УДК 004.93'1:629.735 DOI: 10.18372/2073-4751.86.21278

Лукаш Ю.В.,
orcid.org/0009-0003-1824-8936
e-mail: 8391003@stud.nau.edu.ua

ІНТЕРФЕЙСНА АРХІТЕКТУРА МОДУЛЬНОЇ ПЛАТФОРМИ ВІДЕОАНАЛІЗУ ДЛЯ АВТОНОМНОГО КЕРУВАННЯ БПЛА

Національний університет «Київський авіаційний інститут»

Вступ

Безпілотні літальні апарати все частіше виступають не просто як платформи для польоту, а як повноцінні дослідницькі інструменти – з бортовою обробкою відео, прийняттям рішень у реальному часі та автономним виконанням завдань. Паралельно зростає різноманітність апаратних конфігурацій: різні типи камер, різні одноплатні комп'ютери, різні польотні контролери. Саме це розмаїття і породжує головну проблему дослідницької роботи.

Діапазон задач, де потрібна автономність на основі відеоаналізу, суттєво розширився. Окрім традиційних цивільних застосувань – моніторингу інфраструктури, пошуково-рятувальних операцій, сільське господарство – зросла потреба у платформах, здатних виконувати автономні місії в умовах обмеженого або відсутнього зв'язку. Це висуває додаткові вимоги до гнучкості програмної архітектури: алгоритми мають легко замінюватись і адаптуватись під конкретну задачу без перепроєктування системи.

На практиці виглядає це так: дослідник пише алгоритм трекінгу, але через місяць потрібно замінити USB-камеру на CSI – і доводиться переписувати частину коду. Або інший учасник команди хоче протестувати власний алгоритм детекції, але змушений спочатку розібратись у тому, як влаштована вся система захоплення відео і передачі команд на контролер. Все це витрати часу, які не мають відношення до самого дослідження.

Задача, яку вирішує ця робота - побудувати таку програмну архітектуру, де заміна відеовходу, алгоритму обробки або польотного контролера не торкається решти системи. Кожен компонент має чіткий контракт взаємодії, і цього достатньо.

Аналіз останніх досліджень і публікацій

Тема автономних БпЛА з комп'ютерним зором активно розвивається останні кілька років. Роботи [1, 2] показують, що оптична навігація і позиціонування на основі відеосенсорів стають критичними для широкого класу задач – від навігації, автоматичного розпізнавання об'єктів до супроводу цілей. При цьому автори [3, 4] звертають увагу на те, що реальні умови польоту можуть мати різноманітні обмеження – канал зв'язку, відсутність GPS, жорсткі вимоги до енергоспоживання – суттєво ускладнюють впровадження навіть добре відпрацьованих лабораторних алгоритмів.

Питання модульної побудови БпЛА-платформ розглядається в [5, 6]: там наголошується на тому, що незалежність компонентів – від захоплення кадру до генерації команди – є необхідною умовою для продуктивної дослідницької роботи. Стандартом комунікації та взаємодії між комп'ютером-компаньоном та польотним контролером БпЛА є протокол MAVLink. Більше деталей про протокол та його версії наведено у [7]. Також популярною темою є реалізації систем комп'ютерного зору на

малопотужних платформах, одноплатних комп'ютерах, таких як Raspberry Pi [8].

Водночас у більшості публікацій фокус зроблено на результатах роботи алгоритмів, а не на тому, як саме ці алгоритми підключені до системи. Програмні інтерфейси між відеоджерелами, аналізаторами кадрів і контуром керування або не описуються взагалі, або описуються лише в загальних рисах. Ця стаття заповнює саме цю прогалину.

Загальну апаратно-програмну архітектуру платформи описано у попередній роботі авторів [9]. Тут же мова йде виключно про програмні інтерфейси та взаємодію класів – тобто про те, як влаштовані точки розширення системи.

