УДК 004.424, 004.94, 007.52 https://doi.org/10.15407/pp2025.01.110

Р.С. Шевченко, А.Ю. Дорошенко, В.О. Лесик, О.В. Савчук, О.А. Яценко

ІНТЕРФЕЙСНО-ОРІЄНТОВАНИЙ ПІДХІД ДО ЗАСОБІВ МОДЕЛЮВАННЯ МУЛЬТИАГЕНТНИХ СИСТЕМ

Високорівневі системи моделювання мультиагентних систем здатні суттєво прискорити процес розробки та впровадження програмного забезпечення для автономних мультиагентних місій. Оскільки різні задачі вимагають уваги до специфічних аспектів моделювання, інтерфейсно-орієнтований підхід може стати ефективним засобом адаптації різних моделей середовища до заданих інтерфейсів поведінки. Моделювання мультиагентних систем охоплює широкий спектр процесів – від фізичного руху агентів до формування стратегій поведінки та організації їхньої взаємодії. Інтерфейсно-орієнтований підхід дає змогу створювати гнучкі мультиагентні системи моделювання, які можуть виконувати широкий спектр завдань, забезпечуючи швидку розробку моделей поведінки в мультиагентному середовищі та SITL-тестування низькорівневого коду таких складних систем як автопілоти. В рамках підходу проєктування системи розпочинається з визначення інтерфейсної взаємодії між її компонентами, що дає можливість повторного використання коду та створення індивідуальної реалізації компонентів для специфічних експериментальних завдань. Ведуться роботи над прототипом системи мультиагентного моделювання Blefusku, однією з особливостей якої є інтерфейсно-орієнтований підхід і підтримка високорівневих моделей опису поведінки. Основні модулі системи – це середовище моделювання, яке містить модель середовища та формує сигнали сенсорів, і контейнер агентів, що відповідає за поведінку об’єктів і змістовну частину комунікаційного середовища. Інтерфейс описаний як сервіс gRPC, що дозволяє з’єднувати різні компоненти, написані у різних середовищах програмування. Комунікаційний рівень ґрунтується на протоколі MAVLink. Поведінка визначається об’єктом, у якому запрограмована реакція на дані з сенсорів та повідомлення і періодичні активності. Як приклад наведено фрагмент сценарію пошуково-рятувальної місії за участі дронів.

Ключові слова: агенти, інтерфейсно-орієнтований підхід, моделювання, мультиагентна система, робототехніка, сенсори, MAVLink, SITL, ROS.

UDC 004.424, 004.94, 007.52

R.S. Shevchenko, A.Yu. Doroshenko, V.O. Lesyk, O.V. Savchuk, O.A. Yatsenko

INTERFACE-ORIENTED APPROACH TO MODELING TOOLS FOR MULTI-AGENT SYSTEMS

High-level modeling systems for multi-agent systems can significantly accelerate the process of developing and implementing software for autonomous multi-agent missions. Since different tasks require attention to specific aspects of modeling, an interface-oriented approach can be an effective means of adapting different models of the environment to given behavioral interfaces. Modeling of multi-agent systems covers a wide range of processes – from the physical movement of agents to the formation of behavioral strategies and the organization of their interaction. The interface-oriented approach makes it possible to create flexible multi-agent modeling systems that can perform a wide range of tasks, ensuring the rapid development of behavioral models in a multi-agent environment and SITL testing of low-level autopilot code. As part of the approach, the design of the system begins with the definition of interface interaction between its components, which makes it possible to reuse the code and create individual implementations of components for specific experimental tasks. Work is underway on a prototype of the Blefusku multi-agent modeling system, one of the features of which is an interface-oriented approach and support for high-level behavioral description models. The main modules of the system are a modeling environment that contains a model of the environment and generates sensor signals, and an agent container that is responsible for the behavior of objects and the content of the communication environment. The interface is described as a gRPC service that allows connecting different components written in different programming environments. The communication layer is based on the MAVLink protocol. Behavior is determined by an object that is programmed to respond to sensor data and messages and periodic activities. A fragment of a search and rescue mission scenario with drones is given as an example.

Keywords: agents, interface-oriented approach, modeling, multi-agent system, robotics, sensors, MAVLink, SITL, ROS.

Вступ

В Інституті програмних систем НАН України тривалий час проводяться дослідження, повязані із розробкою та застосуванням систем автоматизації програмування та моделювання [1–4].

