixmap)

    def append_log(self, message):
        self.log_view.appendPlainText(message.strip())
self.log_view.verticalScrollBar().setValue(self.log_view.verticalScrollBar().
maximum())

    async def connect_ws(self):
        while self.is_running:
            try:
                self.ws = await websockets.connect(ESP32_WS_URL)
                self.status_label.setText("Connected to ESP32")
                async for message in self.ws:
                    self.append_log(message)
            except Exception as e:
                self.status_label.setText(f"Error: {e}")
                self.ws = None
                await asyncio.sleep(1)

    @asyncSlot()
    async def send_command(self, command):
        if self.ws and self.ws.open:
            try:
                await self.ws.send(command)
                self.status_label.setText(f"LED state: {command}")
                self.set_led_image("on" if command == "on" else "off")
            except Exception as e:
                self.status_label.setText(f"Error while sending: {e}")
        else:
            self.status_label.setText("WebSocket isn't connected")

    def closeEvent(self, event):
        self.is_running = False
        if self.ws:
            asyncio.ensure_future(self.ws.close())
        event.accept()

def main():
    app = QApplication(sys.argv)
    loop = QEventLoop(app)
    asyncio.set_event_loop(loop)

    window = LEDControl()
    window.show()

    with loop:
        loop.run_forever()

if __name__ == "__main__":
    main()

```

8. Підключити всі компоненти до живлення, перезапустити GUI та перевірити результат роботи, натискаючи кнопки управління світлодіодом та відслідковуючи зміни на платі Arduino UNO та у графічному інтерфейсі.

Контрольні запитання

1. Опишіть основні функції модуля ESP32-CAM у контексті системи дистанційного керування світлодіодом. Які апаратні інтерфейси цього модуля використовуються в лабораторній роботі та як вони сприяють реалізації зв'язку?
2. Поясніть принцип роботи WebSocket-протоколу в порівнянні з HTTP. Чому в даній лабораторній роботі WebSocket є більш ефективним для реалізації двостороннього зв'язку між комп'ютером і ESP32-CAM?
3. Яка роль плати Arduino Uno в системі дистанційного керування? Опишіть, як плата обробляє команди і як забезпечується зворотний зв'язок із ESP32-CAM.
4. Опишіть послідовність дій у функції *loop()* коду для ESP32-CAM. Як ця функція забезпечує обробку повідомлень від Arduino Uno та їх передачу до GUI через WebSocket?
5. Поясніть, як у коді для Arduino Uno реалізується обробка команд "on" і "off". Яке значення має перевірка на символи `\r` і `\n` у вхідному буфері?
6. Опишіть процес створення графічного інтерфейсу в Qt Designer. Яким чином використання Horizontal Spacer і групування віджетів впливає на адаптивність інтерфейсу при зміні розміру вікна?
7. Опишіть, як у Python-програмі забезпечується стійкість до помилок підключення до WebSocket-сервера. Які дії виконуються в методі *connect_ws()* у разі виникнення помилки, і як це відображається в інтерфейсі користувача?

ДИСТАНЦІЙНЕ КЕРУВАННЯ РОБОТОМ

Мета: Ознайомлення з основами дистанційного керування роботом-автомобілем за допомогою Wi-Fi з'єднання та графічного інтерфейсу на PyQt. Вивчення методів передачі потокового відео з мікроконтролера ESP32-CAM із використанням Wi-Fi.

Теоретичні відомості

Система, описана в цій частині лабораторної роботи, призначена для дистанційного керування рухом робота-автомобіля та передачі відеопотоку з камери через WiFi-з'єднання. Вона демонструє принципи інтеграції апаратних і програмних компонентів для реалізації керування в реальному часі, що є важливим аспектом роботи автономних роботизованих систем. Система забезпечує віддалене управління рухом робота та його візуальний контроль через відеотрансляцію,

Як і в попередній частині, система складається з трьох основних компонентів (рис. 1.15): мікроконтролера ESP32-CAM, плати Arduino Uno з моторним шилдом L298P для керування двигунами і програмного забезпечення на базі Python із графічним інтерфейсом, створеним за допомогою бібліотеки PyQt. ESP32-CAM виконує кілька ключових функцій: забезпечує підключення до WiFi-мережі, передає відеопотік із вбудованої камери OV2640 у форматі MJPEG і надсилає команди керування до Arduino через послідовний інтерфейс UART. Плата Arduino Uno відповідає за керування двома двигунами, які забезпечують рух робота вперед, назад, повороти ліворуч або праворуч, а також зупинку. Програмне забезпечення на ПК надає графічний інтерфейс із кнопками для запуску та зупинки відеотрансляції, а також обробляє команди управління рухом робота через клавіші W, A, S, D.

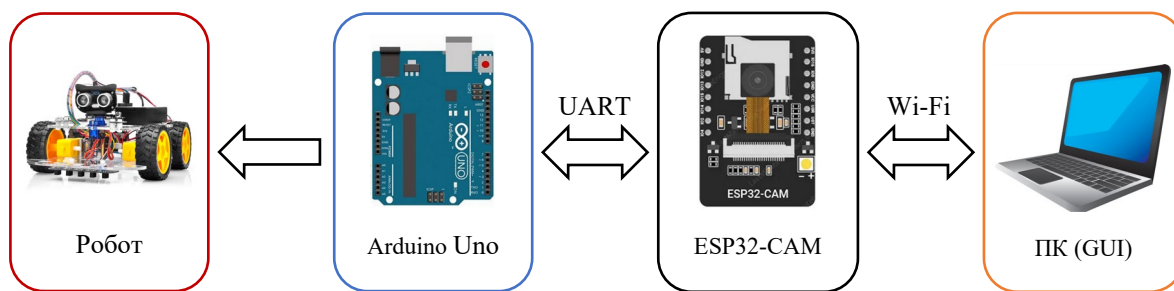


Рис. 1.15. Структурна схема системи дистанційного керування роботом-автомобілем

Принцип роботи системи базується на взаємодії між компонентами в режимі реального часу. ESP32-CAM ініціалізує WiFi-з'єднання та запускає два сервери: WebSocket-сервер на порту 81 для обробки команд керування та HTTP-сервер на порту 80 для передачі відеопотоку. Графічний інтерфейс на комп'ютері встановлює WebSocket-з'єднання з ESP32-CAM, через яке надсилаються команди, такі як "w" (рух вперед), "a" (поворот ліворуч), "s" (рух назад), "d" (поворот праворуч) або "halt" (зупинка). Команди відправляються у разі натискання відповідних клавіш на клавіатурі. Ці команди передаються через UART до плати Arduino Uno, яка запускає двигуни відповідно до отриманої інструкції. При активації відеотрансляції ESP32-CAM надсилає відеопотік у форматі MJPEG, який відображається в інтерфейсі користувача. Програма також відображає журнали подій (логи), що інформують про стан системи, наприклад, про підключення до WebSocket або надіслані команди. При відпусканні клавіш керування рухом на клавіатурі програма автоматично надсилає команду зупинки робота, що забезпечує надійне керування.

Новим важливим компонентом системи є моторний шилд L298P (рис. 1.16). Це модуль розширення для плат Arduino, призначений для керування двигунами постійного струму, серводвигунами або одним кроковим двигуном. Він базується на інтегральній схемі L298P, яка є подвійним повним Н-мостом, що забезпечує керування швидкістю та напрямком обертання двигунів. У лабораторній роботі шилд L298P використовується разом із платою Arduino Uno

для обробки команд, отриманих від ESP32-CAM через UART, і керування двома двигунами робота. Шилд забезпечує надійне електричне з'єднання між Arduino та двигунами, а також захист від зворотного струму, що робить його ефективним і безпечним рішенням для робототехнічних проєктів.

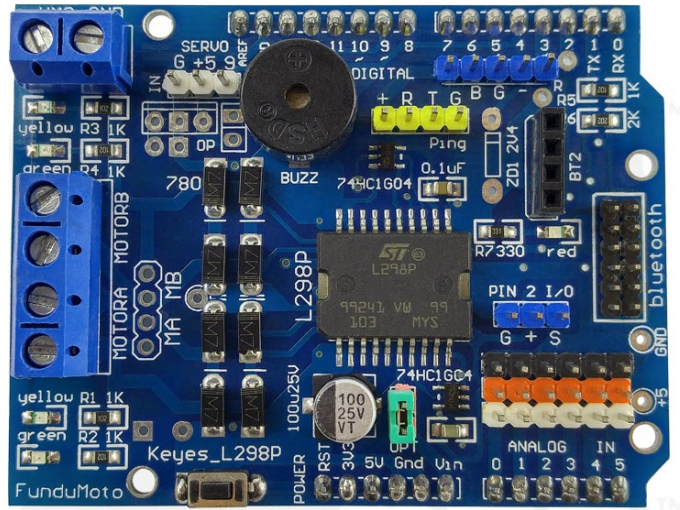


Рис. 1.16. Шилд L298P

Шилд L298P підтримує напругу живлення двигунів від 4.8 В до 35 В (зазвичай 6.5–12 В для оптимальної роботи), що дозволяє використовувати широкий спектр двигунів. Максимальний струм на канал становить 2 А (з піковим значенням до 3 А для коротких імпульсів), що забезпечує достатню потужність для більшості робототехнічних застосувань. Логічна частина шилду працює від 5 В, яке зазвичай подається від Arduino, але для живлення двигунів рекомендується використовувати окремий зовнішній блок живлення через гвинтові клеми (VMS), оскільки струм двигунів часто перевищує можливості USB-порту. Шилд використовує чотири цифрові піни Arduino для керування: D10 (ENA) і D11 (ENB) для регулювання швидкості двигунів А і В через ШІМ (широтно-імпульсну модуляцію), а також D12 (IN1) і D13 (IN3) для керування напрямком обертання.

Шилд L298P має і певні обмеження. Його максимальний струм у 2 А на канал може бути недостатнім для потужних двигунів, що потребують більших струмів. Мікросхема L298P має відносно низьку енергоефективність через значні

теплові втрати, що може призводити до нагрівання при тривалій роботі на високих струмах.

У лабораторній роботі використовується формат MJPEG (Motion JPEG) для передачі відеопотоку з камери OV2640, вбудованої в модуль ESP32-CAM, до графічного інтерфейсу на комп'ютері через HTTP-сервер на порту 80. MJPEG дозволяє транслювати відео як послідовність окремих JPEG-зображень, які надсилаються в форматі multipart/x-mixed-replace. У програмі на Python, створеній за допомогою PyQt, ці зображення декодуються та відображаються в реальному часі, забезпечуючи користувачу можливість спостерігати за середовищем, у якому перебуває робот.

MJPEG є форматом потокового відео, у якому кожен кадр відео стискається окремо у форматі JPEG і передається як незалежне зображення. На відміну від інших форматів, таких як MPEG-4 або H.264, які використовують міжкадрове стиснення для зменшення розміру даних, MJPEG стискає кожен кадр незалежно, що спрощує обробку, але збільшує обсяг даних. У лабораторній роботі ESP32-CAM налаштовує камеру для захоплення кадрів у форматі JPEG із роздільною здатністю QVGA (320x240 пікселів) і якістю стиснення 12. Ці кадри передаються через HTTP-сервер, де кожен кадр відокремлюється спеціальним роздільником (boundary). Клієнт (GUI програма) отримує ці кадри, декодує їх за допомогою бібліотеки OpenCV і відображає як відеопотік із заданою частотою.

MJPEG стискає кожен кадр окремо, що робить його менш ефективним із точки зору використання пропускну здатності мережі. Однак, такий підхід забезпечує стабільну трансляцію зображення навіть на апаратному забезпеченні з обмеженими ресурсами. MJPEG дозволяє реалізувати відеотрансляцію в реальному часі без потреби в складних кодеках, що було б надто ресурсоємним для ESP32-CAM.

MJPEG також не підтримує аудіопотік, що обмежує його використання в задачах, де потрібна синхронізація звуку й відео. У лабораторній роботі це не є проблемою, оскільки аудіо не потрібне, але в більших системах це може бути обмеженням.

Хід виконання роботи

1. Написати код програми для ESP32-CAM для реалізації трансляції відео, зв'язку з Arduino Uno та передачі команд управління роботом по UART та з ПК по Wi-Fi через WebSocket-сервер.

- 1.1. Запустити середовище програмування Arduino IDE та створити новий проект для ESP32-CAM. За основу взяти проект з попередньої частини лабораторної роботи та у подальшому вносити у нього необхідні зміни.

- 1.2. Підключити додаткові бібліотеки в кодї:

```
#include <WebServer.h>
#include "esp_camera.h"
```

Бібліотека *WebServer.h* використовується для створення звичайного синхронного HTTP-сервера, який буде відповідати на запити MJPEG-клієнта. Бібліотека *esp_camera.h* забезпечує доступ до API камери ESP32-CAM, дозволяючи ініціалізувати сенсор і зчитувати кадри.

- 1.3. Створити веб-сервери:

```
WebServer mjpegServer(80);
AsyncWebServer wsServer(81);
```

Замість одного HTTP/WebSocket-сервера, тепер створено два незалежні сервери: *mjpegServer(80)* – стандартний HTTP-сервер на порту 80, який обробляє запити на відеотрансляцію (MJPEG); *wsServer(81)* – асинхронний WebSocket-сервер на порту 81, який використовується для керування та передачі команд.

- 1.4. Оголосити виводи камери:

```
// ===== Camera Config (AI Thinker) =====
#define PWDN_GPIO_NUM    32
#define RESET_GPIO_NUM   -1
#define XCLK_GPIO_NUM     0
#define SIOD_GPIO_NUM    26
#define SIOC_GPIO_NUM    27
#define Y9_GPIO_NUM       35
#define Y8_GPIO_NUM       34
#define Y7_GPIO_NUM       39
#define Y6_GPIO_NUM       36
#define Y5_GPIO_NUM       21
#define Y4_GPIO_NUM       19
#define Y3_GPIO_NUM       18
#define Y2_GPIO_NUM        5
```

```
#define VSYNC_GPIO_NUM    25
#define HREF_GPIO_NUM     23
#define PCLK_GPIO_NUM     22
```

1.5. Оголосити нові глобальні змінні:

```
volatile bool streamActive = false;
WiFiClient mjpegClient;
TaskHandle_t streamTaskHandle = NULL;
```

streamActive – глобальна змінна, що визначає, чи слід транслювати відео з камери. Змінюється через WebSocket-команди start/stop.

mjpegClient – об'єкт типу WiFiClient, який представляє HTTP-клієнта, підключеного до MJPEG-сервера. Через нього надсилаються кадри з камери.

streamTaskHandle – дескриптор задачі FreeRTOS, яка відповідає за трансляцію відеопотоку.

1.6. Оновити функцію onWebSocketMessage():

```
void onWebSocketMessage(AsyncWebSocket *server, AsyncWebSocketClient *client,
                        AwsFrameInfo *info, String data) {
    data.trim();
    if (data == "start") {
        streamActive = true;
        sendLog("Streaming enabled");
    } else if (data == "stop") {
        streamActive = false;
        sendLog("Streaming disabled");
    } else if (data == "w" || data == "a" || data == "s" || data == "d" ||
               data == "halt") {
        sendToArduino(data.c_str());
    }
}
```

Функція тепер обробляє нові команди:

"start" – вмикає трансляцію відео (встановлює *streamActive* = true).

"stop" – вимикає трансляцію відео.

"w", "a", "s", "d", "halt" – передаються через UART на Arduino за допомогою функції sendToArduino.

1.7. Створити функцію для обробки відеозапитів:

```
void handleVideoRequest() {
    mjpegClient = mjpegServer.client();

    mjpegClient.println("HTTP/1.1 200 OK");
    mjpegClient.println("Content-Type: multipart/x-mixed-replace;
boundary=frame");
    mjpegClient.println("Connection: close");
}
```

```

mjpegClient.println();

sendLog("MJPEG client connected");

streamActive = true;
}

```

Ця функція викликається HTTP-сервером при запиті */video*. Вона ініціалізує з'єднання з клієнтом і надсилає HTTP-заголовки, необхідні для формування MJPEG-потoku. Встановлюється MIME-тип *multipart/x-mixed-replace*, що дозволяє браузеру відображати серію JPEG-кадрів як відеопотік. Після цього автоматично активується трансляція через змінну *streamActive*.

1.8. Створити функцію-обробник задачі відеотрансляції:

```

void mjpegTask(void* pvParameters) {
    while (true) {
        if (streamActive && mjpegClient.connected()) {
            camera_fb_t *fb = esp_camera_fb_get();
            if (!fb) continue;

            mjpegClient.print("--frame\r\n");
            mjpegClient.print("Content-Type: image/jpeg\r\n");
            mjpegClient.printf("Content-Length: %u\r\n\r\n", fb->len);
            mjpegClient.write(fb->buf, fb->len);
            mjpegClient.print("\r\n");

            esp_camera_fb_return(fb);
            delay(50); // ~20 FPS
        } else {
            delay(100);
        }
    }
}

```

Це окрема задача FreeRTOS, яка виконується незалежно від основного циклу програми. Вона перевіряє стан *streamActive* і наявність підключеного клієнта. Якщо обидві умови виконуються:

- 1) Отримується кадр з камери (*esp_camera_fb_get()*).
- 2) Формується HTTP-відповідь із розділювачем *--frame*, заголовками *Content-Type* і *Content-Length*.
- 3) JPEG-дані кадру надсилаються клієнту.

Після надсилання кадр повертається назад у пул камерою (*esp_camera_fb_return()*).

Таким чином реалізується передача MJPEG-потoku. Частота оновлення приблизно 20 кадрів на секунду (затримка 50 мс).

1.9. Написати функцію для запуску камери:

```
// ===== Camera Init =====
bool startCamera() {
    camera_config_t config;
    config.ledc_channel = LEDC_CHANNEL_0;
    config.ledc_timer = LEDC_TIMER_0;
    config.pin_d0 = Y2_GPIO_NUM;
    config.pin_d1 = Y3_GPIO_NUM;
    config.pin_d2 = Y4_GPIO_NUM;
    config.pin_d3 = Y5_GPIO_NUM;
    config.pin_d4 = Y6_GPIO_NUM;
    config.pin_d5 = Y7_GPIO_NUM;
    config.pin_d6 = Y8_GPIO_NUM;
    config.pin_d7 = Y9_GPIO_NUM;
    config.pin_xclk = XCLK_GPIO_NUM;
    config.pin_pclk = PCLK_GPIO_NUM;
    config.pin_vsync = VSYNC_GPIO_NUM;
    config.pin_href = HREF_GPIO_NUM;
    config.pin_sscb_sda = SIOD_GPIO_NUM;
    config.pin_sscb_scl = SIOC_GPIO_NUM;
    config.pin_pwdn = PWDN_GPIO_NUM;
    config.pin_reset = RESET_GPIO_NUM;
    config.xclk_freq_hz = 24000000;
    config.pixel_format = PIXFORMAT_JPEG;
    config.frame_size = FRAMESIZE_QVGA;
    config.jpeg_quality = 12;
    config.fb_count = 2;
    config.grab_mode = CAMERA_GRAB_LATEST;
    config.fb_location = CAMERA_FB_IN_PSRAM;

    esp_err_t err = esp_camera_init(&config);
    if (err != ESP_OK) {
        return false;
    }

    // Additional camera settings
    sensor_t *s = esp_camera_sensor_get();
    if (s != nullptr) {
        s->set_vflip(s, 1); // Vertical flip
        s->set_hmirror(s, 1); // Horizontal flip

        // Turn on flashlight (LED on GPIO 4)
        pinMode(4, OUTPUT);
        digitalWrite(4, HIGH); // Always on
    }

    return true;
}
```


Ця функція ініціалізує камеру, налаштовуючи усі необхідні GPIO-виводи згідно зі специфікацією плати AI Thinker ESP32-CAM. Встановлюються такі параметри:

Частота XCLK (внутрішній тактовий генератор) – 24 МГц.

Формат пікселів – JPEG.

Розмір кадру – QVGA (320x240 пікселів).

Якість JPEG – 12 (задовільний компроміс між розміром і якістю).

Кількість буферів кадрів – 2 (для зменшення затримок).

Після ініціалізації також вмикається вбудований світлодіод (GPIO 4) як підсвітка. Якщо камера не ініціалізується, мікроконтролер зупиниться у нескінченному циклі з затримкою.

1.10. Оновити функцію *setup()*:

```
void setup() {
    Serial.begin(115200);
    Serial1.begin(9600, SERIAL_8N1, RXD1, TXD1);

    WiFi.begin(ssid, password);
    Serial.print("Connecting to WiFi");
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("\nWiFi connected: " + WiFi.localIP().toString());