Постановка завдання

Метою даної роботи є розробка, опис та обґрунтування інтерфейсів архітектури програмної платформи відеоаналізу для автономних БпЛА, яка забезпечує підключення різних відеоджерел без змін в аналітичному коді, інтеграцію довільних алгоритмів комп'ютерного зору через єдиний контракт та підтримку різних польотних контролерів через параметризоване з'єднання через протокол MAVLink.

Для цього потрібно вирішити три конкретні задачі:

- формалізувати контракт CameraSource для відеоджерел;
- описати контракт FrameProcessor для алгоритмів обробки кадрів разом з усіма методами взаємодії;
- показати як FlightController приховує апаратні відмінності польотних контролерів за єдиним інтерфейсом.

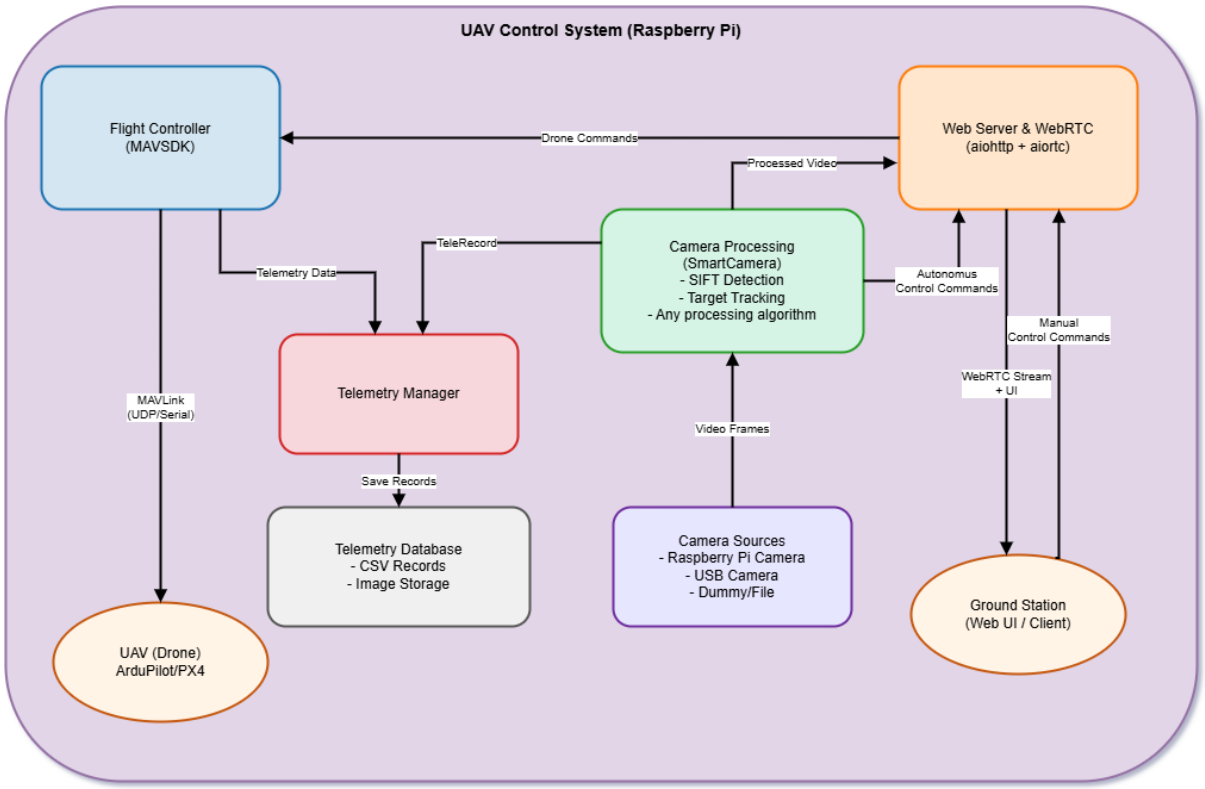

Рис. 1. Загальна архітектура платформи автономного керування БпЛА

Загальна архітектура

Загальну структуру платформи наведено на рис. 1. Система розгортається на бортовому комп'ютері БпЛА і забезпечує повний цикл – від отримання відеосигналу з камери до формування керуючих команд для польотного контролера. Між цими двома точками знаходиться кілька незалежних шарів обробки, кожен з яких має чітко

визначену відповідальність і не знає деталей реалізації сусіднього.