Моделювання мультиагентних систем є актуальною темою, що охоплює широкий спектр процесів – від фізики переміщення агентів до вироблення високорівневої стратегії поведінки та форми взаємодії. Кожна ситуація моделювання в чомусь унікальна і потребує від програмних засобів моделювання інфраструктури різних властивостей. Водночас, велика кількість елементів інфраструктури не змінюється. Чи можна побудувати архітектуру, що з одного боку дозволяє перевикористання компонентів, а з іншого боку достатньо гнучку для використання у широкому спектрі завдань?

Історично більшість симуляційних моделей були сконцентровані на відтворенні фізичного середовища і передачі значення сенсорів до програмного забезпечення керування (автопілоту) та передачі значення актуаторів назад у систему симуляції. Така організація взаємодії отримала назву SITL (Software in the Loop) і стала однією з найважливіших технологій тестування. Із розвитком стандартизації робототехнічних систем деяку популярність здобув також стандарт ROS2 [5–7] (Robot Operating System), що включає в себе стандартизовані формати повідомлень для даних сенсорів. Тому типова схема симуляції виглядає як на Рис. 1.

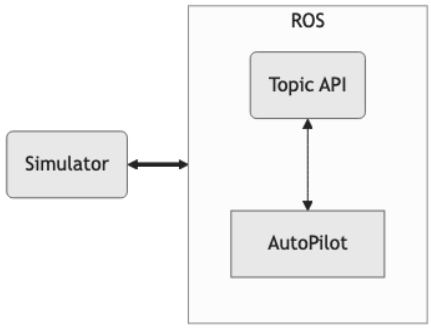

Рис. 1. Типова схема моделювання мультиагентних систем

Як правило, ROS з автопілотом знаходиться в doker-контейнері, запущеному на машині з симуляцією.

Найбільш відомим представником такого типу є Gazebo [8], що де-факто є стандартом у цій сфері, дистрибутиви ROS мають у своєму складі пакети інтеграції з цією системою. Крім того, оскільки для опису динаміки роботів однією з найбільш розповсюджених систем моделювання є Matlab, то MathWorks випустило Matlab ROS Toolbox [9], що дозволяє інтегрувати Matlab моделі з ROS системами.

Подальший розвиток цього напрямку пов'язаний з необхідністю використання більш багатого середовища для поведінки моделей (насамперед пов'язане з розробкою моделей комп'ютерного зору та машинного навчання). В програмуванні є галузь, де однією з головних характеристик розробок є візуалізація віртуальних світів, що містять багато об'єктів, з якими можуть взаємодіяти актори. Це програмування комп'ютерних ігор.

Більше того, в розробці комп'ютерних ігор є такий клас програмного забезпечення, як платформи (або рушії), де однією із задач є моделювання ігрової фізики в режимі м'якого реального часу. Для багатьох задач робототехніки, фізично точне моделювання динаміки може бути замінене приблизною версією, що може виконуватись на ігровій платформі. Знаковим є проєкт AirSim [10] від Microsoft, що використовує для візуалізації Unreal Engine і також підтримував експериментальний плагін для іншої ігрової платформи – Unity. Siemens опублікувала власну бібліотеку інтеграції ROS з Unity [11] та використовувала її у низці індустріальних проєктів. Unity Technology створила Unity Robotic Hub [12] на основі коду Siemens, але згодом припинила її підтримку. Також Unity створила ML-Agent Toolkit [13], що як середовище для тренування нейронних мереж розвивається і зараз.

Якщо переходити до мультиагентних систем, то першою перешкодою, з якою стикаються розробники при еволюції

в цьому напрямку, є швидкодія – паралельне застосування симуляцій не було спроєктоване із самого початку в більшості бібліотек симуляції фізики. LG створили на основі Unity свою версію платформи симуляції – cloisim [14], що використовує той же формат опису світу SDF, що і Gazebo. У Flightmare [15] автори розділили рендеринг та симуляцію і створили API для паралельного генерування даних для сенсорної камери.

Наступна перешкода – складність підтримки. Якщо в нас мультиагентна система, і кожний агент представлений докер-контейнером з автопілотом, як показано на Рис. 2, то конфігурація та підтримка цієї структури стає трудомісткою.

Ще одна річ, про яку слід подбати – це комунікаційний рівень, що відсутній у випадку одного робота. Агенти мають передавати сигнали за допомогою узгоджених протоколів.

В AeroStack [16] пропонується стандартизована ROS-орієнтована архітектура, що включає в себе як SITL симуляцію, так і комунікаційний рівень.