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

    // Start MJPEG WebServer (sync)
    mjpegServer.on("/video", handleVideoRequest);
    mjpegServer.begin();
    Serial.println("MJPEG stream server started (port 80)");

    // Start WebSocket server (async)
    ws.onEvent(onWsEvent);
    wsServer.addHandler(&ws);
    wsServer.begin();
    Serial.println("WebSocket server started (port 81)");

    // Start MJPEG task
    xTaskCreatePinnedToCore(mjpegTask, "mjpeg", 8192, NULL, 1,
    &streamTaskHandle, 1);

    sendLog("ESP32 ready");
}
```


У функцію *setup()* додано такі нові частини:

- 1) Виклик *startCamera()* з перевіркою успішної ініціалізації.
- 2) Реєстрація обробника *handleVideoRequest* на шляху */video*.
- 3) Запуск двох серверів:
 - a. *mjpegServer.begin()* – запускає HTTP-сервер на порту 80.
 - b. *wsServer.begin()* – запускає WebSocket-сервер на порту 81.
- 4) Створення задачі *mjpegTask* за допомогою *xTaskCreatePinnedToCore()*, що забезпечує її фонове виконання на окремому ядрі ESP32.

1.11. Оновити функцію *loop()*:

```
void loop() {
    mjpegServer.handleClient();

    while (Serial1.available()) {
        char inChar = Serial1.read();
        if (inChar == '\n') {
            buffer[bufferIndex] = '\0';
            sendLog("Received from Arduino -> " + String(buffer));
            bufferIndex = 0;
        } else if (bufferIndex < sizeof(buffer) - 1) {
            buffer[bufferIndex++] = inChar;
        }
    }

    ws.cleanupClients();
    delay(1);
}
```

У *loop()* додано виклик *mjpegServer.handleClient()*, що обробляє запити від MJPEG-клієнтів. Також додано виклик *ws.cleanupClients()*, який звільняє ресурси неактивних WebSocket-з'єднань. Ці операції виконуються на кожній ітерації циклу для підтримки стабільного з'єднання.

1.12. Натиснути кнопку *Verify* (🔍) для попередньої компіляції та перевірки повного коду програми. Повний код виглядає так:

```
#include <WiFi.h>
#include <WebServer.h>
#include <ESPAsyncWebServer.h>
#include "esp_camera.h"

// ===== WiFi =====
const char* ssid = "Mystery";
const char* password = "BZ343MJIPH78T01SL";
```

```

// ===== MJPEG WebServer =====
WebServer mjpegServer(80);

// ===== WebSocket Async Server =====
AsyncWebServer wsServer(81);
AsyncWebSocket ws("/ws");

// ===== UART to Arduino =====
#define RXD1 15
#define TXD1 14
char buffer[64];
int bufferIndex = 0;

// ===== Camera Config (AI Thinker) =====
#define PWDN_GPIO_NUM    32
#define RESET_GPIO_NUM  -1
#define XCLK_GPIO_NUM    0
#define SIOD_GPIO_NUM    26
#define SIOC_GPIO_NUM    27
#define Y9_GPIO_NUM      35
#define Y8_GPIO_NUM      34
#define Y7_GPIO_NUM      39
#define Y6_GPIO_NUM      36
#define Y5_GPIO_NUM      21
#define Y4_GPIO_NUM      19
#define Y3_GPIO_NUM      18
#define Y2_GPIO_NUM       5
#define VSYNC_GPIO_NUM   25
#define HREF_GPIO_NUM    23
#define PCLK_GPIO_NUM    22

// ===== Streaming control =====
volatile bool streamActive = false;
WiFiClient mjpegClient;
TaskHandle_t streamTaskHandle = NULL;

// ===== Logging =====
void sendLog(const String& msg) {
    Serial.println(msg);
    ws.textAll(msg);
}

// ===== UART Forwarding =====
void sendToArduino(const char* msg) {
    Serial1.println(msg);
    Serial1.flush();
    sendLog("Sent to Arduino -> " + String(msg));
}

// ===== WebSocket Callback =====
void onWebSocketMessage(AsyncWebSocket *server, AsyncWebSocketClient *client,
                        AwsFrameInfo *info, String data) {
    data.trim();
    if (data == "start") {
        streamActive = true;
        sendLog("Streaming enabled");
    } else if (data == "stop") {
        streamActive = false;
        sendLog("Streaming disabled");
    }
}

```

```

    } else if (data == "w" || data == "a" || data == "s" || data == "d" || data
== "halt") {
        sendToArduino(data.c_str());
    }
}

void onWsEvent(AsyncWebSocket *server, AsyncWebSocketClient *client,
               AwsEventType type, void *arg, uint8_t *data, size_t len) {
    if (type == WS_EVT_DATA) {
        AwsFrameInfo *info = (AwsFrameInfo*)arg;
        if (info->final && info->index == 0 && info->len == len && info->opcode ==
WS_TEXT) {
            String msg = "";
            for (size_t i = 0; i < len; i++) msg += (char)data[i];
            onWebSocketMessage(server, client, info, msg);
        }
    }
}

// ===== MJPEG HTTP Handler (start only) =====
void handleVideoRequest() {
    mjpegClient = mjpegServer.client();

    mjpegClient.println("HTTP/1.1 200 OK");
    mjpegClient.println("Content-Type: multipart/x-mixed-replace;
boundary=frame");
    mjpegClient.println("Connection: close");
    mjpegClient.println();

    sendLog("MJPEG client connected");

    streamActive = true;

    // ===== MJPEG Streaming Task =====
    void mjpegTask(void* pvParameters) {
        while (true) {
            if (streamActive && mjpegClient.connected()) {
                camera_fb_t *fb = esp_camera_fb_get();
                if (!fb) continue;

                mjpegClient.print("--frame\r\n");
                mjpegClient.print("Content-Type: image/jpeg\r\n");
                mjpegClient.printf("Content-Length: %u\r\n\r\n", fb->len);
                mjpegClient.write(fb->buf, fb->len);
                mjpegClient.print("\r\n");

                esp_camera_fb_return(fb);
                delay(50); // ~20 FPS
            } else {
                delay(100);
            }
        }
    }

    // ===== Camera Init =====
    bool startCamera() {
        camera_config_t config;
        config.ledc_channel = LEDC_CHANNEL_0;
        config.ledc_timer = LEDC_TIMER_0;
    }
}

```

```

config.pin_d0      = Y2_GPIO_NUM;
config.pin_d1      = Y3_GPIO_NUM;
config.pin_d2      = Y4_GPIO_NUM;
config.pin_d3      = Y5_GPIO_NUM;
config.pin_d4      = Y6_GPIO_NUM;
config.pin_d5      = Y7_GPIO_NUM;
config.pin_d6      = Y8_GPIO_NUM;
config.pin_d7      = Y9_GPIO_NUM;
config.pin_xclk     = XCLK_GPIO_NUM;
config.pin_pclk     = PCLK_GPIO_NUM;
config.pin_vsync    = VSYNC_GPIO_NUM;
config.pin_href     = HREF_GPIO_NUM;
config.pin_sscb_sda = SIOD_GPIO_NUM;
config.pin_sscb_scl = SIOC_GPIO_NUM;
config.pin_pwdn     = PWDN_GPIO_NUM;
config.pin_reset    = RESET_GPIO_NUM;
config.xclk_freq_hz = 24000000;
config.pixel_format = PIXFORMAT_JPEG;
config.frame_size   = FRAMESIZE_QVGA;
config.jpeg_quality  = 12;
config.fb_count     = 2;
config.grab_mode     = CAMERA_GRAB_LATEST;
config.fb_location   = CAMERA_FB_IN_PSRAM;

esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) {
    return false;
}

// Additional camera settings
sensor_t *s = esp_camera_sensor_get();
if (s != nullptr) {
    s->set_vflip(s, 1); // Vertical flip
    s->set_hmirror(s, 1); // Horizontal flip

    // Turn on flashlight (LED on GPIO 4)
    pinMode(4, OUTPUT);
    digitalWrite(4, HIGH); // Always on
}

return true;
}

// ===== Setup =====
void setup() {
    Serial.begin(115200);
    Serial1.begin(9600, SERIAL_8N1, RXD1, TXD1);

    WiFi.begin(ssid, password);
    Serial.print("Connecting to WiFi");
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("\nWiFi connected: " + WiFi.localIP().toString());

    if (!startCamera()) {
        Serial.println("Camera init failed");
    }
}

```

```

    while (true) delay(1000);
}

// Start MJPEG WebServer (sync)
mjpegServer.on("/video", handleVideoRequest);
mjpegServer.begin();
Serial.println("MJPEG stream server started (port 80)");

// Start WebSocket server (async)
ws.onEvent(onWsEvent);
wsServer.addHandler(&ws);
wsServer.begin();
Serial.println("WebSocket server started (port 81)");

// Start MJPEG task
xTaskCreatePinnedToCore(mjpegTask, "mjpeg", 8192, NULL, 1,
&streamTaskHandle, 1);


    sendLog("ESP32 ready");
}

// ===== Main Loop =====
void loop() {
    mjpegServer.handleClient();

    while (Serial1.available()) {
        char inChar = Serial1.read();
        if (inChar == '\n') {
            buffer[bufferIndex] = '\0';
            sendLog("Received from Arduino -> " + String(buffer));
            bufferIndex = 0;
        } else if (bufferIndex < sizeof(buffer) - 1) {
            buffer[bufferIndex++] = inChar;
        }
    }

    ws.cleanupClients();
    delay(1);
}

```

- 1.13. Вимкнути живлення загальної схеми. Від'єднати лінію Tx ESP32-CAM від RX Arduino Uno. Підключити плату ESP32-CAM до ПК через USB за допомогою адаптера. Натиснути кнопку Upload () для завантаження програми до мікропроцесора. У випадку коректного завантаження, відключити ESP32-CAM від ПК.
2. Написати оновлений код для Arduino Uno.
 - 2.1. Створити новий проект для Arduino Uno. За основу взяти проект з попередньої частини лабораторної роботи та у подальшому вносити у нього необхідні зміни.
 - 2.2. Оголосити константи:

```
// Виводи керування для L298P Motor Shield (згідно з документацією)
const int ENA = 10; // PWM керування швидкістю двигуна А
const int ENB = 11; // PWM керування швидкістю двигуна В
const int IN1 = 12; // напрямок двигуна А
const int IN3 = 13; // напрямок двигуна В
const int SPEED = 255; // Motors speed
```

Ці змінні описують схему підключення двигунів до шилда L298P:

ENA — PWM-вивід для керування швидкістю двигуна А (лівого колеса).

ENB — PWM-вивід для керування швидкістю двигуна В (правого колеса).

IN1 — логічний вивід для задання напрямку обертання двигуна А.

IN3 — логічний вивід для задання напрямку обертання двигуна В.

SPEED — максимальна швидкість обох двигунів (255 — це максимум для PWM 8-біт).

2.3. Оновити функцію *setup()*:

```
void setup() {
    Serial.begin(9600);

    pinMode(ENA, OUTPUT);
    pinMode(ENB, OUTPUT);
    pinMode(IN1, OUTPUT);
    pinMode(IN3, OUTPUT);

    stopMotors();

    Serial.println("Ready to receive");
    Serial.flush();
}
```

pinMode() встановлює режим роботи кожного виводу як вихід.
stopMotors() зупиняє обидва двигуни одразу після запуску, щоб уникнути небажаного руху до отримання першої команди.

2.4. Оновити функцію *loop()*:

```
void loop() {
    if (Serial.available()) {
        char inChar = Serial.read();
        if (inChar == '\n') {
            buffer[bufferIndex] = '\0';

            // Remove trailing \r if present
            if (bufferIndex > 0 && buffer[bufferIndex - 1] == '\r') {
                buffer[--bufferIndex] = '\0';
            }
        }
    }
}
```

```

        handleCommand(buffer);
        Serial.print(buffer);
        Serial.println();
        Serial.flush();
        bufferIndex = 0;
    } else if (bufferIndex < sizeof(buffer) - 1) {
        buffer[bufferIndex++] = inChar;
    }
}
}
}

```

Після зчитування повної команди з UART-інтерфейсу (до символу `\n`) викликається нова функція `handleCommand()`, яка реалізує логіку руху.

2.5. Написати функцію опрацювання команд `handleCommand()`:

```

void handleCommand(const char* cmd) {
    if (strcmp(cmd, "w") == 0) {
        // Вперед (обидва двигуни вперед)
        digitalWrite(IN1, HIGH);
        digitalWrite(IN3, HIGH);
        analogWrite(ENA, SPEED);
        analogWrite(ENB, SPEED);
    } else if (strcmp(cmd, "s") == 0) {
        // Назад (обидва двигуни назад)
        digitalWrite(IN1, LOW);
        digitalWrite(IN3, LOW);
        analogWrite(ENA, SPEED);
        analogWrite(ENB, SPEED);
    } else if (strcmp(cmd, "a") == 0) {
        // Поворот вліво: праве колесо вперед
        digitalWrite(IN1, LOW); // ліве колесо зупинене (або назад)
        digitalWrite(IN3, HIGH); // праве вперед
        analogWrite(ENA, 0);
        analogWrite(ENB, SPEED);
    } else if (strcmp(cmd, "d") == 0) {
        // Поворот вправо: ліве колесо вперед
        digitalWrite(IN1, HIGH); // ліве вперед
        digitalWrite(IN3, LOW); // праве зупинене (або назад)
        analogWrite(ENA, SPEED);
        analogWrite(ENB, 0);
    } else if (strcmp(cmd, "halt") == 0) {
        stopMotors();
    }
}
}

```

Ця функція реалізує обробку текстових команд, які надходять через UART. Команди відповідають напрямкам руху:

- 1) "w" — рух вперед.
- 2) "s" — рух назад.
- 3) "a" — поворот вліво (тільки праве колесо обертається вперед).
- 4) "d" — поворот вправо (тільки ліве колесо обертається вперед).

5) "halt" — зупинка обох коліс.

2.6. Написати функцію зупинки руху *stopMotors()*:

```
void stopMotors() {  
    digitalWrite(IN1, LOW); // ліве колесо зупинене (або назад)  
    digitalWrite(IN3, LOW); // праве колесо зупинене (або назад)  
    analogWrite(ENA, 0);  
    analogWrite(ENB, 0);  
}
```

2.7. Натиснути кнопку *Verify* (🔍) для попередньої компіляції та перевірки повного коду програми. Повний код виглядає так:

```
const int ENA = 10; // PWM керування швидкістю двигуна A  
const int ENB = 11; // PWM керування швидкістю двигуна B  
const int IN1 = 12; // напрямок двигуна A  
const int IN3 = 13; // напрямок двигуна B  
const int SPEED = 255; // Motors speed  
  
char buffer[20];  
int bufferIndex = 0;  
  
void setup() {  
    Serial.begin(9600);  
  
    pinMode(ENA, OUTPUT);  
    pinMode(ENB, OUTPUT);  
    pinMode(IN1, OUTPUT);  
    pinMode(IN3, OUTPUT);  
  
    stopMotors();  
  
    Serial.println("Ready to receive");  
    Serial.flush();  
}  
  
void loop() {  
    if (Serial.available()) {  
        char inChar = Serial.read();  
        if (inChar == '\n') {  
            buffer[bufferIndex] = '\0';  
  
            // Remove trailing \r if present  
            if (bufferIndex > 0 && buffer[bufferIndex - 1] == '\r') {  
                buffer[--bufferIndex] = '\0';  
            }  
  
            handleCommand(buffer);  
            Serial.print(buffer);  
            Serial.println();  
            Serial.flush();  
            bufferIndex = 0;  
        } else if (bufferIndex < sizeof(buffer) - 1) {  
            buffer[bufferIndex++] = inChar;  
        }  
    }  
}
```



```

void handleCommand(const char* cmd) {
    if (strcmp(cmd, "w") == 0) {
        // Вперед (обидва двигуни вперед)
        digitalWrite(IN1, HIGH);
        digitalWrite(IN3, HIGH);
        analogWrite(ENA, SPEED);
        analogWrite(ENB, SPEED);
    } else if (strcmp(cmd, "s") == 0) {
        // Назад (обидва двигуни назад)
        digitalWrite(IN1, LOW);
        digitalWrite(IN3, LOW);
        analogWrite(ENA, SPEED);
        analogWrite(ENB, SPEED);
    } else if (strcmp(cmd, "a") == 0) {
        // Поворот вліво: праве колесо вперед
        digitalWrite(IN1, LOW); // ліве колесо зупинене (або назад)
        digitalWrite(IN3, HIGH); // праве вперед
        analogWrite(ENA, 0);
        analogWrite(ENB, SPEED);
    } else if (strcmp(cmd, "d") == 0) {
        // Поворот вправо: ліве колесо вперед
        digitalWrite(IN1, HIGH); // ліве вперед
        digitalWrite(IN3, LOW); // праве зупинене (або назад)
        analogWrite(ENA, SPEED);
        analogWrite(ENB, 0);
    } else if (strcmp(cmd, "halt") == 0) {
        stopMotors();
    }
}

void stopMotors() {
    digitalWrite(IN1, LOW); // ліве колесо зупинене (або назад)
    digitalWrite(IN3, LOW); // праве колесо зупинене (або назад)
    analogWrite(ENA, 0);
    analogWrite(ENB, 0);
}

```

2.8. Підключити плату Arduino Uno до ПК через USB. На час завантаження програми від'єднати лінію Tx ESP32-CAM від RX Arduino Uno. Натиснути кнопку *Upload* (📤) для завантаження програми до мікропроцесора. У випадку коректного завантаження відключити Arduino Uno від ПК.

3. Створити графічний інтерфейс для програми керування роботом.

3.1. Запустити Qt Designer.

3.2. Створити новий графічний інтерфейс (за аналогією до попередньої частини лабораторної роботи).

3.3. Розмістити у графічному інтерфейсі елементи згідно з таблицею 1.3. Орієнтовний вигляд інтерфейсу показано на рис. 1.17.

Таблиця 1.3. Опис віджетів графічного інтерфейсу керування роботом

Тип віджету	Назва (objectName)	Додаткові властивості
QMainWindow	Form	<i>windowTitle</i> : "Robot Video Control"
Label	video_label	<i>maximumSize</i> : width=320, height=240 <i>scaledContents</i> : true
PushButton	start_stream_button	<i>text</i> : "Старт відео"
PushButton	stop_stream_button	<i>text</i> : "Стоп відео"
PlainTextEdit	log_view	<i>wordWrap</i> : False <i>readOnly</i> : true <i>horizontalScrollBarPolicy</i> : AsNeeded <i>font</i> : Monospace <i>scrollBarsEnabled</i> : true

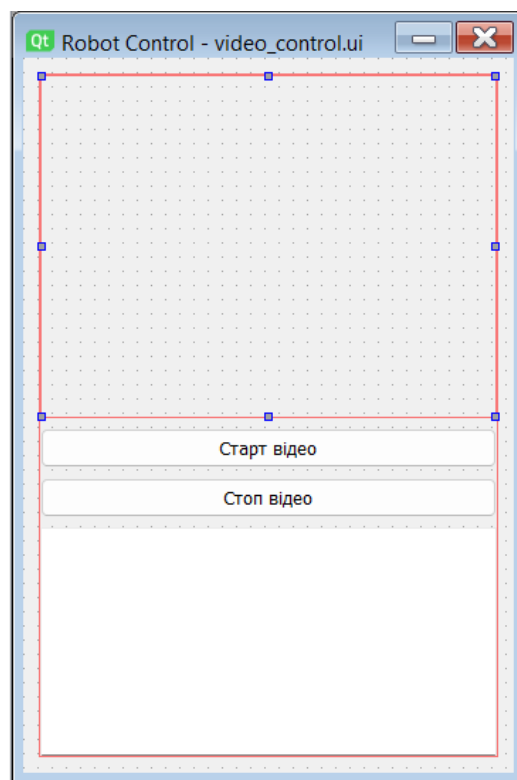


Рис. 1.17. Орієнтовний вигляд графічного інтерфейсу програми дистанційного керування роботом

3.4. Зберегти інтерфейс під назвою «*video_control.ui*».

4. Написати Python-програму для роботи зі створеним графічним інтерфейсом.

4.1. Відкрити будь-який редактор Python-коду або середовище розробки (наприклад, PyCharm) і створити новий *.py*-файл або проект. За основу

взяти файл/проект з попередньої частини лабораторної роботи та у подальшому вносити у нього необхідні зміни.

4.2. Імпортувати бібліотеки:

```
import sys
import asyncio
import aiohttp
import websockets
import cv2
import numpy as np
from PyQt5 import uic, QtGui, QtCore
from PyQt5.QtWidgets import QApplication, QWidget
from qasync import QEventLoop, asyncSlot
```

В цьому коді додано нові бібліотеки:

aiohttp – асинхронна бібліотека для HTTP-запитів, використовується для отримання MJPEG-відеопотоку.

cv2 (OpenCV) – використовується для декодування JPEG-кадрів із потоку.

numpy – необхідна для обробки байтового представлення зображення.

QtCore – модуль, який надає доступ до системного часу, обробки подій клавіатури та асинхронного планування.

4.3. Оголосити глобальні змінні:

```
ESP32_WS_URL = "ws://192.168.31.81:81/ws"
ESP32_VIDEO_URL = "http://192.168.31.81/video"

# Mapping key codes to movement commands
KEY_COMMANDS = {
    QtCore.Qt.Key_W: 'w',
    QtCore.Qt.Key_A: 'a',
    QtCore.Qt.Key_S: 's',
    QtCore.Qt.Key_D: 'd',
}
```

Змінні *ESP32_WS_URL* та *ESP32_VIDEO_URL* зберігають адреси, за якими відбувається з'єднання WebSocket (для керування ESP32) та отримання MJPEG-відеопотоку з камери ESP32 відповідно.

KEY_COMMANDS – це словник для переведення кодів клавіш W, A, S, D у текстові команди.

4.4. Оголосити клас *VideoControl* для обробки подій інтерфейсу та написати його конструктор:

```

class VideoControl(QWidget):
    def __init__(self):
        super().__init__()
        uic.loadUi("video_control.ui", self)

        self.start_stream_button = self.findChild(QWidget,
"start_stream_button")
        self.stop_stream_button = self.findChild(QWidget,
"stop_stream_button")
        self.video_label = self.findChild(QWidget, "video_label")
        self.log_view = self.findChild(QWidget, "log_view")

        self.start_stream_button.clicked.connect(lambda:
self.toggle_video(True))
        self.stop_stream_button.clicked.connect(lambda:
self.toggle_video(False))

        self.ws = None
        self.video_task = None
        self.stream_active = False
        self.latest_frame = None

        self.pressed_keys = set()
        asyncio.ensure_future(self.connect_ws_loop())

        self.setFocusPolicy(QtCore.Qt.StrongFocus)

```

Глобальні змінні класу наступні:

start_stream_button, *stop_stream_button* – кнопки запуску/зупинки відео.

video_label – графічне поле для відображення кадру.

log_view – текстове поле для виведення журналу подій.

ws – об'єкт WebSocket-з'єднання з ESP32.

video_task – задача трансляції відео, яка може бути зупинена.

stream_active – булева змінна, що показує, чи активна трансляція.

latest_frame – останній кадр, прийнятий з ESP32.

pressed_keys – множина натиснутих клавіш для контролю руху.

Рядок *self.setFocusPolicy(QtCore.Qt.StrongFocus)* дозволяє вікну додатку реагувати на натискання клавіш клавіатури.

4.5. Оновити метод *append_log()*:

```

def append_log(self, msg):

self.log_view.appendPlainText(f"[{QtCore.QTime.currentTime().toString()}]
{msg}")

```

Тепер цей метод додає повідомлення до лог-журналу з міткою часу.

4.6. Оновити метод *connect_ws_loop()*:

```

async def connect_ws_loop(self):
    while True:
        if self.ws is None or self.ws.closed:
            try:
                self.ws = await websockets.connect(ESP32_WS_URL)
                self.append_log("Connected to WebSocket")
                asyncio.ensure_future(self.receive_ws())
            except Exception as e:
                self.append_log(f"WebSocket error: {e}")
            await asyncio.sleep(0.5)

```

Метод асинхронно намагається встановити WebSocket-з'єднання у циклі з інтервалом 0.5 сек. Якщо з'єднання успішне — викликає *receive_ws()*.

4.7. Створити метод *receive_ws()*:

```

async def receive_ws(self):
    try:
        async for message in self.ws:
            self.append_log(f"ESP32: {message}")
    except Exception as e:
        self.append_log(f"WebSocket lost: {e}")
        self.ws = None

```

Цей метод асинхронно приймає повідомлення через WebSocket і виводить їх у лог. Якщо з'єднання обривається – записує повідомлення про втрату зв'язку і обнуляє *self.ws*.

4.8. Створити метод *toggle_video()*:

```

@asyncSlot()
async def toggle_video(self, enable):
    if not self.ws or self.ws.closed:
        self.append_log("WebSocket not connected")
        return
    try:
        await self.ws.send("start" if enable else "stop")
        self.stream_active = enable
        if enable:
            if self.video_task:
                self.video_task.cancel()
            self.video_task = asyncio.ensure_future(
                self.video_stream_task()
            )
        else:
            if self.video_task:
                self.video_task.cancel()
            self.video_label.clear()
            self.latest_frame = None
    except Exception as e:
        self.append_log(f"WebSocket error: {e}")

```

Метод керує початком або зупинкою відеопотоку з ESP32-CAM. Метод також надсилає відповідні команди через WebSocket (start або stop) і запускає/зупиняє задачу відображення відео. Загалом, метод виконує наступну послідовність дій:

- 1) Якщо WebSocket не підключено — вивести повідомлення про помилку.
- 2) Надіслати команду "start" або "stop" до ESP32.
- 3) Змінити стан логічної змінної *self.stream_active*.
- 4) Якщо увімкнено — створити задачу *video_stream_task()*.
- 5) Якщо вимкнено — зупинити задачу, очистити віджет зображення.

4.9. Створити метод *video_stream_task()*:

```
async def video_stream_task(self):
    try:
        async with aiohttp.ClientSession() as session:
            async with session.get(ESP32_VIDEO_URL) as resp:
                if resp.status != 200:
                    self.append_log(f"HTTP error: {resp.status}")
                    return

        buffer = b""
        while self.stream_active:
            chunk = await resp.content.read(4096)
            if not chunk:
                break
            buffer += chunk

            start = buffer.find(b'\xff\xd8')
            end = buffer.find(b'\xff\xd9', start)
            if start != -1 and end != -1:
                jpeg = buffer[start:end + 2]
                buffer = buffer[end + 2:]

            img_np = np.frombuffer(jpeg, np.uint8)
            frame = cv2.imdecode(img_np, cv2.IMREAD_COLOR)
            if frame is not None:
                frame = cv2.cvtColor(frame,
                                     cv2.COLOR_BGR2RGB)
                self.latest_frame = QtGui.QImage(
                    frame.data,
                    frame.shape[1],
                    frame.shape[0],
                    frame.strides[0],
                    QtGui.QImage.Format_RGB888
                )
                self.update_frame()
    except Exception as e:
        self.append_log(f"Video error: {e}")
    finally:
        self.stream_active = False
        self.video_label.setText("Stream stopped")
```

Потоково отримує MJPEG-відео з ESP32-CAM через HTTP і декодує кадри для відображення у графічному інтерфейсі. Повний алгоритм роботи методу наступний:

- 1) Відкрити HTTP-з'єднання до ESP32-CAM (*aiohttp.ClientSession()*).

- 2) Перевірити код відповіді. Якщо не 200, зупинити виконання.
- 3) Зібрати потік байтів (*buffer += chunk*).
- 4) Знайти у буфері межі JPEG-зображення:
 - a. Байти початку зображення: `\xff\xd8` (Start Of Image);
 - b. Байти кінця зображення: `\xff\xd9` (End Of Image).
- 5) Вирізати повне JPEG-зображення з буфера.
- 6) Декодувати байти у NumPy-масив, перетворити його в OpenCV-зображення.
- 7) Перетворити у формат *QImage* і зберегти в *self.latest_frame*.
- 8) Викликати метод *update_frame()* для оновлення віджета.
- 9) У разі помилки — вивести повідомлення і зупинити трансляцію.

4.10. Написати метод *update_frame()*:

```
def update_frame(self):
    if self.latest_frame and not self.latest_frame.isNull():
        pix = QtGui.QPixmap.fromImage(self.latest_frame).scaled(
            self.video_label.size(),
            QtCore.Qt.KeepAspectRatio,
            QtCore.Qt.SmoothTransformation
        )
        self.video_label.setPixmap(pix)
```

Метод оновлює віджет *video_label*, відображаючи у ньому останній збережений кадр з камери. Спочатку виконується перевірка, чи існує кадр (*self.latest_frame*) і він не є порожнім. Потім зображення масштабується до розмірів вікна. І після цього відображається в *video_label* за допомогою методу *setPixmap()*.

4.11. Написати метод *resizeEvent()* для правильного масштабування відео у разі зміни розмірів вікна:

```
def resizeEvent(self, event):
    super().resizeEvent(event)
    if self.latest_frame is not None:
        QtCore.QTimer.singleShot(30, self.update_frame)
```

Метод викликає *update_frame()* із затримкою 30 мс, щоб зображення підлаштувалося під новий розмір.

4.12. Написати метод для коректного закриття вікна та відеопотоку:

```
def closeEvent(self, event):
    self.stream_active = False
    if self.video_task:
        self.video_task.cancel()
```

```

if self.ws:
    asyncio.ensure_future(self.ws.close())
event.accept()

```

4.13. Написати метод-обробник натискань на кнопки клавіатури *keyPressEvent()*:

```

def keyPressEvent(self, event):
    if event.isAutoRepeat():
        return
    key = event.key()
    if key in KEY_COMMANDS and key not in self.pressed_keys:
        self.pressed_keys.add(key)
        asyncio.ensure_future(self.send_drive_command(KEY_COMMANDS[key]))

```

Метод реагує на натискання клавіш W, A, S, D та надсилає відповідну команду на ESP32-CAM через WebSocket. Також він ігнорує повторні натискання (*event.isAutoRepeat()*) – це дозволяє тримати клавішу руху затиснутою під час керування роботом. Метод визначає код клавіші, та якщо клавіша входить до *KEY_COMMANDS* і ще не оброблена додає її до множини *self.pressed_keys*. Потім до ESP32-CAM надсилається відповідна команда руху (w, a, s, d).

4.14. Написати метод-обробник відпускання кнопки клавіатури *keyReleaseEvent()*:

```

def keyReleaseEvent(self, event):
    if event.isAutoRepeat():
        return # Ignore auto-repeated key releases
    key = event.key()
    if key in self.pressed_keys:
        self.pressed_keys.remove(key)
        if not self.pressed_keys:
            asyncio.ensure_future(self.send_drive_command("halt"))

```

Метод є подібним до попереднього, але реагує на відпускання кнопки. Відпущена клавіша видаляється з *pressed_keys*, а до ESP32-CAM відправляється команда зупинку руху "halt".

4.15. Написати метод для відправки команд до ESP32-CAM:

```

async def send_drive_command(self, key):
    if not self.ws or self.ws.closed:
        return
    if key in {"w", "a", "s", "d", "halt"}:
        try:
            await self.ws.send(key)
            self.append_log(f"Command sent: {key}")
        except Exception as e:
            self.append_log(f"Send error: {e}")

```


Спочатку метод перевіряє, чи WebSocket активний. Якщо так і команда є припустимою, то він надсилає її через `self.ws.send(key)`. Також додається запис до логів через `append_log()`.

4.16. Провести тестовий запуск програми. Повний код програми:

```
import sys
import asyncio
import aiohttp
import websockets
import cv2
import numpy as np
from PyQt5 import uic, QtGui, QtCore
from PyQt5.QtWidgets import QApplication, QWidget
from qasync import QEventLoop, asyncSlot

ESP32_WS_URL = "ws://192.168.31.81:81/ws"
ESP32_VIDEO_URL = "http://192.168.31.81/video"

# Mapping key codes to movement commands
KEY_COMMANDS = {
    QtCore.Qt.Key_W: 'w',
    QtCore.Qt.Key_A: 'a',
    QtCore.Qt.Key_S: 's',
    QtCore.Qt.Key_D: 'd',
}

class VideoControl(QWidget):
    def __init__(self):
        super().__init__()
        uic.loadUi("video_control.ui", self)

        self.start_stream_button = self.findChild(QWidget,
"start_stream_button")
        self.stop_stream_button = self.findChild(QWidget,
"stop_stream_button")
        self.video_label = self.findChild(QWidget, "video_label")
        self.log_view = self.findChild(QWidget, "log_view")

        self.start_stream_button.clicked.connect(lambda:
self.toggle_video(True))
        self.stop_stream_button.clicked.connect(lambda:
self.toggle_video(False))

        self.ws = None
        self.video_task = None
        self.stream_active = False
        self.latest_frame = None

        self.pressed_keys = set()
        asyncio.ensure_future(self.connect_ws_loop())

        self.setFocusPolicy(QtCore.Qt.StrongFocus)

    def append_log(self, msg):
self.log_view.appendPlainText(f"[{QtCore.QTime.currentTime().toString()}]
{msg}")

    async def connect_ws_loop(self):
        while True:
```

```

        if self.ws is None or self.ws.closed:
            try:
                self.ws = await websockets.connect(ESP32_WS_URL)
                self.append_log("Connected to WebSocket")
                asyncio.ensure_future(self.receive_ws())
            except Exception as e:
                self.append_log(f"WebSocket error: {e}")
            await asyncio.sleep(0.5)

    async def receive_ws(self):
        try:
            async for message in self.ws:
                self.append_log(f"ESP32: {message}")
        except Exception as e:
            self.append_log(f"WebSocket lost: {e}")
            self.ws = None

    @asyncio.coroutine
    async def toggle_video(self, enable):
        if not self.ws or self.ws.closed:
            self.append_log("WebSocket not connected")
            return
        try:
            await self.ws.send("start" if enable else "stop")
            self.stream_active = enable
            if enable:
                if self.video_task:
                    self.video_task.cancel()
                self.video_task =
            asyncio.ensure_future(self.video_stream_task())
        except Exception as e:
            self.append_log(f"WebSocket error: {e}")

    async def video_stream_task(self):
        try:
            async with aiohttp.ClientSession() as session:
                async with session.get(ESP32_VIDEO_URL) as resp:
                    if resp.status != 200:
                        self.append_log(f"HTTP error: {resp.status}")
                        return

            buffer = b""
            while self.stream_active:
                chunk = await resp.content.read(4096)
                if not chunk:
                    break
                buffer += chunk

            start = buffer.find(b'\xff\xd8')
            end = buffer.find(b'\xff\xd9', start)
            if start != -1 and end != -1:
                jpeg = buffer[start:end + 2]
                buffer = buffer[end + 2:]

            img_np = np.frombuffer(jpeg, np.uint8)
            frame = cv2.imdecode(img_np, cv2.IMREAD_COLOR)
            if frame is not None:
                frame = cv2.cvtColor(frame,
cv2.COLOR_BGR2RGB)

```

```

        self.latest_frame = QtGui.QImage(
            frame.data,
            frame.shape[1],
            frame.shape[0],
            frame.strides[0],
            QtGui.QImage.Format_RGB888
        )
        self.update_frame()
    except Exception as e:
        self.append_log(f"Video error: {e}")
    finally:
        self.stream_active = False
        self.video_label.setText("Stream stopped")

def update_frame(self):
    if self.latest_frame and not self.latest_frame.isNull():
        pix = QtGui.QPixmap.fromImage(self.latest_frame).scaled(
            self.video_label.size(),
            QtCore.Qt.KeepAspectRatio,
            QtCore.Qt.SmoothTransformation
        )
        self.video_label.setPixmap(pix)

def resizeEvent(self, event):
    super().resizeEvent(event)
    if self.latest_frame is not None:
        QtCore.QTimer.singleShot(30, self.update_frame)

def closeEvent(self, event):
    self.stream_active = False
    if self.video_task:
        self.video_task.cancel()
    if self.ws:
        asyncio.ensure_future(self.ws.close())
    event.accept()

def keyPressEvent(self, event):
    if event.isAutoRepeat():
        return # Ignore auto-repeated key events
    key = event.key()
    if key in KEY_COMMANDS and key not in self.pressed_keys:
        self.pressed_keys.add(key)
    asyncio.ensure_future(self.send_drive_command(KEY_COMMANDS[key]))

def keyReleaseEvent(self, event):
    if event.isAutoRepeat():
        return
    key = event.key()
    if key in self.pressed_keys:
        self.pressed_keys.remove(key)
    if not self.pressed_keys:
        asyncio.ensure_future(self.send_drive_command("halt"))

async def send_drive_command(self, key):
    if not self.ws or self.ws.closed:
        return
    if key in {"w", "a", "s", "d", "halt"}:
        try:
            await self.ws.send(key)
            self.append_log(f"Command sent: {key}")
        except Exception as e:
            self.append_log(f"Send error: {e}")

```

```
def main():
    app = QApplication(sys.argv)
    loop = QEventLoop(app)
    asyncio.set_event_loop(loop)

    window = VideoControl()
    window.show()

    with loop:
        loop.run_forever()