Детальніший огляд на взаємодію між шарами показано на рис. 2. Платформа складається з чотирьох основних компонентів: CameraSource відповідає за захоплення відео, FrameProcessor – за аналіз кадрів, FlightController – за взаємодію з польотним контролером, TelemetryManager – за збір і синхронізацію телеметрії. Компоненти взаємодіють між собою через чітко визначені інтерфейси, не знаючи деталей реалізації один одного.

Центральний елемент – цикл захоплення відео. Кожен новий кадр потрапляє до спільного буфера і одночасно передається активному аналізатору. Якщо аналізатор реалізував метод getPreparedVector(), результат його роботи у вигляді вектора швидкості передається до FlightController і виконується як команда руху. TelemetryManager при цьому працює паралельно – асинхронно зчитує дані з контролера і прив'язує їх до кадрів за єдиним часовим штампом.

Така архітектура свідомо будується на слабкій зв'язності: жоден компонент не залежить від конкретної реалізації сусіднього. Це і є основою для тієї варіативності, яка описується далі.

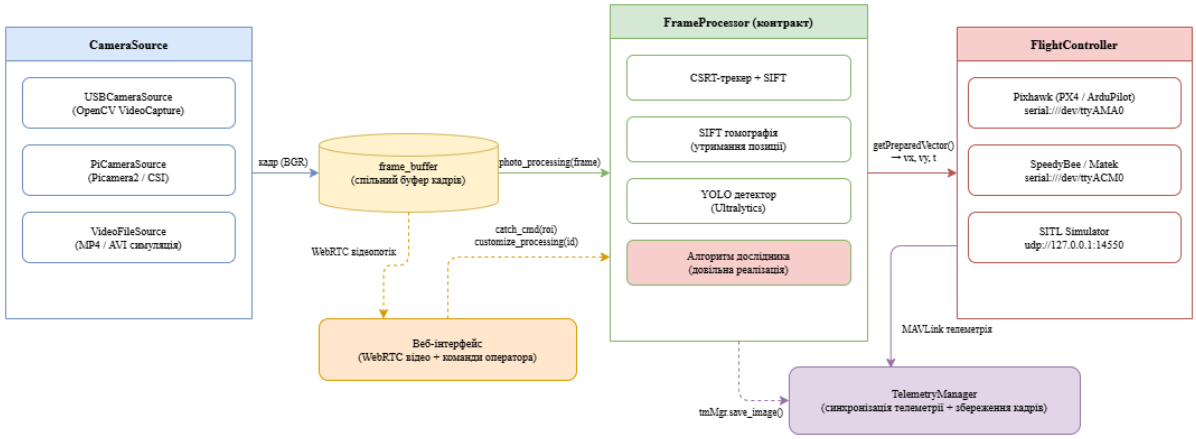

Рис. 2. Потік даних між компонентами платформи відеоаналізу БпЛА

Контракт підключення відеоджерел (CameraSource)

Перше питання, яке постає при розробці бортової системи відеоаналізу – як організувати захоплення відео так, щоб алгоритм обробки не залежав від типу камери. Вирішення, реалізоване у платформі, досить просте: кожне відеоджерело є окремим класом, що запускає фоновий потік захоплення і складає отримані кадри у спільний буфер. Аналізатор кадрів завжди отримує кадр з цього буфера - і йому байдуже, звідки він прийшов.