Також уваги потребує опис поведінки агентів, але, якщо система симуляції ґрунтується на моделі SITL, то тоді опис поведінки – це модифікація автопілотів агентів. Однак, автопілоти, як правило, написані низькорівневими мовами програмування, їх модифікувати та дописувати дуже трудомістко. Було б добре мати можливість описувати поведінку за допомогою більш високорівневих моделей.

Мультиагентні системи моделювання, що сфокусовані на абстрактному описі поведінки та не прив'язані до точного моделювання нижнього рівня, представляють собою окремий напрямок. З найбільш відомих систем можна вказати JADE [17, 18] (Java Agent Based Environment), де алгоритм роботи агента представляється Java-класом, що може виконувати запити до середовища та взаємодіяти з іншими такими ж об'єктами і обмінюватись інформацією між ними, використовуючи FIPA-сумісні протоколи. Ще один підхід – це представлення поведінки не як програми, а як функції вибору поведінки з певного набору елементарних елементів поведінки у часі. Такий підхід цікавий тим, що до розробки поведінки, представленої у такому вигляді, можна застосувати методи машинного навчання, такі як навчання з підкріпленням [19]. В CAMAS [20] недетерміністична поведінка сукупності роботів формально описується за допомогою марковських ланцюгів і моделюється за допомогою швидкодіючого симулятора дискретних подій.

1. Інтерфейсно-орієнтований підхід

Чи можна побудувати мультиагентну систему моделювання, призначену для виконання широкого спектру задач, і з одного боку мати можливість швидкої розробки моделей поведінки у мультиагентному середовищі, а з іншого – мати можливість SITL-тестування низькорівневого коду автопілота? Це дасть можливість швидкої розробки і валідації множини моделей поведінки для вибору однієї з них, яку можна буде реалізувати у низькорівне-

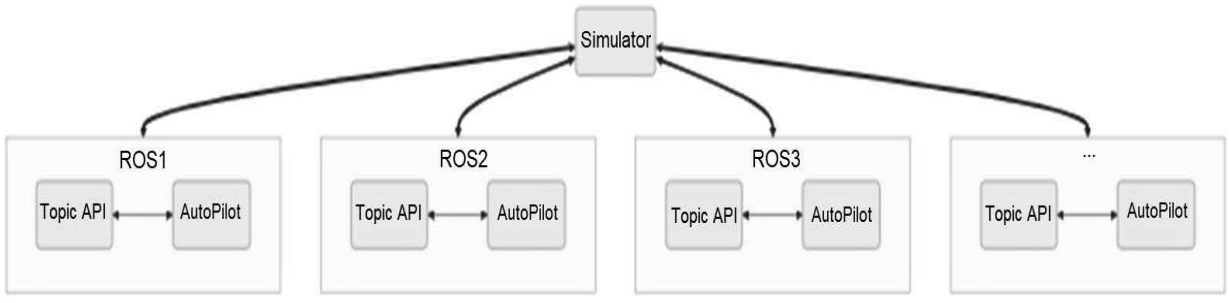

Рис. 2. Мультиагентна система, де кожний агент представлений докер-контейнером з автопілотом

вому автопілоті. Зараз ведуться роботи над прототипом системи мультиагентного моделювання, що має назву Blefusku. Однією з її особливостей є інтерфейсно-орієнтований підхід і підтримка високорівневих моделей опису поведінки.

Інтерфейсно-орієнтований підхід полягає в тому, що опис інтерфейсної взаємодії між різними частинами системи є першим артефактом, з якого починається проєктування системи. Маючи різні реалізації компонент для різних задач, ми можемо водночас і перевикористовувати код, і розробляти індивідуальний варіант функції під експеримент.

В найбільш загальному вигляді спрощену структуру системи можна зобразити, як показано на Рис. 3 – в нас є певне середовище, що включає в себе функціонал фізичних розрахунків, об'єктів та полів, у яких відбувається комунікація.

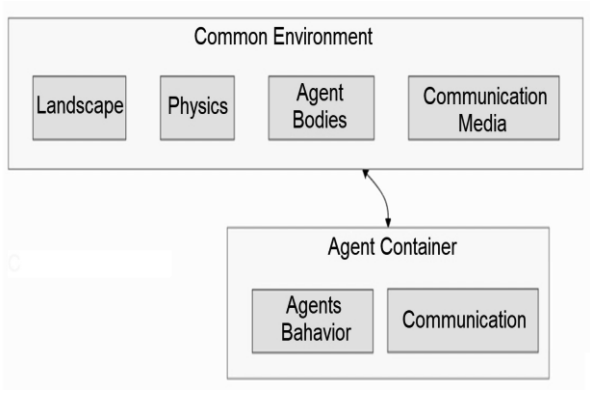

Рис. 3. Основні компоненти симуляції

З іншого боку, є контейнер агентів, що відповідає за поведінку об'єктів та змістовну частину комунікаційного середовища. Основний інтерфейс – це взаємодія між середовищем та контейнером агентів. Інтерфейс описаний як сервіс gRPC, що дозволяє з'єднувати різні компоненти, написані у різних середовищах програмування. Наприклад, для середовища є реалізації API мовами Rust, C# (для Unity), та планується підключення C++ інтерфейсу Unreal Engine.