if __name__ == "__main__":
    main()
```

- Зібрати схему, показану на рис. 1.18, та перевірити правильність роботи системи. Для цього підключити всі компоненти до живлення, запустити GUI та перевірити результат встановлення з'єднання з ESP32-CAM. Використовуючи створений графічний інтерфейс, увімкнути трансляцію відео з ESP32-CAM та протестувати керування рухом робота з клавіатури, натискаючи відповідні клавіші.

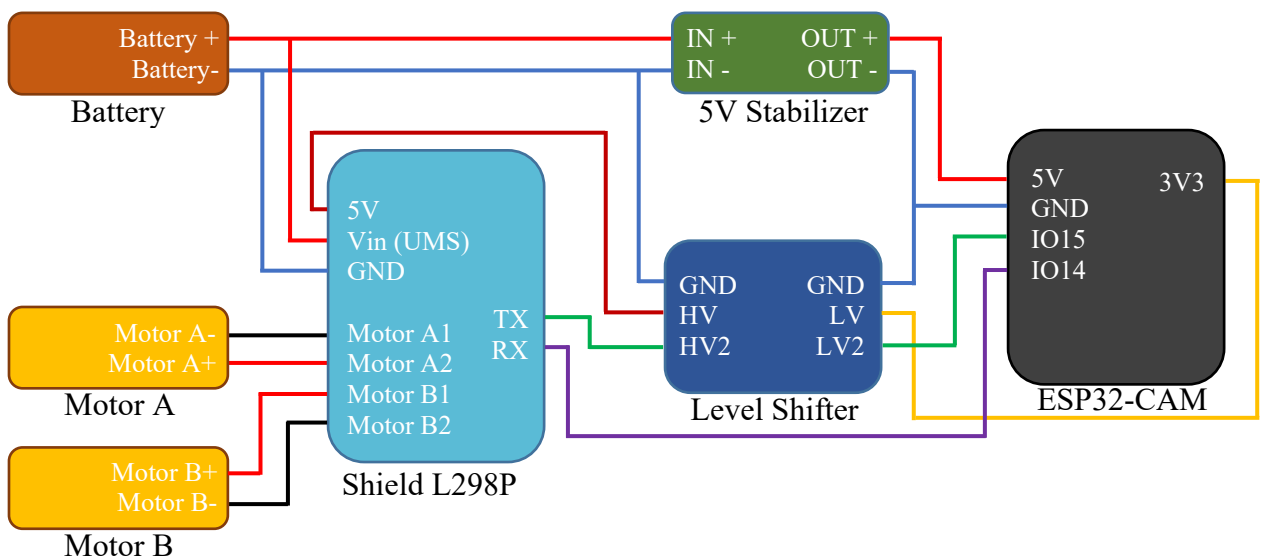


Рис. 1.18. Схема підключення компонентів робота

Завдання для самостійного виконання

Додати до GUI кнопку для увімкнення/вимкнення спалаху на ESP32-CAM (світлодіод спалаху підключений до GPIO 4).

Контрольні запитання

1. Опишіть основні функції та можливості моторного шилда L298P у системі дистанційного керування роботом. Які його переваги та обмеження для керування двигунами в робототехнічних проєктах?
2. Поясніть, чому формат MJPEG використовується для передачі відеопотоку з ESP32-CAM. У чому полягають його переваги та недоліки порівняно з іншими форматами H.264?
3. Яким чином у коді для ESP32-CAM реалізується одночасна робота WebSocket-сервера і HTTP-сервера? Як це впливає на обробку команд керування та відеотрансляцію?
4. Опишіть алгоритм роботи функції *mjpegTask()* у коді для ESP32-CAM. Як забезпечується стабільна передача відеопотоку з частотою ~20 кадрів на секунду?
5. Поясніть, як у Python-програмі обробляються події натискання та відпускання клавіш.
6. Опишіть процес декодування MJPEG-потoku в методі *video_stream_task()* Python-програми. Яким чином байтові дані перетворюються на зображення для відображення у віджеті *video_label*?
7. Поясніть, як у Python-програмі реалізується автоматична зупинка роботи при відпусканні всіх клавіш керування.
8. Як функція *toggle_video()* у Python-програмі взаємодіє з WebSocket-сервером для запуску та зупинки відеотрансляції? Які дії виконуються для коректного завершення відеопотоку?
9. Поясніть, як у Python-програмі реалізовано стійкість до втрати WebSocket-з'єднання. Які дії виконує метод *connect_ws_loop()* у разі помилки підключення, і як це відображається в інтерфейсі користувача?
10. Яким чином у коді для Arduino Uno реалізується керування рухом робота?

Лабораторна робота №2

АЛГОРИТМ АВТОНОМНОЇ НАВІГАЦІЇ РОБОТА

Мета: Ознайомлення з основами автономної навігації мобільних роботів. Навчитись реалізовувати алгоритм слідування мобільного робота в межах визначених ліній.

Теоретичні відомості

Система, описана в даній лабораторній роботі, призначена для автономного слідування робота за лінією з використанням комп'ютерного зору та відеотрансляції через WiFi. Вона демонструє принципи інтеграції апаратного забезпечення, обробки зображень і мережевих технологій для реалізації автономного керування в реальному часі. Система дозволяє роботу автоматично рухатися вздовж лінії, аналізуючи відеопотік із камери, та відображає процес на у графічному інтерфейсі на ПК.

Як і в попередній роботі, система складається з трьох основних компонентів: мікроконтролера ESP32-CAM, плати Arduino Uno з моторним щитом L298P і програмного забезпечення на базі Python із графічним інтерфейсом, створеним за допомогою PyQt. ESP32-CAM забезпечує WiFi-з'єднання, передачу відеопотоку у форматі MJPEG із вбудованої камери OV2640 та надсилання команд керування до Arduino Uno через послідовний інтерфейс UART. Плата Arduino Uno з моторним щитом L298P керує двома двигунами постійного струму, дозволяючи роботу виконувати рухи вперед, повороти ліворуч або праворуч і зупинятися. Програмне забезпечення на комп'ютері відображає відеопотік, аналізує його для виявлення ліній та автоматично формує команди керування, які надсилаються до ESP32-CAM. Структурна схема системи є повністю ідентичною тій, яка використовувалась у попередній роботі.

Алгоритм слідування за лінією, реалізований у цій лабораторній роботі, базується на обробці відеопотоку з камери ESP32-CAM за допомогою комп'ютерного зору для визначення траєкторії руху робота. Обробка зображень виконується на ПК за допомогою бібліотеки OpenCV, яка аналізує кадри для виявлення лінії та генерує команди керування ("w" для руху вперед, "a" для повороту ліворуч, "d" для повороту праворуч, "halt" для зупинки). Ці команди передаються через WebSocket до ESP32-CAM, а звідти через UART до плати Arduino Uno. Алгоритм включає кілька ключових кроків обробки зображення для виявлення лінії та визначення напрямку руху. Нижче наведено детальний опис кожного кроку алгоритму.

Крок 1: Конвертація зображення в градації сірого. Першим етапом обробки є перетворення кольорового кадру, отриманого з камери ESP32-CAM у форматі JPEG, у градації сірого. Конвертація зменшує обсяг даних, оскільки кольорове зображення з трьома каналами RGB (рис. 2.1а) перетворюється в одноканальне зображення (рис. 2.1б), де кожен піксель представлений інтенсивністю сірого кольору. Це спрощує подальшу обробку, оскільки лінія (зазвичай контрастна, наприклад, чорна на білому тлі) краще виділяється за інтенсивністю пікселів, а кольорова інформація є надлишковою. Сіре зображення зменшує обчислювальне навантаження та покращує ефективність наступних етапів.

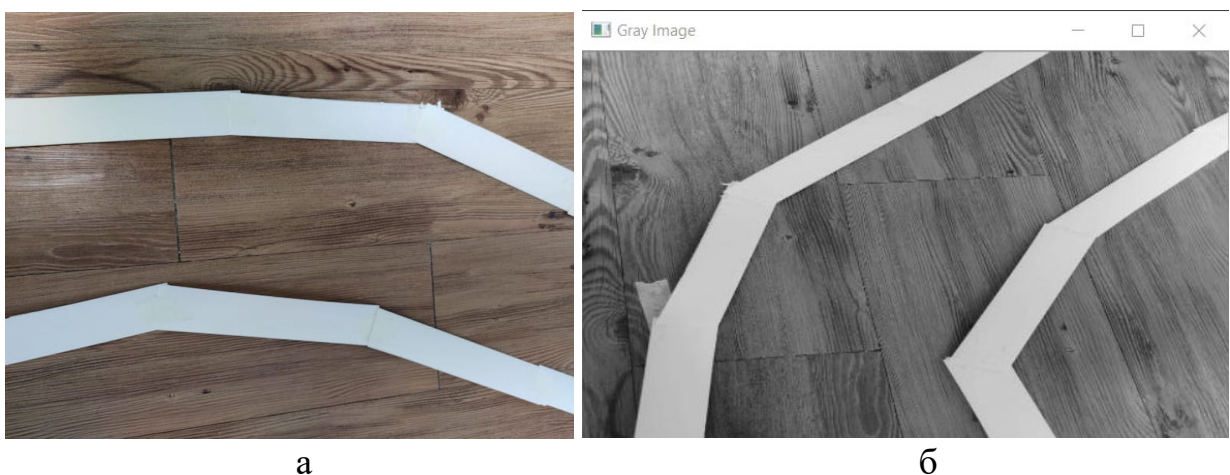


Рис. 2.1. Початкове зображення: а – кольорове; б – в градаціях сірого

Крок 2: Гаусівське розмиття. Наступним кроком є застосування гаусівського розмиття до сірого зображення (рис. 2.2). Цей метод використовує ядро розміром $N \times N$ пікселів із нульовим стандартним відхиленням для згладжування зображення. Метою розмиття є зменшення шумів, таких як дрібні деталі чи нерівності на поверхні, які можуть заважати виявленню лінії. Цей крок є важливим, оскільки відеопотік із камери OV2640 може містити шуми через низьку якість зображення або недостатнє освітлення, а гаусівське розмиття підвищує надійність подальшого виявлення контурів.

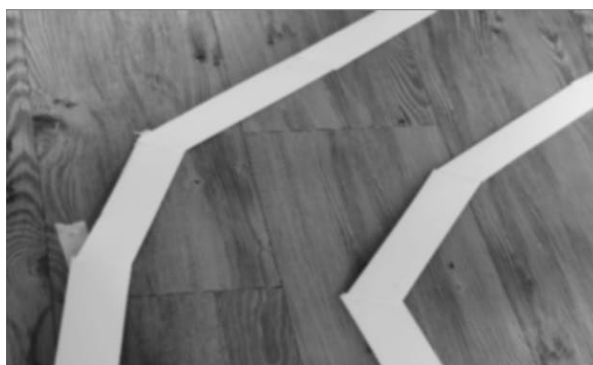


Рис. 2.2. Гаусівське розмиття зображення

Крок 3: Виявлення контурів за допомогою алгоритму Canny. Після розмиття зображення застосовується алгоритм Canny для виявлення контурів. Цей алгоритм виділяє краї об'єктів, таких як лінія, на основі градієнтів інтенсивності пікселів. У лабораторній роботі задаються порогові значення для нижнього та верхнього порогів градієнта, що дозволяє керувати виявленням лише значних країв, відфільтровуючи слабкі або шумові. Алгоритм Canny створює бінарне зображення, де краї лінії позначені білими пікселями (рис. 2.3), що є основою для подальшого виявлення ліній.

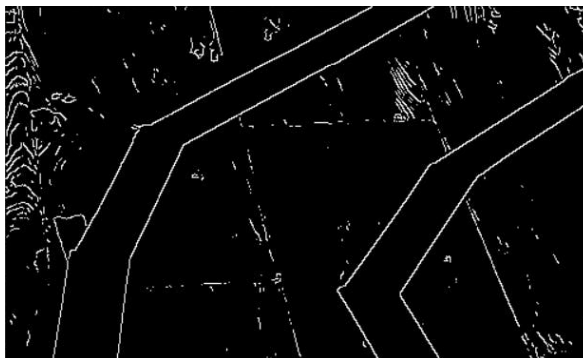


Рис. 2.3. Виявлені краї за алгоритмом Canny

Крок 4: Виявлення ліній за допомогою методу Хога. На основі отриманого бінарного зображення з контурами застосовується перетворення Хога для виявлення відрізків ліній (рис. 2.4). Ми будемо аналізувати лише нижню половину кадру (регіон інтересу, ROI), щоб зосередитися на лініях, які розташовані ближче до робота. Під час реалізації методу потрібно враховувати роздільну здатність (1 піксель і 1 градус або $\pi/180$ радіан), поріг для виявлення ліній, мінімальну довжину лінії і максимальний розрив між сегментами.

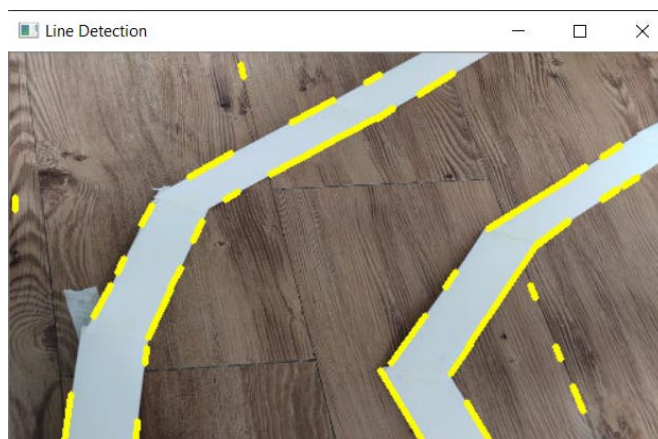


Рис. 2.4. Виявлені лінії за методом Хога

Крок 5: Визначення напрямку руху. Останній крок полягає у визначенні команди керування на основі середнього кута нахилу ліній. Для кожної виявленої лінії обчислюється кут нахилу θ за формулою $\text{atan}(y2 - y1, x2 - x1)$, що визначає її орієнтацію відносно горизонталі. Обчислений середній сумарний кут використовується для визначення напрямку руху (рис. 2.5).

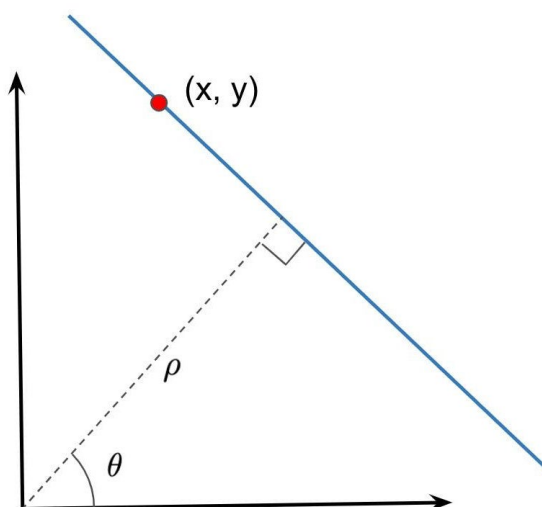


Рис. 2.5. Геометричне визначення кута нахилу лінії

Обчислений середній сумарний кут порівнюється з пороговим значенням. Якщо кут більший за поріг, надсилається команда "a" (поворот ліворуч), оскільки лінія повертає вліво; якщо менший – команда "d" (поворот праворуч); якщо кут знаходиться у заданих межах [-поріг, поріг] – команда "w" (рух вперед), оскільки лінія вважається прямою. Якщо ліній не виявлено, надсилається команда "halt" (зупинка).

Описаний алгоритм є ефективним для слідування за лінією завдяки послідовній обробці зображення, яка зменшує шум і зосереджується на ключових елементах. Використання ROI зменшує обчислювальне навантаження, оскільки аналізується лише нижня частина кадру, де лінія найімовірніше розташована. Робота з відеопотоком у реальному часі вимагає оптимізації параметрів на кожному кроці роботи алгоритму для стабільності в динамічних умовах. Обробка на ПК, а не на ESP32-CAM, дозволяє використовувати потужніші обчислювальні ресурси, що підвищує швидкість аналізу.

Хід виконання роботи

1. Коди програм для ESP32-CAM та Arduino Uno в даній лабораторній роботі нічим не будуть відрізнятися від останніх версій кодів з попередньої роботи. Тому змінювати їх не потрібно.
2. Створити графічний інтерфейс для програми автоматичної навігації робота за лінією.
 - 2.1. Запустити Qt Designer.
 - 2.2. Створити новий графічний інтерфейс (за аналогією до попередньої лабораторної роботи).
 - 2.3. Розмістити у графічному інтерфейсі елементи згідно з таблицею 2.1. Розміщення та візуальний стиль елементів обрати самостійно або орієнтуючись на аналогічний інтерфейс з попередньої лабораторної роботи.

Таблиця 2.1. Опис віджетів графічного інтерфейсу керування роботом

Тип віджету	Назва (objectName)	Додаткові властивості
QMainWindow	Form	<i>windowTitle</i> : "Lane Following Robot"
Label	video_label	<i>maximumSize</i> : width=320, height=240 <i>scaledContents</i> : true
PushButton	start_stream_button	<i>text</i> : "Старт відео"
PushButton	stop_stream_button	<i>text</i> : "Стоп відео"
PushButton	start_drive_button	<i>text</i> : "Почати рух"
PushButton	stop_drive_button	<i>text</i> : "Зупинити рух"
PlainTextEdit	log_view	<i>wordWrap</i> : False <i>readOnly</i> : true <i>horizontalScrollBarPolicy</i> : AsNeeded <i>font</i> : Monospace <i>scrollBarsEnabled</i> : true

2.4. Зберегти інтерфейс під назвою «*auto_control.ui*».

3. Написати Python-програму для роботи зі створеним графічним інтерфейсом.

3.1. Відкрити будь-який редактор Python-коду або середовище розробки (наприклад, PyCharm) і створити новий .py-файл або проект. За основу взяти файл/проект з попередньої лабораторної роботи та у подальшому вносити у нього необхідні зміни.

3.2. Створити нову глобальну змінну:

```
ANGLE_THRESHOLD = 0.15 # Radians ~ 8.5 degrees
```

Ця константа визначає допустимий діапазон середнього кута нахилу знайдених ліній на зображенні, у межах якого система вважає, що рухатися прямо (команда 'w') без необхідності корекції. Якщо середній кут перевищує цю межу, виконується поворот (ліворуч або праворуч).

3.3. До конструктора класу інтерфейсу додати нові елементи:

```
class VideoControl(QWidget):
    def __init__(self):
        super().__init__()
        uic.loadUi("auto_control.ui", self)

        self.start_stream_button = self.findChild(QWidget,
            "start_stream_button")
        self.stop_stream_button = self.findChild(QWidget,
```

```

"stop_stream_button")
self.video_label = self.findChild(QWidget, "video_label")
self.log_view = self.findChild(QWidget, "log_view")
self.start_drive_button = self.findChild(QWidget,
"start_drive_button")
self.stop_drive_button = self.findChild(QWidget,
"stop_drive_button")

self.start_stream_button.clicked.connect(lambda:
self.toggle_video(True))
self.stop_stream_button.clicked.connect(lambda:
self.toggle_video(False))
self.start_drive_button.clicked.connect(self.start_autonomous_drive)
self.stop_drive_button.clicked.connect(self.stop_autonomous_drive)

self.ws = None
self.video_task = None
self.stream_active = False
self.autonomous_drive = False
self.latest_frame = None
self.last_command = None
asyncio.ensure_future(self.connect_ws_loop())

```

У порівнянні з попереднім кодом, тут додаються нові кнопки до інтерфейсу, які дозволяють запускати або зупиняти автономний режим руху. Їхні сигнали підключаються до відповідних методів.

Також додано змінну стану *self.autonomous_drive*. Вона фіксує, чи знаходиться система в режимі автономного управління. Якщо так, кадри відео обробляються з метою визначення напрямку руху, а команда надсилається автоматично на ESP32-CAM через WebSocket.

Додано змінну стану *self.last_command*. Вона зберігає останню відправлену команду руху ("w", "a", "d" або "halt"). Це дозволяє уникнути надсилання повторюваних команд без потреби, що знижує навантаження на мережу та ESP32-CAM.

3.4. Оновити метод *video_stream_task()*:

```

async def video_stream_task(self):
    try:
        async with aiohttp.ClientSession() as session:
            async with session.get(ESP32_VIDEO_URL) as resp:
                if resp.status != 200:
                    self.append_log(f"HTTP error: {resp.status}")
                    return

        buffer = b""
        while self.stream_active:
            chunk = await resp.content.read(4096)
            if not chunk:
                break
            buffer += chunk

        start = buffer.find(b'\xff\xd8')

```

```

        end = buffer.find(b'\xff\xd9', start)
        if start != -1 and end != -1:
            jpeg = buffer[start:end + 2]
            buffer = buffer[end + 2:]

        img_np = np.frombuffer(jpeg, np.uint8)
        frame = cv2.imdecode(img_np, cv2.IMREAD_COLOR)
        if frame is not None:
            command, debug_frame =
                self.process_frame(frame)
            if self.autonomous_drive and command and
                command != self.last_command:
                await self.send_drive_command(command)
                self.last_command = command

            frame_rgb = cv2.cvtColor(debug_frame,
                                     cv2.COLOR_BGR2RGB)
            self.latest_frame = QtGui.QImage(
                frame_rgb.data,
                frame_rgb.shape[1],
                frame_rgb.shape[0],
                frame_rgb.strides[0],
                QtGui.QImage.Format_RGB888
            )
            self.update_frame()
    except Exception as e:
        self.append_log(f"Video error: {e}")
    finally:
        self.stream_active = False
        self.video_label.setText("Stream stopped")

```

Тут лише додається виклик методу *process_frame()* для аналізу кадру та автоматичного надсилення команди, якщо активовано автономний режим і команда відрізняється від попередньої.

3.5. Додати метод *process_frame()*:

```

def process_frame(self, frame):
    debug_frame = frame.copy()

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray, (5, 5), 0)
    edges = cv2.Canny(blur, 45, 80)

    height, width = edges.shape
    roi = edges[int(height/2):, :]

    lines = cv2.HoughLinesP(roi, 1, np.pi/180, 30,
                           minLineLength=20, maxLineGap=20)

    angle_sum = 0
    count = 0

    if lines is not None:
        offset_y = int(height / 2)
        for line in lines:
            x1, y1, x2, y2 = line[0]
            y1 += offset_y
            y2 += offset_y
            cv2.line(debug_frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
            angle = np.arctan2(y2 - y1, x2 - x1)
            angle_sum += angle

```

```

        count += 1

    if count == 0:
        return ("halt" if self.autonomous_drive and
                self.last_command != "halt" else None), debug_frame

    avg_angle = angle_sum / count

    if avg_angle > ANGLE_THRESHOLD:
        return "a", debug_frame
    elif avg_angle < -ANGLE_THRESHOLD:
        return "d", debug_frame
    else:
        return "w", debug_frame

```

Цей метод аналізує відеокадр із камери для визначення напрямку руху. Спочатку створюється копія кадру для виведення з накладеними лініями. Далі вхідний кадр переводиться в сірий (gray), що зменшує обсяг даних для обробки. Після цього Застосовується Gaussian Blur з ядром 5×5 і $\sigma=0$ для зменшення шумів. Потім використовується детектор Canny з порогоми 45 і 80. Визначається ROI, оскільки на наступному етапі аналізуються лише нижні 50% кадру. Потім застосовується метод Хога HoughLinesP для знаходження ліній у виділеній області.

На наступному етапі виконується обчислення середнього кута нахилу ліній та приймається рішення щодо команди руху. Метод повертає кортеж з команди управління та кадру з накладеними визначеними лініями.

3.6. Створити методи для увімкнення та вимкнення автоматичного режиму руху:

```

def start_autonomous_drive(self):
    self.autonomous_drive = True
    self.last_command = None
    self.append_log("Autonomous drive started")

def stop_autonomous_drive(self):
    self.autonomous_drive = False
    self.last_command = None
    asyncio.ensure_future(self.send_drive_command("halt"))
    self.append_log("Autonomous drive stopped")

```

Перший метод вмикає автономний режим. Змінна *self.autonomous_drive* стає True, а попередня команда скидається, щоб перша команда точно була відправлена. Другий метод вимикає автономний режим руху.

3.7. До методу `closeEvent()` додати рядок `self.autonomous_drive = False` для зупинки руху у разі закриття програми.

3.8. Провести тестовий запуск програми. Повний код програми:

```
import sys
import asyncio
import aiohttp
import websockets
import cv2
import numpy as np
from PyQt5 import uic, QtGui, QtCore
from PyQt5.QtWidgets import QApplication, QWidget
from qasync import QEventLoop, asyncSlot

ESP32_WS_URL = "ws://192.168.31.81:81/ws"
ESP32_VIDEO_URL = "http://192.168.31.81/video"

ANGLE_THRESHOLD = 0.15 # Radians ~ 8.5 degrees

class VideoControl(QWidget):
    def __init__(self):
        super().__init__()
        uic.loadUi("auto_control.ui", self)

        self.start_stream_button = self.findChild(QWidget,
"start_stream_button")
        self.stop_stream_button = self.findChild(QWidget,
"stop_stream_button")
        self.video_label = self.findChild(QWidget, "video_label")
        self.log_view = self.findChild(QWidget, "log_view")
        self.start_drive_button = self.findChild(QWidget,
"start_drive_button")
        self.stop_drive_button = self.findChild(QWidget,
"stop_drive_button")

        self.start_stream_button.clicked.connect(lambda:
self.toggle_video(True))
        self.stop_stream_button.clicked.connect(lambda:
self.toggle_video(False))
        self.start_drive_button.clicked.connect(self.start_autonomous_drive)
        self.stop_drive_button.clicked.connect(self.stop_autonomous_drive)

        self.ws = None
        self.video_task = None
        self.stream_active = False
        self.autonomous_drive = False
        self.latest_frame = None
        self.last_command = None
        asyncio.ensure_future(self.connect_ws_loop())

    def append_log(self, msg):
self.log_view.appendPlainText(f"[{QtCore.QTime.currentTime().toString()}]
{msg}")

    async def connect_ws_loop(self):
        while True:
            if self.ws is None or self.ws.closed:
                try:
                    self.ws = await websockets.connect(ESP32_WS_URL)
                    self.append_log("Connected to WebSocket")
                    asyncio.ensure_future(self.receive_ws())
```

```

        except Exception as e:
            self.append_log(f"WebSocket error: {e}")
            await asyncio.sleep(0.5)

    async def receive_ws(self):
        try:
            async for message in self.ws:
                self.append_log(f"ESP32: {message}")
        except Exception as e:
            self.append_log(f"WebSocket lost: {e}")
            self.ws = None

    @asyncio.coroutine
    async def toggle_video(self, enable):
        if not self.ws or self.ws.closed:
            self.append_log("WebSocket not connected")
            return
        try:
            await self.ws.send("start" if enable else "stop")
            self.stream_active = enable
            if enable:
                if self.video_task:
                    self.video_task.cancel()
                self.video_task =
            asyncio.ensure_future(self.video_stream_task())
        except Exception as e:
            self.append_log(f"WebSocket send error: {e}")

    async def video_stream_task(self):
        try:
            async with aiohttp.ClientSession() as session:
                async with session.get(ESP32_VIDEO_URL) as resp:
                    if resp.status != 200:
                        self.append_log(f"HTTP error: {resp.status}")
                        return

            buffer = b""
            while self.stream_active:
                chunk = await resp.content.read(4096)
                if not chunk:
                    break
                buffer += chunk

                start = buffer.find(b'\xff\xd8')
                end = buffer.find(b'\xff\xd9', start)
                if start != -1 and end != -1:
                    jpeg = buffer[start:end + 2]
                    buffer = buffer[end + 2:]

                img_np = np.frombuffer(jpeg, np.uint8)
                frame = cv2.imdecode(img_np, cv2.IMREAD_COLOR)
                if frame is not None:
                    command, debug_frame =
self.process_frame(frame)
                    if self.autonomous_drive and command and
command != self.last_command:
                        await self.send_drive_command(command)
                        self.last_command = command

```



```

        frame_rgb = cv2.cvtColor(debug_frame,
cv2.COLOR_BGR2RGB)

        self.latest_frame = QtGui.QImage(
            frame_rgb.data,
            frame_rgb.shape[1],
            frame_rgb.shape[0],
            frame_rgb.strides[0],
            QtGui.QImage.Format_RGB888
        )
        self.update_frame()

    except Exception as e:
        self.append_log(f"Video error: {e}")
    finally:
        self.stream_active = False
        self.video_label.setText("Stream stopped")

    def process_frame(self, frame):
        debug_frame = frame.copy()

        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        blur = cv2.GaussianBlur(gray, (5, 5), 0)
        edges = cv2.Canny(blur, 45, 80)

        height, width = edges.shape
        roi = edges[int(height/2):, :]

        lines = cv2.HoughLinesP(roi, 1, np.pi/180, 30, minLineLength=20,
maxLineGap=20)
        angle_sum = 0
        count = 0

        if lines is not None:
            offset_y = int(height / 2)
            for line in lines:
                x1, y1, x2, y2 = line[0]
                y1 += offset_y
                y2 += offset_y
                cv2.line(debug_frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
                angle = np.arctan2(y2 - y1, x2 - x1)
                angle_sum += angle
                count += 1

        if count == 0:
            return ("halt" if self.autonomous_drive and self.last_command !=
"halt" else None), debug_frame

        avg_angle = angle_sum / count

        if avg_angle > ANGLE_THRESHOLD:
            return "a", debug_frame
        elif avg_angle < -ANGLE_THRESHOLD:
            return "d", debug_frame
        else:
            return "w", debug_frame

    def update_frame(self):
        if self.latest_frame and not self.latest_frame.isNull():
            pix = QtGui.QPixmap.fromImage(self.latest_frame).scaled(
                self.video_label.size(),
                QtCore.Qt.KeepAspectRatio,
                QtCore.Qt.SmoothTransformation
            )
            self.video_label.setPixmap(pix)

```

```

def resizeEvent(self, event):
    super().resizeEvent(event)
    if self.latest_frame is not None:
        QtCore.QTimer.singleShot(30, self.update_frame)

def closeEvent(self, event):
    self.stream_active = False
    self.autonomous_drive = False
    if self.video_task:
        self.video_task.cancel()
    if self.ws:
        asyncio.ensure_future(self.ws.close())
    event.accept()

def start_autonomous_drive(self):
    self.autonomous_drive = True
    self.last_command = None
    self.append_log("Autonomous drive started")

def stop_autonomous_drive(self):
    self.autonomous_drive = False
    self.last_command = None
    asyncio.ensure_future(self.send_drive_command("halt"))
    self.append_log("Autonomous drive stopped")

async def send_drive_command(self, key):
    if not self.ws or self.ws.closed:
        return
    if key in {"w", "a", "s", "d", "halt"}:
        try:
            await self.ws.send(key)
            self.append_log(f"Sent command: {key}")
        except Exception as e:
            self.append_log(f"Send error: {e}")

def main():
    app = QApplication(sys.argv)
    loop = QEventLoop(app)
    asyncio.set_event_loop(loop)

    window = VideoControl()
    window.show()

    with loop:
        loop.run_forever()

if __name__ == "__main__":
    main()

```

4. Перевірити правильність роботи системи. Для цього підключити всі компоненти до живлення, запустити GUI та перевірити результат встановлення з'єднання з ESP32-CAM. Використовуючи створений графічний інтерфейс, увімкнути трансляцію відео з ESP32-CAM та протестувати режим автономної навігації робота, увімкнувши його за допомогою відповідних кнопок.

Завдання для самостійного виконання

Знайти спосіб забезпечити більш плавну реакцію роботи на різкі зміни траєкторії лінії.

Контрольні запитання

1. Опишіть основні етапи алгоритму слідування за лінією, реалізованого в лабораторній роботі. Яку роль відіграє кожен етап у процесі обробки зображення для визначення траєкторії руху робота?
2. Поясніть, чому для обробки відеопотоку використовується конвертація зображення в градації сірого. Як це впливає на обчислювальне навантаження та точність виявлення лінії?
3. Яка мета застосування гаусівського розмиття в алгоритмі обробки зображення? Як вибір розміру ядра впливає на результат виявлення контурів?
4. Опишіть принцип роботи алгоритму Canny для виявлення контурів. Чому в лабораторній роботі використовуються два порогових значення для методу Canny, і як їх зміна може вплинути на результат?
5. Як у методі *process_frame()* обчислюється середній кут нахилу ліній, і як він використовується для визначення команди керування ("w", "a", "d" або "halt")?
6. Яким чином у Python-програмі забезпечується уникнення надсилання повторюваних команд керування?
7. Опишіть, як параметри методу Хога (роздільна здатність, поріг, мінімальна довжина лінії, максимальний розрив) впливають на точність і надійність виявлення ліній. Як їх можна оптимізувати для різних умов освітлення?
8. Яким чином у системі забезпечується стабільність роботи при низькій якості зображення або недостатньому освітленні? Які кроки алгоритму обробки зображень сприяють цьому?
9. Як можна модифікувати алгоритм слідування за лінією для врахування криволінійних траєкторій із різним радіусом повороту? Які зміни потрібно внести в метод *process_frame()* для реалізації такої функціональності?

КЕРУВАННЯ РОБОТИЗОВАНИМ МАНІПУЛЯТОРОМ НА ПРИКЛАДІ МЕХАНІЗМУ PAN-TILT

Мета: Ознайомлення з принципами роботи маніпулятора типу Pan-Tilt для керування положенням камери. Вивчення особливостей віддаленого керування серводвигунами через мережу WiFi.

Теоретичні відомості

Система, описана в даній лабораторній роботі, призначена для дистанційного керування рухом робота та позиціонуванням камери за допомогою механізму Pan-Tilt, що забезпечує поворот і нахил камери. Вона дозволяє користувачу не лише спрямовувати рух робота, але й змінювати кут огляду камери для спостереження за середовищем через відеотрансляцію.

Основними компонентами системи є мікроконтролер ESP32-CAM, плата Arduino з моторним щитом L298P, два серводвигуни для механізму Pan-Tilt і програмне забезпечення на базі Python із графічним інтерфейсом, створеним за допомогою PyQt. Плата Arduino Uno з моторним шилдом L298P керує двома двигунами постійного струму, які забезпечують рух робота вперед, назад, повороти ліворуч або праворуч, а також зупинку. Два серводвигуни реалізують механізм Pan-Tilt, дозволяючи повертати камеру по горизонталі (pan) і вертикалі (tilt). Програмне забезпечення на комп'ютері відображає відеопотік і надає інтерфейс із кнопками для запуску та зупинки трансляції, клавішами W, A, S, D для керування рухом і числовими полями для встановлення кутів повороту серводвигунів. Команди позиціонування камери (наприклад, "pan:90" або "tilt:90") надсилаються через WebSocket до ESP32-CAM, який передає їх через UART до Arduino Uno. Arduino обробляє ці команди, активуючи двигуни через

шилд L298P для руху робота або змінюючи положення серводвигунів для повороту камери.

Механізм Pan-Tilt складається з двох серводвигунів, механічної платформи та кріплення для камери (рис. 3.1). Два серводвигуни підключені до моторного шилда через піни даних: один серводвигун (tilt) відповідає за вертикальний нахил камери, другий (pan) — за горизонтальний поворот. У цій лабораторній роботі серводвигуни будуть підключені до пінів 9 (pan) і 6 (tilt) на Arduino Uno. Механічна платформа забезпечує кріплення камери та її рух у двох площинах. Серводвигуни живляться від зовнішнього джерела 5 В, щоб уникнути перевантаження Arduino, а їхні сигнальні піни отримують команди від мікроконтролера.

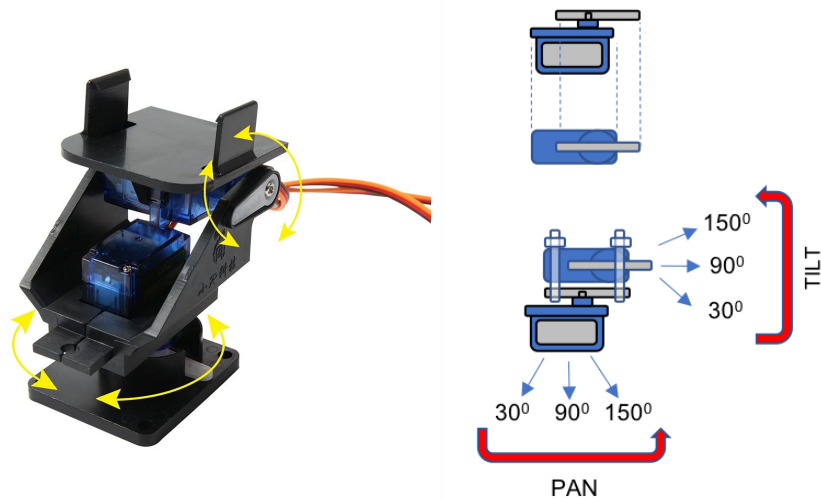


Рис. 3.1. Механізм Pan-Tilt

Серводвигуни в механізмі Pan-Tilt працюють на основі широтно-імпульсної модуляції (ШІМ), де кут повороту визначається тривалістю імпульсу сигналу. Бібліотека Servo.h на Arduino забезпечує зручне керування серводвигунами, дозволяючи встановлювати кути від 0 до 180 градусів. Наприклад, команда "pan:90" або "tilt:90" надсилається з графічного інтерфейсу через WebSocket до ESP32-CAM, а потім через UART до Arduino Uno, яка викликає відповідну функцію для встановлення заданого кута. Кути обмежуються діапазоном 0–180 градусів, щоб запобігти надсиланню некоректних значень.

Хід виконання роботи

1. Написати код програми для ESP32-CAM для реалізації можливості керування механізмом Pan-Tilt.

1.1. Запустити середовище програмування Arduino IDE та створити новий проект для ESP32-CAM. За основу взяти проект з лабораторної роботи №1 (частина 2) та у подальшому вносити у нього необхідні зміни.

1.2. Оновити тип аргументу функції *sendToArduino()*:

```
void sendToArduino(const String& msg) {  
    Serial1.println(msg);  
    Serial1.flush();  
    sendLog("Sent to Arduino -> " + msg);  
}
```

Заміна типу з *const char** на *const String&* спрощує виклик функції з об'єктами типу *String*, особливо під час обробки команд «pan:» та «tilt:».

1.3. Оновити функцію *onWebSocketMessage()*:

```
void onWebSocketMessage(AsyncWebSocket *server, AsyncWebSocketClient *client,  
                        AwsFrameInfo *info, String data) {  
    data.trim();  
  
    if (data == "start") {  
        streamActive = true;  
        sendLog("Streaming enabled");  
    } else if (data == "stop") {  
        streamActive = false;  
        sendLog("Streaming disabled");  
    } else if (  
        data == "w" || data == "a" || data == "s" || data == "d" || data == "halt"  
    ) {  
        sendToArduino(data);  
    } else if (  
        data.startsWith("pan:") || data.startsWith("tilt:")  
    ) {  
        // Validate and forward absolute servo commands  
        int colonPos = data.indexOf(':');  
        if (colonPos != -1) {  
            String valuePart = data.substring(colonPos + 1);  
            int angle = valuePart.toInt();  
            if (angle >= 0 && angle <= 180) {  
                sendToArduino(data);  
            } else {  
                sendLog("Invalid servo angle: " + valuePart);  
            }  
        }  
    } else {  
        sendLog("Unknown command: " + data);  
    }  
}
```

Тепер функція дозволяє приймати абсолютні команди керування кутами повороту (pan) та нахилу (tilt) камери. Спочатку відбувається перевірка формату команди. Команда керування механізмом Pan-Tilt має починатись з «pan:» або «tilt:». Далі перевіряється наявність символу «:» (двокрапка) як роздільника. Після цього відбувається перевірка коректності значення кута, яке має бути в межах від 0 до 180. Якщо значення допустиме, викликається *sendToArduino(data)*, тобто передається вся команда без змін. Якщо значення некоректне або команда невідома, функція логування *sendLog()* надсилає повідомлення про помилку назад клієнту.

- 1.4. Натиснути кнопку *Verify* (🔍) для попередньої компіляції та перевірки повного коду програми. Повний код виглядає так:

```
#include <WiFi.h>
#include <WebServer.h>
#include <ESPAsyncWebServer.h>
#include "esp_camera.h"

// ===== WiFi =====
const char* ssid = "Mystery";
const char* password = "BZ343MJIPH78T01SL";

// ===== MJPEG WebServer =====
WebServer mjpegServer(80);

// ===== WebSocket Async Server =====
AsyncWebServer wsServer(81);
AsyncWebSocket ws("/ws");

// ===== UART to Arduino =====
#define RXD1 15
#define TXD1 14
char buffer[64];
int bufferIndex = 0;

// ===== Camera Config (AI Thinker) =====
#define PWDN_GPIO_NUM    32
#define RESET_GPIO_NUM  -1
#define XCLK_GPIO_NUM    0
#define SIOD_GPIO_NUM    26
#define SIOC_GPIO_NUM    27
#define Y9_GPIO_NUM      35
#define Y8_GPIO_NUM      34
#define Y7_GPIO_NUM      39
#define Y6_GPIO_NUM      36
#define Y5_GPIO_NUM      21
#define Y4_GPIO_NUM      19
#define Y3_GPIO_NUM      18
```

```

#define Y2_GPIO_NUM      5
#define VSYNC_GPIO_NUM  25
#define HREF_GPIO_NUM    23
#define PCLK_GPIO_NUM    22

// ===== Streaming control =====
volatile bool streamActive = false;
WiFiClient mjpegClient;
TaskHandle_t streamTaskHandle = NULL;

// ===== Logging =====
void sendLog(const String& msg) {
    Serial.println(msg);
    ws.textAll(msg);
}

// ===== UART Forwarding =====
void sendToArduino(const String& msg) {
    Serial1.println(msg);
    Serial1.flush();
    sendLog("Sent to Arduino -> " + msg);
}

// ===== WebSocket Callback =====
void onWebSocketMessage(AsyncWebSocket *server, AsyncWebSocketClient *client,
    AwsFrameInfo *info, String data) {
    data.trim();

    if (data == "start") {
        streamActive = true;
        sendLog("Streaming enabled");
    } else if (data == "stop") {
        streamActive = false;
        sendLog("Streaming disabled");
    } else if (
        data == "w" || data == "a" || data == "s" || data == "d" || data == "halt"
    ) {
        sendToArduino(data);
    } else if (
        data.startsWith("pan:") || data.startsWith("tilt:")
    ) {
        // Validate and forward absolute servo commands
        int colonPos = data.indexOf(':');
        if (colonPos != -1) {
            String valuePart = data.substring(colonPos + 1);
            int angle = valuePart.toInt();
            if (angle >= 0 && angle <= 180) {
                sendToArduino(data);
            } else {
                sendLog("Invalid servo angle: " + valuePart);
            }
        }
    } else {
        sendLog("Unknown command: " + data);
    }
}

void onWsEvent(AsyncWebSocket *server, AsyncWebSocketClient *client,
    AwsEventType type, void *arg, uint8_t *data, size_t len) {

```



```

    if (type == WS_EVT_DATA) {
        AwsFrameInfo *info = (AwsFrameInfo*)arg;
        if (info->final && info->index == 0 && info->len == len && info->opcode ==
WS_TEXT) {
            String msg;
            for (size_t i = 0; i < len; i++) msg += (char)data[i];
            onWebSocketMessage(server, client, info, msg);
        }
    }
}

// ===== MJPEG HTTP Handler =====
void handleVideoRequest() {
    mjpegClient = mjpegServer.client();

    mjpegClient.println("HTTP/1.1 200 OK");
    mjpegClient.println("Content-Type: multipart/x-mixed-replace;
boundary=frame");
    mjpegClient.println("Connection: close");
    mjpegClient.println();

    sendLog("MJPEG client connected");

    streamActive = true;
}

// ===== MJPEG Streaming Task =====
void mjpegTask(void* pvParameters) {
    while (true) {
        if (streamActive && mjpegClient.connected()) {
            camera_fb_t *fb = esp_camera_fb_get();
            if (!fb) continue;

            mjpegClient.print("--frame\r\n");
            mjpegClient.print("Content-Type: image/jpeg\r\n");
            mjpegClient.printf("Content-Length: %u\r\n\r\n", fb->len);
            mjpegClient.write(fb->buf, fb->len);
            mjpegClient.print("\r\n");

            esp_camera_fb_return(fb);
            delay(50); // ~20 FPS
        } else {
            delay(100); // Idle
        }
    }
}