Реалізовано три джерела. USBCameraSource використовує OpenCV VideoCapture – стандартний спосіб роботи з USB-камерами, перевірений на кількох моделях з різними характеристиками. PiCameraSource працює через бібліотеку Picamera2 і розрахований на CSI-камери Raspberry Pi, включно з інфрачервоними модулями - вони особливо корисні для нічних або слабко освітлених сценаріїв. VideoFileSource зчитує кадри з MP4 або AVI файлу і при досягненні кінця починає відтворення спочатку. Останнє особливо цінне на етапі розробки: можна записати реальний політ і відпрацьовувати алгоритм без повторних запусків дрону.

Структурно всі три реалізації виглядають однаково – фоновий потік, буфер, виклик аналізатора:

```
def start_camera_capture(alg):
    cap = cv2.VideoCapture(0)
```

```
while True:
  ret, frame = cap.read()
  if ret:
    frame_buffer[0] = frame
    alg.photo_processing(frame)
```

Щоб перейти з USB на CSI або на відеофайл, достатньо змінити один рядок у конфігурації серверного модуля. Жодного іншого коду чіпати не потрібно.

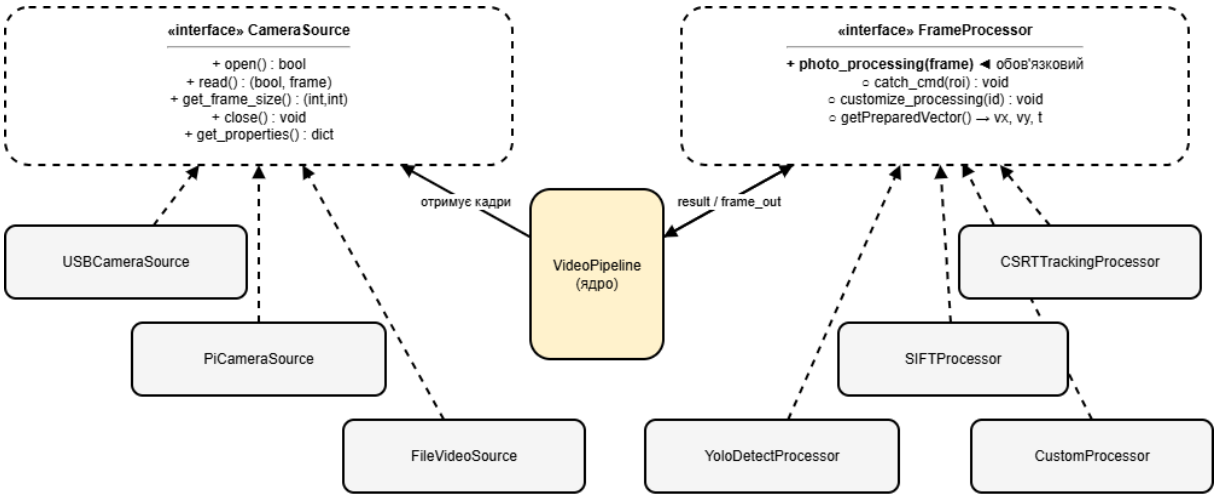

Рис. 3. Контракти інтерфейсної архітектури: CameraSource та FrameProcessor

Контракт інтеграції алгоритмів обробки кадрів (FrameProcessor)

Другий рівень модульності – це те, як підключаються самі алгоритми комп'ютерного зору. Тут свідомо обраний підхід duck typing замість формальної ієрархії класів: дослідник просто реалізує клас з потрібними методами і передає його екземпляр у систему. Ніяких базових класів успадковувати не потрібно – це знижує поріг входження і не нав'язує зайвих залежностей. І це є поширеним підходом в Python розробці.

Контракт складається з одного обов'язкового і трьох опціональних методів, зображених на рис. 3. Обов'язковий – photo_processing(frame) – отримує поточний BGR-кадр у вигляді NumPy-масиву. Що з ним робити далі – повністю на розсуд дослідника: виявляти об'єкти, рахувати оптичний потік, класифікувати сцену. Єдина вимога – метод має бути присутній.