// ===== Camera Init =====
bool startCamera() {
    camera_config_t config;
    config.ledc_channel = LEDC_CHANNEL_0;
    config.ledc_timer = LEDC_TIMER_0;
    config.pin_d0 = Y2_GPIO_NUM;
    config.pin_d1 = Y3_GPIO_NUM;
    config.pin_d2 = Y4_GPIO_NUM;
    config.pin_d3 = Y5_GPIO_NUM;
    config.pin_d4 = Y6_GPIO_NUM;
    config.pin_d5 = Y7_GPIO_NUM;
    config.pin_d6 = Y8_GPIO_NUM;

```

```

config.pin_d7      = Y9_GPIO_NUM;
config.pin_xclk    = XCLK_GPIO_NUM;
config.pin_pclk    = PCLK_GPIO_NUM;
config.pin_vsync   = VSYNC_GPIO_NUM;
config.pin_href    = HREF_GPIO_NUM;
config.pin_sscb_sda = SIOD_GPIO_NUM;
config.pin_sscb_scl = SIOC_GPIO_NUM;
config.pin_pwdn    = PWDN_GPIO_NUM;
config.pin_reset   = RESET_GPIO_NUM;
config.xclk_freq_hz = 24000000;
config.pixel_format = PIXFORMAT_JPEG;
config.frame_size   = FRAMESIZE_QVGA;
config.jpeg_quality  = 12;
config.fb_count     = 2;
config.grab_mode     = CAMERA_GRAB_LATEST;
config.fb_location   = CAMERA_FB_IN_PSRAM;

esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) {
    return false;
}

sensor_t *s = esp_camera_sensor_get();
if (s != nullptr) {
    s->set_vflip(s, 1);
    s->set_hmirror(s, 1);
    pinMode(4, OUTPUT);
    digitalWrite(4, HIGH);
}

return true;
}

// ===== Setup =====
void setup() {
    Serial.begin(115200);
    Serial1.begin(9600, SERIAL_8N1, RXD1, TXD1);

    WiFi.begin(ssid, password);
    Serial.print("Connecting to WiFi");
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("\nWiFi connected: " + WiFi.localIP().toString());

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

    mjpegServer.on("/video", handleVideoRequest);
    mjpegServer.begin();
    Serial.println("MJPEG stream server started (port 80)");

    ws.onEvent(onWsEvent);
    wsServer.addHandler(&ws);
    wsServer.begin();
    Serial.println("WebSocket server started (port 81)");
}

```

```

    xTaskCreatePinnedToCore(mjpegTask, "mjpeg", 8192, NULL, 1,
&streamTaskHandle, 1);

    sendLog("ESP32 ready");
}

// ===== Main Loop =====
void loop() {
    mjpegServer.handleClient();

    while (Serial1.available()) {
        char inChar = Serial1.read();
        if (inChar == '\n') {
            buffer[bufferIndex] = '\0';
            sendLog("Received from Arduino -> " + String(buffer));
            bufferIndex = 0;
        } else if (bufferIndex < sizeof(buffer) - 1) {
            buffer[bufferIndex++] = inChar;
        }
    }

    ws.cleanupClients();
    delay(1);
}

```

1.5. Вимкнути живлення загальної схеми. Від'єднати лінію Tx ESP32-CAM від RX Arduino Uno. Підключити плату ESP32-CAM до ПК через USB за допомогою адаптера. Натиснути кнопку Upload (📤) для завантаження програми до мікропроцесора. У випадку коректного завантаження, відключити ESP32-CAM від ПК.

2. Написати оновлений код для Arduino Uno.

2.1. Створити новий проект для Arduino Uno. За основу взяти проект з попередньої частини лабораторної роботи та у подальшому вносити у нього необхідні зміни.

2.2. Підключити бібліотеку *Servo.h*:

```
#include <Servo.h>
```

Ця бібліотека надає високорівневі засоби для керування сервоприводами через ШІМ-сигнали, не використовуючи прямого керування таймерами.

2.3. Оголосити піни, до яких підключені сервоприводи:

```
const int PAN_PIN = 9;
const int TILT_PIN = 6;
```

PAN_PIN – горизонтальне обертання, TILT_PIN – вертикальний нахил.

2.4. Створити змінні для зберігання поточних кутів сервоприводів:

```
int panAngle = 90;
int tiltAngle = 90;
```

2.5. Створити об'єкти сервоприводів:

```
Servo panServo;
Servo tiltServo;
```

2.6. Ініціалізувати сервоприводи у функції *setup()*:

```
void setup() {
    Serial.begin(9600);

    // Motor pins
    pinMode(ENA, OUTPUT);
    pinMode(ENB, OUTPUT);
    pinMode(IN1, OUTPUT);
    pinMode(IN3, OUTPUT);
    stopMotors();

    // Attach servos
    panServo.attach(PAN_PIN);
    tiltServo.attach(TILT_PIN);
    panServo.write(panAngle);
    tiltServo.write(tiltAngle);

    Serial.println("Ready to receive");
    Serial.flush();
}
```

Метод *attach()* прив'язує об'єкт *Servo* до відповідного цифрового піна.

Метод *write()* встановлює сервопривід у початкову позицію, яка в цьому випадку дорівнює 90 градусам.

2.7. Додати алгоритм обробки команд керування сервоприводами до функції *handleCommand()*:

```
void handleCommand(const char* cmd) {
    if (strcmp(cmd, "w") == 0) {
        // Forward
        digitalWrite(IN1, HIGH);
        digitalWrite(IN3, HIGH);
        analogWrite(ENA, SPEED);
        analogWrite(ENB, SPEED);
    } else if (strcmp(cmd, "s") == 0) {
        // Backward
        digitalWrite(IN1, LOW);
        digitalWrite(IN3, LOW);
        analogWrite(ENA, SPEED);
        analogWrite(ENB, SPEED);
    } else if (strcmp(cmd, "a") == 0) {
        // Turn left
        digitalWrite(IN1, LOW);
        digitalWrite(IN3, HIGH);
        analogWrite(ENA, 0);
    }
}
```

```

    analogWrite(ENB, SPEED);
  } else if (strcmp(cmd, "d") == 0) {
    // Turn right
    digitalWrite(IN1, HIGH);
    digitalWrite(IN3, LOW);
    analogWrite(ENA, SPEED);
    analogWrite(ENB, 0);
  } else if (strcmp(cmd, "halt") == 0) {
    stopMotors();
  } else if (strncmp(cmd, "pan:", 4) == 0) {
    // Absolute pan angle
    int val = atoi(cmd + 4);
    panAngle = constrain(val, 0, 180);
    panServo.write(panAngle);
  } else if (strncmp(cmd, "tilt:", 5) == 0) {
    // Absolute tilt angle
    int val = atoi(cmd + 5);
    tiltAngle = constrain(val, 0, 180);
    tiltServo.write(tiltAngle);
  }
}
}

```

Під час обробки нових команд перевіряється, чи команда починається з «pan:» або «tilt:». Потім з рядка *cmd* після символу «:» виділяється числова частина. Функція *atoi()* перетворює виділений підрядок на ціле число. Функція *constrain(val, 0, 180)* обмежує значення в межах дозволеного діапазону. Далі сервопривід переміщається у задане положення.

2.8. Натиснути кнопку *Verify* (🔍) для попередньої компіляції та перевірки повного коду програми. Повний код виглядає так:

```

#include <Servo.h>

// Motor control pins (L298P Motor Shield)
const int ENA = 10; // PWM speed control for motor A
const int ENB = 11; // PWM speed control for motor B
const int IN1 = 12; // Direction for motor A
const int IN3 = 13; // Direction for motor B
const int SPEED = 255; // Max speed

// Servo pins
const int PAN_PIN = 9;
const int TILT_PIN = 6;
int panAngle = 90;
int tiltAngle = 90;

// Servo objects
Servo panServo;
Servo tiltServo;

// Serial buffer
char buffer[30];
int bufferIndex = 0;

```

```

void setup() {
    Serial.begin(9600);

    // Motor pins
    pinMode(ENA, OUTPUT);
    pinMode(ENB, OUTPUT);
    pinMode(IN1, OUTPUT);
    pinMode(IN3, OUTPUT);
    stopMotors();

    // Attach servos
    panServo.attach(PAN_PIN);
    tiltServo.attach(TILT_PIN);
    panServo.write(panAngle);
    tiltServo.write(tiltAngle);

    Serial.println("Ready to receive");
    Serial.flush();
}

void loop() {
    if (Serial.available()) {
        char inChar = Serial.read();
        if (inChar == '\n') {
            buffer[bufferIndex] = '\0';

            // Remove trailing \r if present
            if (bufferIndex > 0 && buffer[bufferIndex - 1] == '\r') {
                buffer[--bufferIndex] = '\0';
            }

            handleCommand(buffer);
            Serial.print(buffer);
            Serial.println();
            Serial.flush();
            bufferIndex = 0;
        } else if (bufferIndex < sizeof(buffer) - 1) {
            buffer[bufferIndex++] = inChar;
        }
    }
}


void handleCommand(const char* cmd) {
    if (strcmp(cmd, "w") == 0) {
        // Forward
        digitalWrite(IN1, HIGH);
        digitalWrite(IN3, HIGH);
        analogWrite(ENA, SPEED);
        analogWrite(ENB, SPEED);
    } else if (strcmp(cmd, "s") == 0) {
        // Backward
        digitalWrite(IN1, LOW);
        digitalWrite(IN3, LOW);
        analogWrite(ENA, SPEED);
        analogWrite(ENB, SPEED);
    } else if (strcmp(cmd, "a") == 0) {
        // Turn left
        digitalWrite(IN1, LOW);
        digitalWrite(IN3, HIGH);
    }
}

```

```

    analogWrite(ENA, 0);
    analogWrite(ENB, SPEED);
} else if (strcmp(cmd, "d") == 0) {
    // Turn right
    digitalWrite(IN1, HIGH);
    digitalWrite(IN3, LOW);
    analogWrite(ENA, SPEED);
    analogWrite(ENB, 0);
} else if (strcmp(cmd, "halt") == 0) {
    stopMotors();
} else if (strncmp(cmd, "pan:", 4) == 0) {
    // Absolute pan angle
    int val = atoi(cmd + 4);
    panAngle = constrain(val, 0, 180);
    panServo.write(panAngle);
} else if (strncmp(cmd, "tilt:", 5) == 0) {
    // Absolute tilt angle
    int val = atoi(cmd + 5);
    tiltAngle = constrain(val, 0, 180);
    tiltServo.write(tiltAngle);
}
}
}

```

2.9. Підключити плату Arduino Uno до ПК через USB. На час завантаження програми від'єднати лінію Tx ESP32-CAM від RX Arduino Uno. Натиснути кнопку *Upload* () для завантаження програми до мікропроцесора. У випадку коректного завантаження відключити Arduino Uno від ПК.

3. Створити графічний інтерфейс для програми керування роботом.

3.1. Запустити Qt Designer.

3.2. Створити новий графічний інтерфейс (за аналогією до попередніх лабораторних робіт).

3.3. Розмістити у графічному інтерфейсі елементи згідно з таблицею 3.1. Дизайн інтерфейсу розробити самостійно або зробити аналогічним до попередніх лабораторних робіт.

3.4. Зберегти інтерфейс під назвою «*pan-tilt_control.ui*».

4. Написати Python-програму для роботи зі створеним графічним інтерфейсом.

4.1. Відкрити будь-який редактор Python-коду або середовище розробки (наприклад, PyCharm) і створити новий *.py*-файл або проект. За основу взяти файл/проект з лабораторної роботи №1 (частина 2) та у подальшому вносити у нього необхідні зміни.

Таблиця 3.1. Опис віджетів графічного інтерфейсу керування роботом

Тип віджету	Назва (objectName)	Додаткові властивості
MainWindow	Form	<i>windowTitle</i> : "Pan-Tilt control"
Label	video_label	<i>maximumSize</i> : width=320, height=240 <i>scaledContents</i> : true
PushButton	start_stream_button	<i>text</i> : "Старт відео"
PushButton	stop_stream_button	<i>text</i> : "Стоп відео"
SpinBox	pan_spinbox	<i>maximum</i> : 180 <i>singleStep</i> : 5 <i>value</i> : 90
SpinBox	tilt_spinbox	<i>maximum</i> : 180 <i>singleStep</i> : 5 <i>value</i> : 90
PlainTextEdit	log_view	<i>wordWrap</i> : False <i>readOnly</i> : true <i>horizontalScrollBarPolicy</i> : AsNeeded <i>font</i> : Monospace <i>scrollBarsEnabled</i> : true

4.2. Додати імпорт класу *QSpinBox*:

```
from PyQt5.QtWidgets import QApplication, QWidget, QSpinBox
```

Клас *QSpinBox* дозволяє створювати елементи інтерфейсу для числового введення (наприклад, кути від 0 до 180 для сервоприводів).

4.3. Додати змінні для управління сервоприводами до класу *VideoControl*:

```
class VideoControl(QWidget):
    def __init__(self):
        super().__init__()
        uic.loadUi("pan-tilt_control.ui", self)

        # Video + log
        self.start_stream_button = self.findChild(QWidget,
                                                    "start_stream_button")
        self.stop_stream_button = self.findChild(QWidget,
                                                  "stop_stream_button")
        self.video_label = self.findChild(QWidget, "video_label")
        self.log_view = self.findChild(QWidget, "log_view")

        # Servo spinboxes
        self.pan_spinbox = self.findChild(QSpinBox, "pan_spinbox")
        self.tilt_spinbox = self.findChild(QSpinBox, "tilt_spinbox")

        self.pan_spinbox.setRange(0, 180)
        self.tilt_spinbox.setRange(0, 180)
        self.pan_spinbox.setValue(90)
        self.tilt_spinbox.setValue(90)
```



```

self.pan_spinbox.valueChanged.connect(self.on_pan_changed_sync)
self.tilt_spinbox.valueChanged.connect(self.on_tilt_changed_sync)

# Button callbacks
self.start_stream_button.clicked.connect(lambda:
                                          self.toggle_video(True))
self.stop_stream_button.clicked.connect(lambda:
                                         self.toggle_video(False))

self.ws = None
self.video_task = None
self.stream_active = False
self.latest_frame = None

self.pressed_keys = set()
asyncio.ensure_future(self.connect_ws_loop())

self.setFocusPolicy(QtCore.Qt.StrongFocus)

```

Створюються змінні *self.pan_spinbox* та *self.tilt_spinbox*, для яких потім встановлюється діапазон можливих значень від 0 до 180 (відповідає допустимим кутам сервоприводу) та початкове значення 90° (нейтральне положення). Коли значення в *QSpinBox* змінюється користувачем, викликаються відповідні методи обробки події. Оскільки асинхронні функції не можна напряму з'єднувати з сигналами PyQt, створено синхронні обгортки (*on_pan_changed_sync*, *on_tilt_changed_sync*), які всередині викликають відповідні *async*-методи.

4.4. Створити методи *on_pan_changed_sync()*, *on_tilt_changed_sync()*, *on_pan_changed()*, *on_tilt_changed()*:

```

def on_pan_changed_sync(self, value):
    asyncio.ensure_future(self.on_pan_changed(value))

def on_tilt_changed_sync(self, value):
    asyncio.ensure_future(self.on_tilt_changed(value))

async def on_pan_changed(self, value):
    await self.send_servo_command(f"pan:{value}")

async def on_tilt_changed(self, value):
    await self.send_servo_command(f"tilt:{180 - value}")

```

Метод *on_pan_changed()* надсилає WebSocket-команду «*pan:<значення>*», яка керує сервоприводом горизонтального обертання. Метод *on_tilt_changed()* надсилає команду «*tilt:<значення>*», де значення інвертується: $180 - value$. Це необхідно, щоб графічне зменшення (вниз) відповідало фізичному руху вверх.

4.5. Створити метод `send_servo_command()`:

```
async def send_servo_command(self, command):
    if not self.ws or self.ws.closed:
        self.append_log("WebSocket not connected")
        return
    try:
        await self.ws.send(command)
        self.append_log(f"Command sent: {command}")
    except Exception as e:
        self.append_log(f"Send error: {e}")
```

Метод призначений для надсилання команди керування сервоприводами на ESP32-CAM через WebSocket.

4.6. Провести тестовий запуск програми. Повний код програми:

```
import sys
import asyncio
import aiohttp
import websockets
import cv2
import numpy as np
from PyQt5 import uic, QtGui, QtCore
from PyQt5.QtWidgets import QApplication, QWidget, QSpinBox
from qasync import QEventLoop, asyncSlot

ESP32_WS_URL = "ws://192.168.31.81:81/ws"
ESP32_VIDEO_URL = "http://192.168.31.81/video"

KEY_COMMANDS = {
    QtCore.Qt.Key_W: 'w',
    QtCore.Qt.Key_A: 'a',
    QtCore.Qt.Key_S: 's',
    QtCore.Qt.Key_D: 'd',
}

class VideoControl(QWidget):
    def __init__(self):
        super().__init__()
        uic.loadUi("pan-tilt_control.ui", self)

        # Video + log
        self.start_stream_button = self.findChild(QWidget,
"start_stream_button")
        self.stop_stream_button = self.findChild(QWidget,
"stop_stream_button")
        self.video_label = self.findChild(QWidget, "video_label")
        self.log_view = self.findChild(QWidget, "log_view")

        # Servo spinboxes
        self.pan_spinbox = self.findChild(QSpinBox, "pan_spinbox")
        self.tilt_spinbox = self.findChild(QSpinBox, "tilt_spinbox")

        self.pan_spinbox.setRange(0, 180)
        self.tilt_spinbox.setRange(0, 180)
        self.pan_spinbox.setValue(90)
        self.tilt_spinbox.setValue(90)

        self.pan_spinbox.valueChanged.connect(self.on_pan_changed_sync)
        self.tilt_spinbox.valueChanged.connect(self.on_tilt_changed_sync)
```

```

        # Button callbacks
        self.start_stream_button.clicked.connect(lambda:
self.toggle_video(True))
        self.stop_stream_button.clicked.connect(lambda:
self.toggle_video(False))

        self.ws = None
        self.video_task = None
        self.stream_active = False
        self.latest_frame = None

        self.pressed_keys = set()
        asyncio.ensure_future(self.connect_ws_loop())

        self.setFocusPolicy(QtCore.Qt.StrongFocus)

    def append_log(self, msg):
self.log_view.appendPlainText(f"[{QtCore.QTime.currentTime().toString()}]
{msg}")

    async def connect_ws_loop(self):
        while True:
            if self.ws is None or self.ws.closed:
                try:
                    self.ws = await websockets.connect(ESP32_WS_URL)
                    self.append_log("Connected to WebSocket")
                    asyncio.ensure_future(self.receive_ws())
                except Exception as e:
                    self.append_log(f"WebSocket error: {e}")
                await asyncio.sleep(0.5)

    async def receive_ws(self):
        try:
            async for message in self.ws:
                self.append_log(f"ESP32: {message}")
        except Exception as e:
            self.append_log(f"WebSocket lost: {e}")
            self.ws = None

    @asyncSlot()
    async def toggle_video(self, enable):
        if not self.ws or self.ws.closed:
            self.append_log("WebSocket not connected ")
            return
        try:
            await self.ws.send("start" if enable else "stop")
            self.stream_active = enable
            if enable:
                if self.video_task:
                    self.video_task.cancel()
                self.video_task =
asyncio.ensure_future(self.video_stream_task())
            else:
                if self.video_task:
                    self.video_task.cancel()
                self.video_label.clear()
                self.latest_frame = None
        except Exception as e:
            self.append_log(f"WebSocket error: {e}")

    async def video_stream_task(self):
        try:
            async with aiohttp.ClientSession() as session:

```

```

        async with session.get(ESP32_VIDEO_URL) as resp:
            if resp.status != 200:
                self.append_log(f"HTTP error: {resp.status}")
                return

            buffer = b""
            while self.stream_active:
                chunk = await resp.content.read(4096)
                if not chunk:
                    break
                buffer += chunk

            start = buffer.find(b'\xff\xd8')
            end = buffer.find(b'\xff\xd9', start)
            if start != -1 and end != -1:
                jpeg = buffer[start:end + 2]
                buffer = buffer[end + 2:]

            img_np = np.frombuffer(jpeg, np.uint8)
            frame = cv2.imdecode(img_np, cv2.IMREAD_COLOR)
            if frame is not None:
                frame = cv2.cvtColor(frame,
cv2.COLOR_BGR2RGB)

                self.latest_frame = QtGui.QImage(
                    frame.data,
                    frame.shape[1],
                    frame.shape[0],
                    frame.strides[0],
                    QtGui.QImage.Format_RGB888
                )
                self.update_frame()
            except Exception as e:
                self.append_log(f"Video error: {e}")
            finally:
                self.stream_active = False
                self.video_label.setText("Stream stopped")

    def update_frame(self):
        if self.latest_frame and not self.latest_frame.isNull():
            pix = QtGui.QPixmap.fromImage(self.latest_frame).scaled(
                self.video_label.size(),
                QtCore.Qt.KeepAspectRatio,
                QtCore.Qt.SmoothTransformation
            )
            self.video_label.setPixmap(pix)

    def resizeEvent(self, event):
        super().resizeEvent(event)
        if self.latest_frame is not None:
            QtCore.QTimer.singleShot(30, self.update_frame)

    def closeEvent(self, event):
        self.stream_active = False
        if self.video_task:
            self.video_task.cancel()
        if self.ws:
            asyncio.ensure_future(self.ws.close())
        event.accept()

    def keyPressEvent(self, event):
        if event.isAutoRepeat():
            return
        key = event.key()
        if key in KEY_COMMANDS and key not in self.pressed_keys:

```

```

        self.pressed_keys.add(key)

asyncio.ensure_future(self.send_drive_command(KEY_COMMANDS[key]))

def keyReleaseEvent(self, event):
    if event.isAutoRepeat():
        return
    key = event.key()
    if key in self.pressed_keys:
        self.pressed_keys.remove(key)
    if not self.pressed_keys:
        asyncio.ensure_future(self.send_drive_command("halt"))

async def send_drive_command(self, key):
    if not self.ws or self.ws.closed:
        return
    try:
        await self.ws.send(key)
        self.append_log(f"Command sent: {key}")
    except Exception as e:
        self.append_log(f"Send error: {e}")

async def send_servo_command(self, command):
    if not self.ws or self.ws.closed:
        self.append_log("WebSocket not connected")
        return
    try:
        await self.ws.send(command)
        self.append_log(f"Command sent: {command}")
    except Exception as e:
        self.append_log(f"Send error: {e}")

def on_pan_changed_sync(self, value):
    asyncio.ensure_future(self.on_pan_changed(value))

def on_tilt_changed_sync(self, value):
    asyncio.ensure_future(self.on_tilt_changed(value))

async def on_pan_changed(self, value):
    await self.send_servo_command(f"pan:{value}")

async def on_tilt_changed(self, value):
    await self.send_servo_command(f"tilt:{180 - value}")

def main():
    app = QApplication(sys.argv)
    loop = QEventLoop(app)
    asyncio.set_event_loop(loop)

    window = VideoControl()
    window.show()

    with loop:
        loop.run_forever()

if __name__ == "__main__":
    main()

```

5. Зібрати схему, показану на рис. 3.2, та перевірити правильність роботи системи. Для цього підключити всі компоненти до живлення, запустити GUI

та перевірити результат встановлення з'єднання з ESP32-CAM. Використовуючи створений графічний інтерфейс, увімкнути трансляцію відео з ESP32-CAM та протестувати керування рухом робота з клавіатури та механізмом Pan-Tilt, натискаючи відповідні кнопки.

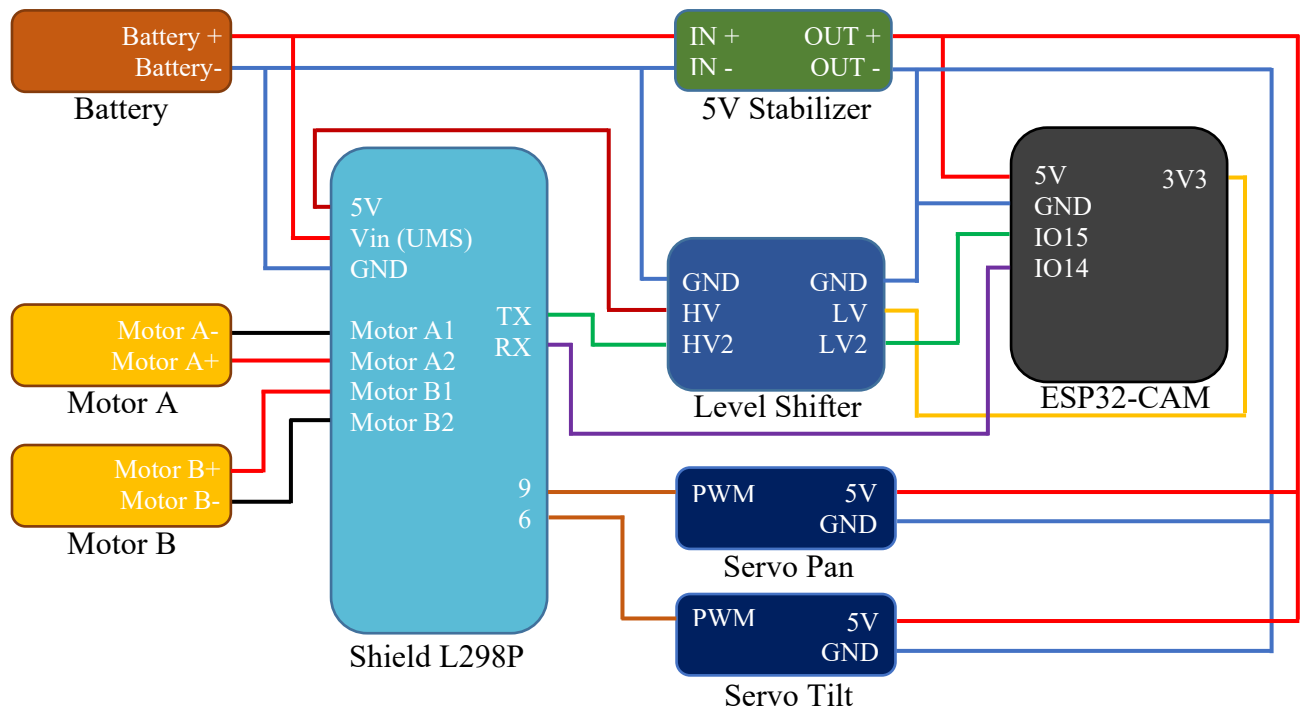


Рис. 3.2. Схема підключення компонентів робота та Pan-Tilt механізму

Контрольні запитання

1. Опишіть основні компоненти механізму Pan-Tilt і поясніть, як вони взаємодіють з іншими елементами системи.
2. Як в коді для Arduino Uno реалізовано керування серводвигунами?
3. Поясніть, як у Python-програмі реалізовано керування кутами серводвигунів.
4. Як можна модифікувати систему, щоб додати можливість автоматичного повернення камери до початкового положення (90° для pan і tilt) при натисканні певної кнопки в інтерфейсі програми?

АВТОМАТИЧНЕ ДЕТЕКТУВАННЯ ТА ВІДСТЕЖЕННЯ ОБ'ЄКТІВ МОБІЛЬНИМ РОБОТОМ

Мета: Ознайомлення з принципами автоматичного детектування та відстеження об'єктів за допомогою моделі комп'ютерного зору YOLOv8 і механізму Pan-Tilt. Вивчення інтеграції апаратного забезпечення, мережових технологій і програмного забезпечення для реалізації дистанційного керування роботом і наведення камери на цільові об'єкти в реальному часі..

Теоретичні відомості

Система, представлена в даній частині лабораторної роботи, призначена для дистанційного керування рухом робота та автоматичного наведення камери на об'єкти за допомогою механізму Pan-Tilt, який забезпечує поворот і нахил камери. Вона дозволяє користувачу керувати рухом робота за допомогою клавіатури, а також автоматично спрямовувати камеру на виявлені об'єкти, використовуючи моделі глибинного навчання. Система демонструє інтеграцію апаратного забезпечення, мережових технологій і нейронних мереж для реалізації керування в реальному часі.

Основними компонентами системи, як і раніше, є мікроконтролер ESP32-CAM, плата Arduino Uno з моторним щитом L298P, два серводвигуни для механізму Pan-Tilt і програмне забезпечення на базі Python із графічним інтерфейсом, створеним за допомогою PyQt. Програмне забезпечення на ПК відображає відеопотік, обробляє його за допомогою моделі YOLOv8 для виявлення об'єктів, і автоматично надсилає команди для наведення камери на цільовий об'єкт. Графічний інтерфейс забезпечує кнопки для запуску та зупинки відеотрансляції, клавіші для керування рухом і журнал подій для моніторингу.

ESP32-CAM підключається до WiFi-мережі, запускає HTTP-сервер на порту 80 для передачі відеопотоку і WebSocket-сервер на порту 81 для обміну командами. Користувач через графічний інтерфейс може запускати або зупиняти відеотрансляцію та керувати рухом робота за допомогою клавіш W, A, S, D. Відеопотік обробляється на комп'ютері за допомогою YOLOv8, яка ідентифікує об'єкти і визначає їхні координати на зображенні. Якщо виявлено заданий об'єкт, система розраховує відхилення його центру від середини кадру (рис. 3.3) і надсилає команди через WebSocket до ESP32-CAM, а потім через UART до Arduino Uno для корекції кутів серводвигунів, щоб центр об'єкта залишався в середині кадру. Arduino обробляє ці команди, змінюючи положення серводвигунів для наведення камери.

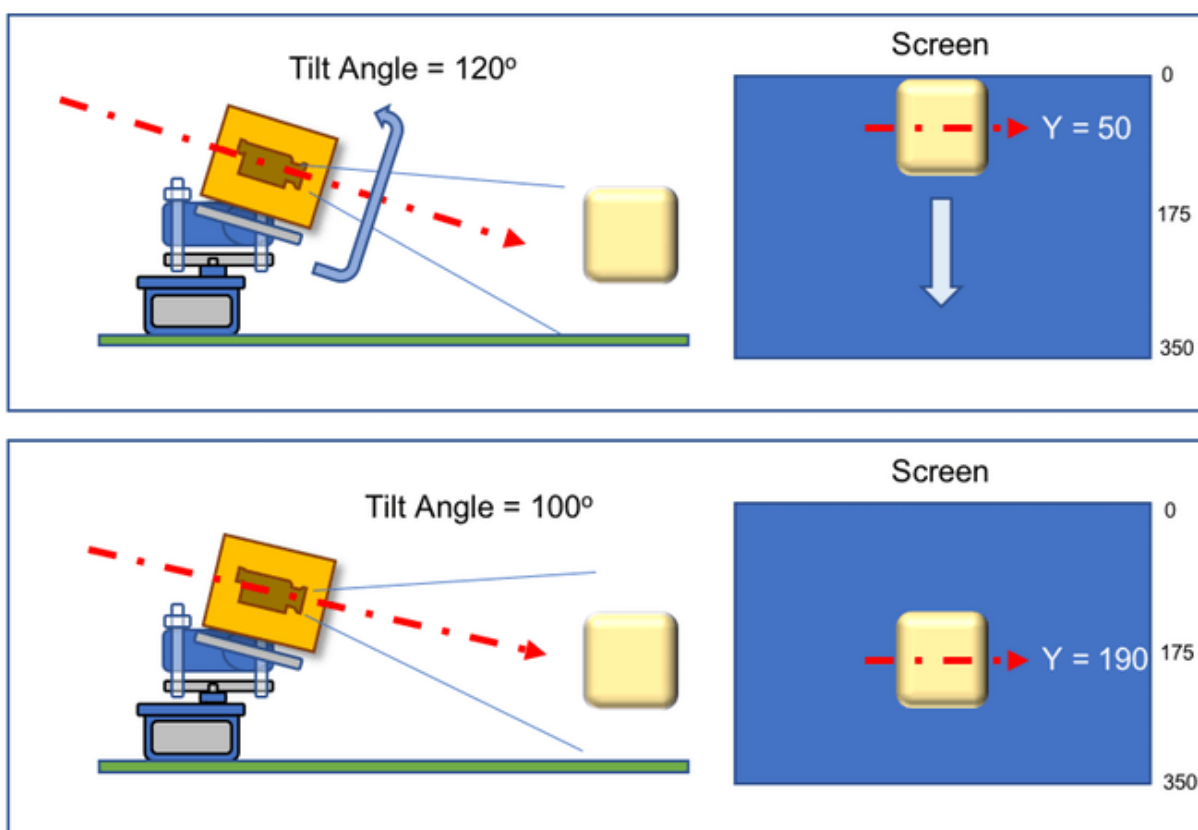


Рис. 3.3. Принцип наведення камери на визначений об'єкт

YOLOv8 (You Only Look Once, версія 8) – це сучасна модель комп'ютерного зору для виявлення об'єктів, розроблена компанією Ultralytics. Вона належить до сімейства моделей YOLO, відомих своєю швидкістю та достовірністю в задачах обробки зображень у реальному часі. YOLOv8 є однією

з найшвидших версій, яка використовує досягнення в глибинному навчанні для забезпечення високої продуктивності в задачах виявлення, сегментації та класифікації об'єктів. У контексті лабораторної роботи застосовується модель YOLOv8 у форматі PyTorch (yolov8n.pt, нано-версія), яка оптимізує швидкість обробки для роботи на обмежених обчислювальних ресурсах, таких як ПК без графічної карти або мікрокомп'ютери.

У лабораторній роботі YOLOv8 використовується для детектування об'єктів у відеопотоці. Модель обробляє кожен кадр, отриманий із камери ESP32-CAM, і повертає координати обмежувальних рамок (x_1, y_1, x_2, y_2), ідентифікатори класів (*cls*) і ймовірності впевненості (*conf*). У кодї Python програма аналізує результати, малює рамки навколо виявлених об'єктів із мітками визначеного класу і обирає об'єкт із найбільшою площею рамки, якщо він відноситься до заданого класу. Координати центру цього об'єкта використовуються для автоматичного наведення камери через механізм Pan-Tilt. Модель налаштована з порогом впевненості ($conf=0.5$), що означає, що об'єкти з ймовірністю менш ніж 50% ігноруються, зменшуючи кількість хибних спрацьовувань. YOLOv8 підтримує попередньо навчені ваги для 80 класів із набору даних COCO.

Механізм Pan-Tilt в даній роботі використовується для автоматичного наведення камери на заданий об'єкт, виявлений YOLOv8. Спочатку YOLOv8, повертає координати обмежувальних рамок для виявлених об'єктів. Якщо серед них є об'єкт класу заданого класу, розраховується центр рамки (s_x, s_y) як середнє арифметичне координат $(x_1+x_2)/2$ і $(y_1+y_2)/2$. Далі визначається відхилення цього центру від середини кадру. Якщо відхилення по горизонталі або вертикалі перевищує 10 пікселів, кути pan і tilt коригуються на ± 1 градус залежно від напрямку відхилення: для горизонталі – зменшення кута при русі вправо, збільшення – вліво; для вертикалі – збільшення кута при русі вниз, зменшення – вгору. Ці кути обмежуються діапазоном 0–180 градусів, і команди керування надсилаються через WebSocket до ESP32-CAM, а потім через UART до Arduino для виконання.

Хід виконання роботи

1. Коди програм для ESP32-CAM та Arduino Uno в даній лабораторній роботі нічим не будуть відрізнятись від останній версій кодів з попередньої роботи. Тому змінювати їх не потрібно.
2. Створити графічний інтерфейс для програми детектування об'єктів на відео.
 - 2.1. Запустити Qt Designer.
 - 2.2. Створити новий графічний інтерфейс (за аналогією до попередніх лабораторних робіт).
 - 2.3. Розмістити у графічному інтерфейсі елементи згідно з таблицею 3.2. Розміщення та візуальний стиль елементів обрати самостійно або орієнтуючись на аналогічний інтерфейс з попередньої частини лабораторної роботи (достатньо прибрати елементи SpinBox).

Таблиця 3.2. Опис віджетів графічного інтерфейсу

Тип віджету	Назва (objectName)	Додаткові властивості
MainWindow	Form	<i>windowTitle</i> : "Object Tracking Robot"
Label	video_label	<i>maximumSize</i> : width=320, height=240 <i>scaledContents</i> : true
PushButton	start_stream_button	<i>text</i> : "Старт відео"
PushButton	stop_stream_button	<i>text</i> : "Стоп відео"
PlainTextEdit	log_view	<i>wordWrap</i> : False <i>readOnly</i> : true <i>horizontalScrollBarPolicy</i> : AsNeeded <i>font</i> : Monospace <i>scrollBarsEnabled</i> : true

- 2.4. Зберегти інтерфейс під назвою «*pan-tilt_autocontrol.ui*».
3. Написати Python-програму для роботи зі створеним графічним інтерфейсом.
 - 3.1. Відкрити будь-який редактор Python-коду або середовище розробки (наприклад, PyCharm) і створити новий .py-файл або проект. За основу

взяти файл/проект з попередньої частини лабораторної роботи та у подальшому вносити у нього необхідні зміни.

3.2. Імпортувати модель YOLO:

```
from ultralytics import YOLO
```

Цей рядок імпортує клас YOLO з бібліотеки *ultralytics*. Він дозволяє завантажити модель YOLOv8 та використовувати її для детектування об'єктів на зображеннях.

3.3. У конструкторі класу *VideoControl* створити нову змінну:

```
class VideoControl(QWidget):
    def __init__(self):
        super().__init__()
        uic.loadUi("pan-tilt_autocontrol.ui", self)

        # Load YOLOv8 nano model (PyTorch format)
        self.yolo_model = YOLO("yolov8n.pt")
```

self.yolo_model – це об'єкт моделі YOLOv8, завантаженої з файлу "yolov8n.pt" (найменша версія YOLOv8, підвантажиться автоматично).

3.4. Оновити метод *video_stream_task()*:

```
async def video_stream_task(self):
    try:
        async with aiohttp.ClientSession() as session:
            async with session.get(ESP32_VIDEO_URL) as resp:
                if resp.status != 200:
                    self.append_log(f"HTTP error: {resp.status}")
                    return

        buffer = b""
        while self.stream_active:
            chunk = await resp.content.read(4096)
            if not chunk:
                break
            buffer += chunk
            start = buffer.find(b'\xff\xd8')
            end = buffer.find(b'\xff\xd9', start)
            if start != -1 and end != -1:
                jpeg = buffer[start:end + 2]
                buffer = buffer[end + 2:]
                img_np = np.frombuffer(jpeg, np.uint8)
                frame = cv2.imdecode(img_np, cv2.IMREAD_COLOR)
                if frame is not None:
                    # Perform detection
                    frame = self.process_yolo(frame)
                    frame = cv2.cvtColor(frame,
                                         cv2.COLOR_BGR2RGB)
                    self.latest_frame = QtGui.QImage(
                        frame.data,
                        frame.shape[1],
                        frame.shape[0],
                        frame.strides[0],
                        QtGui.QImage.Format_RGB888
                    )
```

```

        self.update_frame()
    except Exception as e:
        self.append_log(f"Video error: {e}")
    finally:
        self.stream_active = False
        self.video_label.setText("Stream stopped")

```

Рядок `frame = self.process_yolo(frame)` викликає обробку кадру через модель YOLO після його декодування з потоку MJPEG. Метод `process_yolo()` повертає той самий кадр з накладеними прямокутниками та написами для виявлених об'єктів.

3.5. Створити новий метод `process_yolo()`:

```

def process_yolo(self, frame):
    # Run YOLO detection (input size automatically handled)
    results = self.yolo_model(frame, imgsiz=320, conf=0.5)

    for r in results:
        for box in r.bboxes:
            x1, y1, x2, y2 = map(int, box.xyxy[0])
            cls = int(box.cls[0])
            conf = float(box.conf[0])
            label = f"{self.yolo_model.names[cls]} {conf:.2f}"

            # Draw bounding box
            cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
            cv2.putText(frame, label, (x1, y1 - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

    return frame

```

Цей метод виконує детектування об'єктів за допомогою YOLOv8 на вхідному кадрі та малює прямокутники з підписами навколо знайдених об'єктів. Спочатку метод запускає модель на кадрі, де параметр `imgsiz=320` задає розмір зображення, а параметр `conf=0.5` – поріг детектування. Далі для кожного результату (один на кадр) обробляються всі виявлені об'єкти (`boxes`) та отримуються їх координати `x1, y1, x2, y2`. Це координати прямокутника навколо об'єкта. Потім для кожного об'єкту визначається код класу (наприклад, "person", "car") і достовірність. Після цього формується підпис і малюється прямокутник з цим підписом на кадрі.

3.6. Видалити з коду всі змінні та методи, які стосувались ручного керування серводвигунами (`self.pan_spinbox, self.tilt_spinbox, on_pan_changed_sync, on_tilt_changed_sync, on_pan_changed, on_tilt_changed, send_servo_command`).

3.7. Провести тестовий запуск програми. Повний код програми:

```

import sys
import asyncio
import aiohttp
import websockets
import cv2
import numpy as np
from PyQt5 import uic, QtGui, QtCore
from PyQt5.QtWidgets import QApplication, QWidget
from qasync import QEventLoop, asyncSlot
from ultralytics import YOLO