Три опціональних методи розширюють можливості взаємодії з платформою. catch_cmd(roi) спрацьовує коли оператор клікає мишею на відеопотік у веб-інтерфейсі – система передає координати і розмір виділеної прямокутної області у форматі (x, y, w, h). Це зручно для ініціалізації трекера або вибору зони інтересу прямо під час польоту. customize_processing(id) викликається при натисканні однієї з чотирьох кнопок на панелі керування; значення id від 1 до 4 дозволяє, наприклад, перемикати режими роботи алгоритму або запускати певну фазу обробки без перезапуску системи.

Найважливіший з опціональних – getPreparedVector(). Саме він замикає контур автономного керування: метод повертає vx, vy (складові вектора швидкості у тілесній системі координат, в межах [-1.0; 1.0] м/с) і t – час виконання команди в секундах. Рекомендоване значення t – не більше 2 с, щоб система могла своєчасно реагувати на зміни. Якщо метод не реалізований, платформа просто не генерує автономних команд – апарат управляється вручну.

Мінімальний шаблон для нового алгоритму виглядає так:

```
class MyAnalyzer:
  def __init__(self):
    self.tmMgr = None   # set by platform
  # required
  def photo_processing(self, frame):
    pass
  # optional
  def catch_cmd(self, roi):
    pass
  # optional
  def customize_procesing(self, id):
    pass
  # optional
  def getPreparedVector(self):
    return vx, vy, t
```

Атрибут tmMgr – посилання на TelemetryManager – платформа встановлює сама після ініціалізації. Через нього аналізатор може зберігати оброблені кадри і накладати телеметрію одним викликом, без будь-якого додаткового коду.

Реалізовані алгоритми обробки кадрів

У межах платформи апробовано чотири реалізації контракту FrameProcessor, що охоплюють різні підходи до відеоаналізу. Шаблон дослідника – мінімальна реалізація з одним методом, яка обчислює середню яскравість кадру; використовується як точка відліку і приклад для нових учасників. Алгоритм трекінгу CSRT реалізує супровід об'єкта з автоматичним обчисленням вектора відхилення цілі від центру кадру; підтримує також пошук цілі за ключовими точками SIFT для автоматичної ініціалізації трекера. Алгоритм утримання позиції на основі SIFT алгоритму порівнює ключові точки поточного кадру з еталонним і генерує вектор корекції – апарат повертається до вихідної візуальної позиції без GPS. Алгоритм детекції об'єктів на базі YOLO використовує попередньо навчену модель через бібліотеку Ultralytics для розпізнавання об'єктів у реальному часі.

Усі чотири алгоритми підключалися до платформи однаково – одним рядком зміни у конфігурації. Жоден інший модуль при цьому не змінювався. Порівняння реалізацій за підтримуваними методами контракту наведено у табл. 1.

Таблиця 1. – Порівняння реалізацій контракту FrameProcessor

Реалізація	**Алгоритм**	**photo_ processing**	**catch_cmd**	**getPrepared Vector**	**FPS RPi 4B**
Шаблон дослідника	Яскравість кадру	+	+	+	30
CSRT-трекер + SIFT	Трекінг об'єкта	+	+	+	18
SIFT позиціонування	Утримання позиції	+	+	+	2
YOLO детектор	Детекція об'єктів	+	–	–	3

Тестування алгоритмів на Raspberry Pi 4B з відеопотоком 640×480 виявило суттєві відмінності у продуктивності (табл. 1). Шаблон дослідника і CSRT-трекер показали прийнятну швидкість - 30 і 18 FPS відповідно - достатню для безперервної обробки відеопотоку в реальному часі. Натомість позиціонування на основі SIFT і YOLO-детектор на тій самій платформі видають лише 2 і 3 FPS - це унеможливлює їх використання для покадрової обробки. Тому їх запуск реалізовано по команді оператора або за подією: алгоритм викликається не для кожного кадру, а лише коли це дійсно потрібно. Це підтверджує доцільність модульного підходу: дослідник сам

визначає логіку виклику свого алгоритму всередині photo_processing(), не будучи обмеженим частотою відеопотоку.