ESP32_WS_URL = "ws://192.168.31.81:81/ws"
ESP32_VIDEO_URL = "http://192.168.31.81/video"

KEY_COMMANDS = {
    QtCore.Qt.Key_W: 'w',
    QtCore.Qt.Key_A: 'a',
    QtCore.Qt.Key_S: 's',
    QtCore.Qt.Key_D: 'd',
}

class VideoControl(QWidget):
    def __init__(self):
        super().__init__()
        uic.loadUi("pan-tilt_autocontrol.ui", self)

        # Load YOLOv8 nano model (PyTorch format)
        self.yolo_model = YOLO("yolov8n.pt")

        # Video + log
        self.start_stream_button = self.findChild(QWidget,
"start_stream_button")
        self.stop_stream_button = self.findChild(QWidget,
"stop_stream_button")
        self.video_label = self.findChild(QWidget, "video_label")
        self.log_view = self.findChild(QWidget, "log_view")

        # Button callbacks
        self.start_stream_button.clicked.connect(lambda:
self.toggle_video(True))
        self.stop_stream_button.clicked.connect(lambda:
self.toggle_video(False))

        self.ws = None
        self.video_task = None
        self.stream_active = False
        self.latest_frame = None
        self.pressed_keys = set()

        asyncio.ensure_future(self.connect_ws_loop())
        self.setFocusPolicy(QtCore.Qt.StrongFocus)

    def append_log(self, msg):
self.log_view.appendPlainText(f"[{QtCore.QTime.currentTime().toString()}]
{msg}")

    async def connect_ws_loop(self):
        while True:
            if self.ws is None or self.ws.closed:
                try:
                    self.ws = await websockets.connect(ESP32_WS_URL)
                    self.append_log("Connected to WebSocket")

```

```

        asyncio.ensure_future(self.receive_ws())
    except Exception as e:
        self.append_log(f"WebSocket error: {e}")
    await asyncio.sleep(0.5)

async def receive_ws(self):
    try:
        async for message in self.ws:
            self.append_log(f"ESP32: {message}")
    except Exception as e:
        self.append_log(f"WebSocket lost: {e}")
        self.ws = None

@asyncSlot()
async def toggle_video(self, enable):
    if not self.ws or self.ws.closed:
        self.append_log("WebSocket not connected")
        return
    try:
        await self.ws.send("start" if enable else "stop")
        self.stream_active = enable
        if enable:
            if self.video_task:
                self.video_task.cancel()
            self.video_task =
asyncio.ensure_future(self.video_stream_task())
        else:
            if self.video_task:
                self.video_task.cancel()
            self.video_label.clear()
            self.latest_frame = None
    except Exception as e:
        self.append_log(f"WebSocket error: {e}")

async def video_stream_task(self):
    try:
        async with aiohttp.ClientSession() as session:
            async with session.get(ESP32_VIDEO_URL) as resp:
                if resp.status != 200:
                    self.append_log(f"HTTP error: {resp.status}")
                    return

        buffer = b""
        while self.stream_active:
            chunk = await resp.content.read(4096)
            if not chunk:
                break
            buffer += chunk

            start = buffer.find(b'\xff\xd8')
            end = buffer.find(b'\xff\xd9', start)
            if start != -1 and end != -1:
                jpeg = buffer[start:end + 2]
                buffer = buffer[end + 2:]

            img_np = np.frombuffer(jpeg, np.uint8)
            frame = cv2.imdecode(img_np, cv2.IMREAD_COLOR)
            if frame is not None:
                # Perform detection
                frame = self.process_yolo(frame)
                frame = cv2.cvtColor(frame,
cv2.COLOR_BGR2RGB)

                self.latest_frame = QtGui.QImage(
                    frame.data,

```

```

        frame.shape[1],
        frame.shape[0],
        frame.strides[0],
        QtGui.QImage.Format_RGB888
    )
    self.update_frame()
except Exception as e:
    self.append_log(f"Video error: {e}")
finally:
    self.stream_active = False
    self.video_label.setText("Stream stopped")

def process_yolo(self, frame):
    # Run YOLO detection (input size automatically handled)
    results = self.yolo_model(frame, imgsz=320, conf=0.5)

    for r in results:
        for box in r.bboxes:
            x1, y1, x2, y2 = map(int, box.xyxy[0])
            cls = int(box.cls[0])
            conf = float(box.conf[0])
            label = f"{self.yolo_model.names[cls]} {conf:.2f}"

            # Draw bounding box
            cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
            cv2.putText(frame, label, (x1, y1 - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

    return frame

def update_frame(self):
    if self.latest_frame and not self.latest_frame.isNull():
        pix = QtGui.QPixmap.fromImage(self.latest_frame).scaled(
            self.video_label.size(),
            QtCore.Qt.KeepAspectRatio,
            QtCore.Qt.SmoothTransformation
        )
        self.video_label.setPixmap(pix)

def resizeEvent(self, event):
    super().resizeEvent(event)
    if self.latest_frame is not None:
        QtCore.QTimer.singleShot(30, self.update_frame)

def closeEvent(self, event):
    self.stream_active = False
    if self.video_task:
        self.video_task.cancel()
    if self.ws:
        asyncio.ensure_future(self.ws.close())
    event.accept()

def keyPressEvent(self, event):
    if event.isAutoRepeat():
        return
    key = event.key()
    if key in KEY_COMMANDS and key not in self.pressed_keys:
        self.pressed_keys.add(key)

    asyncio.ensure_future(self.send_drive_command(KEY_COMMANDS[key]))

def keyReleaseEvent(self, event):
    if event.isAutoRepeat():
        return
    key = event.key()

```

```

        if key in self.pressed_keys:
            self.pressed_keys.remove(key)
            if not self.pressed_keys:
                asyncio.ensure_future(self.send_drive_command("halt"))

    async def send_drive_command(self, key):
        if not self.ws or self.ws.closed:
            return
        try:
            await self.ws.send(key)
            self.append_log(f"Command sent: {key}")
        except Exception as e:
            self.append_log(f"Send error: {e}")

def main():
    app = QApplication(sys.argv)
    loop = QEventLoop(app)
    asyncio.set_event_loop(loop)

    window = VideoControl()
    window.show()

    with loop:
        loop.run_forever()

if __name__ == "__main__":
    main()

```

4. Перевірити правильність роботи системи. Для цього підключити всі компоненти до живлення, запустити GUI та перевірити результат встановлення з'єднання з ESP32-CAM. Використовуючи створений графічний інтерфейс, увімкнути трансляцію відео з ESP32-CAM та відслідкувати результат розпізнавання об'єктів на кадрах.
5. Модифікувати програму таким чином, щоб камера автоматично наводилась на об'єкт заданого класу (наприклад, смартфон). Для наведення має вибиратись той об'єкт заданого класу, який знаходиться найближче до камери (має найбільшу за розмірами рамку).

5.1. У конструкторі класу *VideoControl* додати нові змінні:

```

class VideoControl(QWidget):
    def __init__(self):
        super().__init__()
        uic.loadUi("pan-tilt_autocontrol.ui", self)

        # Load YOLOv8 nano model (PyTorch format)
        self.yolo_model = YOLO("yolov8n.pt")

        # Video + log
        self.start_stream_button = self.findChild(QWidget,
                                                    "start_stream_button")
        self.stop_stream_button = self.findChild(QWidget,

```



```

        "stop_stream_button")
self.video_label = self.findChild(QWidget, "video_label")
self.log_view = self.findChild(QWidget, "log_view")

# Button callbacks
self.start_stream_button.clicked.connect(lambda:
                                          self.toggle_video(True))
self.stop_stream_button.clicked.connect(lambda:
                                         self.toggle_video(False))

self.ws = None
self.video_task = None
self.stream_active = False
self.latest_frame = None
self.pressed_keys = set()
self.pan_angle = 90
self.tilt_angle = 90

asyncio.ensure_future(self.connect_ws_loop())
self.setFocusPolicy(QtCore.Qt.StrongFocus)

```

Змінні *self.pan_angle* та *self.tilt_angle* ведені для збереження поточних значень кутів повороту (pan) та нахилу (tilt) камери. Вони використовуються для автоматичного коригування положення камери згідно з положенням виявленого об'єкта, щоб уникнути повторного надсилання однакових команд і забезпечити поступову зміну кута.

5.2. Створити новий метод *get_color_for_class()*:

```

def get_color_for_class(self, cls_id):
    np.random.seed(cls_id)
    color = np.random.randint(0, 255, size=3).tolist()
    return tuple(int(c) for c in color)

```

Цей метод генерує окремий колір прямокутника для кожного класу об'єкта, який виявляє YOLO. Спочатку ініціалізується генератор випадкових чисел NumPy псевдовипадковим числом *cls_id* (ідентифікатор класу). Потім він генерує трійку випадкових значень (R, G, B) в межах 0–255 та повертає колір як кортеж цілих чисел.

5.3. Модифікувати метод *process_yolo()*:

```

def process_yolo(self, frame):
    results = self.yolo_model(frame, imgsz=320, conf=0.5)
    frame_h, frame_w = frame.shape[:2]
    max_area = 0
    target_center = None

    for r in results:
        for box in r.bboxes:
            x1, y1, x2, y2 = map(int, box.xyxy[0])
            cls = int(box.cls[0])
            conf = float(box.conf[0])
            label = f"{self.yolo_model.names[cls]} {conf:.2f}"
            color = self.get_color_for_class(cls)

```

```

cv2.rectangle(frame, (x1, y1), (x2, y2), color, 2)
cv2.putText(frame, label, (x1, y1 - 10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

if self.yolo_model.names[cls] == "cell phone":
    area = (x2 - x1) * (y2 - y1)
    if area > max_area:
        max_area = area
        target_center = ((x1 + x2) // 2, (y1 + y2) // 2)

if target_center:
    cx, cy = target_center
    offset_x = cx - frame_w // 2
    offset_y = cy - frame_h // 2

    if abs(offset_x) > 10:
        self.pan_angle += -1 if offset_x > 0 else 1
        self.pan_angle = max(0, min(180, self.pan_angle))
        asyncio.ensure_future(
            self.send_drive_command(f"pan:{self.pan_angle}")
        )

    if abs(offset_y) > 10:
        self.tilt_angle += 1 if offset_y > 0 else -1
        self.tilt_angle = max(0, min(180, self.tilt_angle))
        asyncio.ensure_future(
            self.send_drive_command(f"tilt:{self.tilt_angle}")
        )

return frame

```

Цей метод був суттєво доповнений і переведений на логіку наведення камери на найбільший виявлений об'єкт класу "cell phone". Спочатку завантажене зображення передається у модель YOLO для детектування об'єктів. Потім ініціалізуються змінні:

frame_h, frame_w – висота та ширина зображення (в пікселях).

max_area – площа найбільшого об'єкту класу "cell phone".

target_center – координати центру цього найбільшого об'єкта.

Далі в циклі малюється прямокутник навколо кожного об'єкта та ставиться відповідна текстова мітка. Якщо клас виявленого об'єкта – "cell phone", обчислюється площа рамки. Серед усіх таких об'єктів вибирається найбільший (за площею). Зберігається координата його центру. Після цього обчислюється зміщення центра об'єкта відносно центра кадру. Якщо центр об'єкта правіше/лівше центра кадру більше ніж на 10 пікселів, змінюється *pan_angle* на ± 1 градус. Команда з новим кутом надсилається через WebSocket на ESP32. Аналогічно і для *tilt_angle*.

5.4. Провести тестовий запуск програми. Повний код програми:

```

import sys
import asyncio
import aiohttp
import websockets
import cv2
import numpy as np
from PyQt5 import uic, QtGui, QtCore
from PyQt5.QtWidgets import QApplication, QWidget
from qasync import QEventLoop, asyncSlot
from ultralytics import YOLO

ESP32_WS_URL = "ws://192.168.31.81:81/ws"
ESP32_VIDEO_URL = "http://192.168.31.81/video"

KEY_COMMANDS = {
    QtCore.Qt.Key_W: 'w',
    QtCore.Qt.Key_A: 'a',
    QtCore.Qt.Key_S: 's',
    QtCore.Qt.Key_D: 'd',
}

class VideoControl(QWidget):
    def __init__(self):
        super().__init__()
        uic.loadUi("pan-tilt_autocontrol.ui", self)

        # Load YOLOv8 nano model (PyTorch format)
        self.yolo_model = YOLO("yolov8n.pt")

        # Video + log
        self.start_stream_button = self.findChild(QWidget,
"start_stream_button")
        self.stop_stream_button = self.findChild(QWidget,
"stop_stream_button")
        self.video_label = self.findChild(QWidget, "video_label")
        self.log_view = self.findChild(QWidget, "log_view")

        # Button callbacks
        self.start_stream_button.clicked.connect(lambda:
self.toggle_video(True))
        self.stop_stream_button.clicked.connect(lambda:
self.toggle_video(False))

        self.ws = None
        self.video_task = None
        self.stream_active = False
        self.latest_frame = None
        self.pressed_keys = set()
        self.pan_angle = 90
        self.tilt_angle = 90

        asyncio.ensure_future(self.connect_ws_loop())
        self.setFocusPolicy(QtCore.Qt.StrongFocus)

    def append_log(self, msg):
self.log_view.appendPlainText(f"[{QtCore.QTime.currentTime().toString()}]
{msg}")

    async def connect_ws_loop(self):
        while True:
            if self.ws is None or self.ws.closed:
                try:

```

```

        self.ws = await websockets.connect(ESP32_WS_URL)
        self.append_log("Connected to WebSocket")
        asyncio.ensure_future(self.receive_ws())
    except Exception as e:
        self.append_log(f"WebSocket error: {e}")
    await asyncio.sleep(0.5)

async def receive_ws(self):
    try:
        async for message in self.ws:
            self.append_log(f"ESP32: {message}")
    except Exception as e:
        self.append_log(f"WebSocket lost: {e}")
        self.ws = None

@asyncSlot()
async def toggle_video(self, enable):
    if not self.ws or self.ws.closed:
        self.append_log("WebSocket not connected")
        return
    try:
        await self.ws.send("start" if enable else "stop")
        self.stream_active = enable
        if enable:
            if self.video_task:
                self.video_task.cancel()
            self.video_task =
asyncio.ensure_future(self.video_stream_task())
        else:
            if self.video_task:
                self.video_task.cancel()
            self.video_label.clear()
            self.latest_frame = None
    except Exception as e:
        self.append_log(f"WebSocket error: {e}")

async def video_stream_task(self):
    try:
        async with aiohttp.ClientSession() as session:
            async with session.get(ESP32_VIDEO_URL) as resp:
                if resp.status != 200:
                    self.append_log(f"HTTP error: {resp.status}")
                    return

                buffer = b""
                while self.stream_active:
                    chunk = await resp.content.read(4096)
                    if not chunk:
                        break
                    buffer += chunk

                start = buffer.find(b'\xff\xd8')
                end = buffer.find(b'\xff\xd9', start)
                if start != -1 and end != -1:
                    jpeg = buffer[start:end + 2]
                    buffer = buffer[end + 2:]

                img_np = np.frombuffer(jpeg, np.uint8)
                frame = cv2.imdecode(img_np, cv2.IMREAD_COLOR)
                if frame is not None:
                    frame = self.process_yolo(frame)
                    frame = cv2.cvtColor(frame,
cv2.COLOR_BGR2RGB)

                    self.latest_frame = QtGui.QImage(

```

```

        frame.data,
        frame.shape[1],
        frame.shape[0],
        frame.strides[0],
        QtGui.QImage.Format_RGB888
    )
    self.update_frame()
except Exception as e:
    self.append_log(f"Video error: {e}")
finally:
    self.stream_active = False
    self.video_label.setText("Stream stopped ")

def get_color_for_class(self, cls_id):
    np.random.seed(cls_id)
    color = np.random.randint(0, 255, size=3).tolist()
    return tuple(int(c) for c in color)

def process_yolo(self, frame):
    results = self.yolo_model(frame, imgsz=320, conf=0.5)
    frame_h, frame_w = frame.shape[:2]

    max_area = 0
    target_center = None

    for r in results:
        for box in r.bboxes:
            x1, y1, x2, y2 = map(int, box.xyxy[0])
            cls = int(box.cls[0])
            conf = float(box.conf[0])
            label = f"{self.yolo_model.names[cls]} {conf:.2f}"
            color = self.get_color_for_class(cls)

            cv2.rectangle(frame, (x1, y1), (x2, y2), color, 2)
            cv2.putText(frame, label, (x1, y1 - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

            if self.yolo_model.names[cls] == "cell phone":
                area = (x2 - x1) * (y2 - y1)
                if area > max_area:
                    max_area = area
                    target_center = ((x1 + x2) // 2, (y1 + y2) // 2)

    if target_center:
        cx, cy = target_center
        offset_x = cx - frame_w // 2
        offset_y = cy - frame_h // 2

        if abs(offset_x) > 10:
            self.pan_angle += -1 if offset_x > 0 else 1
            self.pan_angle = max(0, min(180, self.pan_angle))

    asyncio.ensure_future(self.send_drive_command(f"pan:{self.pan_angle}"))

    if abs(offset_y) > 10:
        self.tilt_angle += 1 if offset_y > 0 else -1
        self.tilt_angle = max(0, min(180, self.tilt_angle))

    asyncio.ensure_future(self.send_drive_command(f"tilt:{self.tilt_angle}"))

    return frame

def update_frame(self):
    if self.latest_frame and not self.latest_frame.isNull():

```

```

        pix = QtGui.QPixmap.fromImage(self.latest_frame).scaled(
            self.video_label.size(),
            QtCore.Qt.KeepAspectRatio,
            QtCore.Qt.SmoothTransformation
        )
        self.video_label.setPixmap(pix)

    def resizeEvent(self, event):
        super().resizeEvent(event)
        if self.latest_frame is not None:
            QtCore.QTimer.singleShot(30, self.update_frame)

    def closeEvent(self, event):
        self.stream_active = False
        if self.video_task:
            self.video_task.cancel()
        if self.ws:
            asyncio.ensure_future(self.ws.close())
        event.accept()

    def keyPressEvent(self, event):
        if event.isAutoRepeat():
            return
        key = event.key()
        if key in KEY_COMMANDS and key not in self.pressed_keys:
            self.pressed_keys.add(key)

asyncio.ensure_future(self.send_drive_command(KEY_COMMANDS[key]))

    def keyReleaseEvent(self, event):
        if event.isAutoRepeat():
            return
        key = event.key()
        if key in self.pressed_keys:
            self.pressed_keys.remove(key)
            if not self.pressed_keys:
                asyncio.ensure_future(self.send_drive_command("halt"))

    async def send_drive_command(self, key):
        if not self.ws or self.ws.closed:
            return
        try:
            await self.ws.send(key)
            self.append_log(f"Command sent: {key}")
        except Exception as e:
            self.append_log(f"Send error: {e}")

def main():
    app = QApplication(sys.argv)
    loop = QEventLoop(app)
    asyncio.set_event_loop(loop)

    window = VideoControl()
    window.show()

    with loop:
        loop.run_forever()

if __name__ == "__main__":
    main()

```

6. Перевірити правильність роботи системи. Для цього підключити всі компоненти до живлення, запустити GUI та перевірити результат встановлення з'єднання з ESP32-CAM. Використовуючи створений графічний інтерфейс, увімкнути трансляцію відео з ESP32-CAM та відслідкувати результат розпізнавання об'єктів та автоматичного наведення камери на найбільший за площею об'єкт класу «смартфон».

Завдання для самостійного виконання

Модифікувати програму таким чином, щоб робот автоматично слідував (їхав) за найближчою в кадрі людиною.

Контрольні запитання

1. Як у методі *process_yolo()* реалізовано вибір об'єкта заданого класу для відслідковування? Поясніть, як обчислюється центр рамки та його відхилення від середини кадру.
2. Поясніть, яким чином робот розуміє, що камеру потрібно повернути у певну сторону.
3. Яким чином у коді Python-програми забезпечується поступове коригування кутів?
4. Як у системі забезпечується інтеграція результатів роботи YOLOv8 із механізмом Pan-Tilt? Поясніть послідовність передачі команд від обробки кадру до виконання руху серводвигунів.
5. Опишіть, як у системі забезпечується стабільність наведення камери на об'єкт при швидкому русі об'єкта в кадрі. Які обмеження механізму Pan-Tilt можуть впливати на це?
6. Як можна модифікувати алгоритм наведення камери, щоб враховувати не лише площу рамки об'єкта, а й його відстань від центру кадру?
7. Поясніть, як у коді забезпечується уникнення помилкових детектувань об'єктів. Які параметри YOLOv8 можна налаштувати для цього?