Варіативність підключення польотних контролерів

Ще один вимір модульності – підключення до польотного контролера. FlightController взаємодіє з апаратом через бібліотеку MAVSDK, що реалізує протокол MAVLink. Вибір конкретного контролера або режиму з'єднання визначається єдиним рядком – system_address – який змінюється у конфігурації без правок у коді:

```
# UART via GPIO — Pixhawk, SpeedyBee, Matek
fcIP = "serial:///dev/ttyAMA0:57600"
# USB-кабель
fcIP = "serial:///dev/ttyACM0:57600"
# SITL симулятор ArduPilot/PX4
fcIP = "udp://127.0.0.1:14550"
# TCP через мережу
fcIP = "tcp://192.168.1.100:5762"
```

На практиці це означає наступне: алгоритм розробляється і налагоджується на SITL-симуляторі, потім той самий код запускається з реальним Pixhawk або SpeedyBee – змінюється лише рядок з'єднання. Така можливість виявилась особливо корисною: кілька алгоритмів спочатку були повністю відпрацьовані в симуляції, і лише після цього перенесені на реальний апарат.

FlightController надає набір асинхронних команд: arm/disarm, takeoff/land, set_fly_vector для руху в заданому напрямку через Offboard mode, do_orbit та return_to_home. Всі команди побудовані на async/await і інтегруються у єдиний event loop разом з циклом обробки відео – без блокувань і гонок потоків.

Підсистема телеметрії та синхронізації даних

TelemetryManager вирішує задачу, яка на перший погляд здається другорядною, але без неї аналіз результатів польоту перетворюється на головний біль: синхронізація відеокадрів з телеметричними даними. Підсистема паралельно зчитує з контролера GPS-координати, кути орієнтації (крен, тангаж, курс), стан акумулятора і NED-вектори швидкості та положення.

Кожен запис зберігається у CSV-файлі з міткою часу в наносекундах. Відповідні кадри – оброблені аналізатором – зберігаються поруч, з тим самим ідентифікатором. Після польоту дослідник отримує повністю синхронізований набір даних: для кожного кадру відомо де і як орієнтувався апарат у момент його захоплення. Це дозволяє не тільки відтворити траєкторію, але й кількісно оцінити роботу алгоритму – наприклад, наскільки точно CSRT-трекер утримував ціль при різних кутах крену.

Висновки

У роботі описано інтерфейсну архітектуру модульної платформи відеоаналізу для автономного керування БпЛА. Ключова ідея – три незалежних рівні варіативності: відеоджерела, алгоритми обробки кадрів і польотні контролери. Кожен рівень має чіткий програмний контракт, і заміна будь-якого компонента не впливає на решту системи.

Контракт CameraSource реалізовано для USB-камер, CSI-камер Raspberry Pi і відеофайлів. Контракт FrameProcessor апробовано на чотирьох алгоритмах – CSRT-трекінгу, SIFT-позиціонування, YOLO-детекції та мінімальному шаблоні. Всі чотири підключалися без змін в інших компонентах платформи. Підключення польотних контролерів параметризоване рядком MAVLink-з'єднання і підтримує Pixhawk, SpeedyBee, Matek та SITL-симулятор.

Практичний досвід роботи з платформою показав: дослідник, який хоче протестувати власний алгоритм, реалізує один клас з одним обов'язковим

методом. Цього достатньо щоб алгоритм отримував кадри в реальному часі, мав доступ до телеметрії і міг керувати польотом. Все інше вже є у платформі.

Серед напрямків подальшої роботи – формалізація контрактів через абстрактні базові класи Python (ABC), розширення підтримуваних відеоджерел (RTSP, стереокамери), а також систематичне вимірювання затримок у контурі відеоаналіз–керування для різних алгоритмів і апаратних конфігурацій.

Література

1. *Prystavka P., Cholyshkina O.* Estimation of the aircraft's position based on optical channel data. *CEUR Workshop Proceedings*. 2024. Vol. 3925. P. 93-105.

2. *Ruzhentsev N. et al*. Radio-heat contrasts of UAVs and their weather variability at 12 GHz, 20 GHz, 34 GHz, and 94 GHz frequencies. *ECTI Transactions on Electrical Engineering, Electronics, and Communications*. 2022. Vol. 20. P. 163-173. DOI: 10.37936/ecti-eec.2022202.246878

3. *Liu S., Li X., Lu H., He Y*. Multi-object tracking meets moving UAV. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022. P. 8866-8875. DOI: 10.1109/CVPR52688.2022.00867

4. *Mohsan S.A.H. et al.* Unmanned aerial vehicles (UAVs): practical aspects, applications, open challenges, security issues, and future trends. *Intelligent Service Robotics*. 2023. Vol. 16. P. 109-137. DOI: 10.1007/s11370-022-00452-4

5. *Hong T. et al.* A real-time tracking algorithm for multi-target UAV based on deep learning. *Remote Sensing*. 2023. Vol. 15, No. 1. DOI: 10.3390/rs15010002

6. *Arafat M.Y., Alam M.M., Moh S.* Vision-based navigation techniques for unmanned aerial vehicles: Review and challenges. *Drones*. 2023. Vol. 7. DOI: 10.3390/drones7020089

7. *Koubaa A. et al*. Micro Air Vehicle Link (MAVLink) in a nutshell: A survey. *IEEE Access*. 2019. Vol. 7. P. 87658-87680. DOI: 10.1109/ACCESS.2019.2924410

8. *Ortega L.D., Olivares-Mendez M.A., Campoy P.* Low-cost computer-vision-based embedded systems for UAVs. *Robotics*. 2023. Vol. 12. DOI: 10.3390/robotics12060145

9. *Lukash Y., Prystavka P.* A research platform for vision-based UAV autonomy: Architecture and implementation. *CEUR Workshop Proceedings*. 2025. Vol. 4024. P. 250-259.

10. *Жуков І., Лукаш Ю.* Використання алгоритму трекінгу об’єкту по відеозображенню для реалізації автономної функції слідування за ціллю для БПЛА. *Проблеми інформатизації та управління*. 2024. Т. 2, № 78. С. 14-17. DOI: 10.18372/2073-4751.78.18956

Лукаш Ю.В.

ІНТЕРФЕЙСНА АРХІТЕКТУРА МОДУЛЬНОЇ ПЛАТФОРМИ ВІДЕОАНАЛІЗУ ДЛЯ АВТОНОМНОГО КЕРУВАННЯ БПЛА

Метою роботи є представлення інтерфейсної архітектури модульної програмної платформи відеоаналізу для систем автономного керування БпЛА з акцентом на розширюваність та взаємозамінність компонентів.

Платформа організована навколо трьох незалежних рівнів модульності. Перший – це контракт відеоджерел(CameraSource): незалежно від типу камери, кожне джерело запускає фоновий потік захоплення та записує кадри до спільного буферу. Доступні три реалізації: USBCameraSource через OpenCV VideoCapture, PiCameraSource з використанням бібліотеки Picamera2 для CSI-камер Raspberry Pi (включаючи інфрачервоні моделі) та VideoFileSource для офлайн-розробки із записаним відеоматеріалом. Перемикання між ними вимагає зміни одного рядка в конфігурації сервера. Другий рівень – це контракт обробки кадрів (FrameProcessor), побудований на структурній типізації замість формального наслідування. Дослідник реалізує один обов'язковий метод – photo_processing(frame) – який отримує поточний кадр BGR як масив NumPy. Три додаткові методи розширюють взаємодію: catch_cmd(roi) викликається, коли оператор клікає на відеопотік,

щоб вибрати область інтересу; customize_processing(id) реагує на одну з чотирьох кнопок панелі керування, що дозволяє параметризувати алгоритм під час польоту; getPreparedVector() замикає автономний цикл керування, повертаючи значення складових швидкості vx, vy та час виконання команди t. Третій рівень охоплює підключення контролера польоту: клас FlightController огортає MAVSDK та приймає рядок підключення MAVLink як єдиний параметр конфігурації, що дозволяє прозоре перемикання між апаратним забезпеченням Pixhawk, SpeedyBee, Matek та програмним симулятором SITL. Чотири реалізації FrameProcessor – трекер CSRT з ініціалізацією SIFT, утримання позиції на основі гомографії SIFT, виявлення об'єктів YOLO та мінімальний шаблон – були інтегровані та перевірені на платформі, кожна з яких реалізована як окремий клас без внесення змін в інші компоненти системи. Запропонована архітектура знижує бар'єр для інтеграції нових алгоритмів обробки відео до реалізації інтерфейсу мінімального класу. Дослідник, який працює з платформою, повинен зосередитися лише на самому алгоритмі – робота з камерою, реєстрація телеметрії та виконання команд польоту вжереалізовані. Це робить платформу практичною та відтворюваною базою для досліджень автономності БПЛА на основі комп'ютерного зору.

Ключові слова: *БпЛА, комп'ютерний зір, модульна архітектура, відеоаналіз, MAVLink, MAVSDK, програмний інтерфейс.*

Lukash Y.V.

INTERFACE ARCHITECTURE OF A MODULAR VIDEO ANALYSIS PLATFORM FOR AUTONOMOUS UAV CONTROL

The purpose of this work is to present the interface architecture of a modular software platform for video analysis in autonomous UAV control systems, with emphasis on extensibility and component replaceability.

The platform is organized around three independent modularity levels. The first is the video source contract (CameraSource): regardless of the camera type, each source runs a background capture thread and writes frames to a shared buffer. Three implementations are available – USBCameraSource via OpenCV VideoCapture, PiCameraSource using the Picamera2 library for Raspberry Pi CSI cameras (including infrared models), and VideoFileSource for offline development with recorded footage. Switching between them requires changing a single line in the server configuration. The second level is the frame processing contract (FrameProcessor), built on duck typing rather than formal inheritance. A researcher implements one mandatory method – photo_processing(frame) – which receives the current BGR frame as a NumPy array. Three optional methods extend the interaction: catch_cmd(roi) is called when the operator clicks on the video stream to select a region of interest; customize_processing(id) responds to one of four control panel buttons, enabling in-flight algorithm parameterization; getPreparedVector() closes the autonomous control loop by returning velocity components vx, vy and execution time t. The third level covers flight controller connectivity: the FlightController class wraps MAVSDK and accepts a MAVLink connection string as its only configuration parameter, enabling transparent switching between Pixhawk, SpeedyBee, Matek hardware and the SITL software simulator. Four FrameProcessor implementations – a CSRT tracker with SIFT initialization, SIFT homography-based position holding, YOLO object detection, and a minimal template – were integrated and validated on the platform, each implemented as a single class without touching any other component.

The proposed architecture lowers the barrier for integrating new video processing algorithms to implementing a minimal class interface. A researcher working with the platform needs to focus only on the algorithm itself – camera handling, telemetry logging, and flight command execution are already provided. This makes the platform a practical and reproducible baseline for vision-based UAV autonomy research.

Keywords: *UAV, computer vision, modular architecture, video analysis, MAVLink, MAVSDK, software interface, object tracking*

Стаття подана до редакції: 27/04/2026
Стаття прийнята до опублікування: 05/05/2026
Стаття опублікована: 30/05/2026