Функціональність агента можна описати як функцію реакції, що отримує на вхід значення сенсорів та вхідні повідомлення і повертає значення актуаторів та вихідні повідомлення. На protobuf це можна представити як вхідні та вихідні повідомлення:

```
message ActorBehaviorInput {
  map<string, SensorData>
    sensor_data = 1;
  map<string, ActuatorReply>
    actuator_replies = 2;
  Messages input_messages = 3;
}

message ActorBehaviorOutput {
  map<string, ActuatorCommand>
    actuator_commands = 1;
  Messages output_messages = 2;
}
```

Структура SensorData – це дані сенсорів, описані максимально близько до відповідних структур в ROS2:

```
message SensorData {
  oneof data {
    Image image = 1;
    NavSatInfo navsat = 2;
    ImuInfo imu = 3;
    …
  }
}
```

Аналогічно структура ActuatorData – це типова структура вводу актуатора. Репліки з актуаторів надходять до агента у наступному циклі управління. Можна уявити роботу агента як функцію

$$f: ActorBehaviorInput \times State \rightarrow State \times ActorBehaviorOutput,$$

де *State* – внутрішній стан агента, структура якого описана у поведінці.

Комунікаційний рівень ґрунтується на протоколі MAVLink – вважається, що всі агенти об'єднані у спільну MAVLink-мережу. Поряд зі стандартними повідомленнями MAVLink також підтримуються і додаткові повідомлення, визначені користувачем – розроблено транслятор, який приймає на вхід XML-опис MAVLink повідомлення і додає його до protobuf струк-

тури, що об'єднує всі типи можливих повідомлень.

Контейнер агентів містить бібліотеку можливих поведінок. Поведінка визначається об'єктом, у якому запрограмована реакція на дані з сенсорів, та повідомлення і періодичні активності. Ще один компонент, SITL-проксі, може використовуватись для підключення реального автопілоту у режимі SITL-тестування, де ми можемо його запускати в одному середовищі з чисто програмними агентами. Структура такого підключення зображена на Рис. 4.

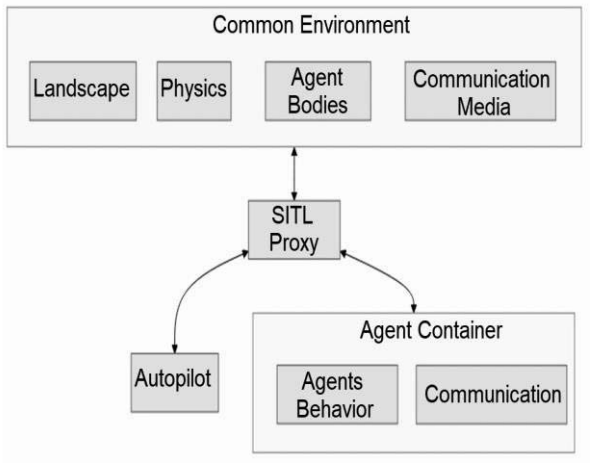

Рис. 4. Симуляція, сумісна з SITL

2. Програмний опис поведінки

Високорівневий опис поведінки задається за допомогою опису класу мовою Python. Екземпляр такого класу створюється для кожного актора і при ініціалізації отримує стан та контекст актора, де визначені методи взаємодії з сенсорами та актуаторами.

Програміст може описати методи реакції на події в об'єкті поведінки та пов'язати їх із повідомленнями за допомогою декораторів.

Як приклад наведемо фрагмент сценарію пошуково-рятувальної місії, у якій беруть участь N дронів. Нехай у нас є певна сфера відповідальності, за сигналом початку місії дрони розподіляють між собою поля пошуку і виконують обліт території за визначеним маршрутом. Водночас камери відслідковують об'єкт пошуку і, знаходячи схоже зображення, надсилають його для підтвердження на наземну станцію. Якщо підтвердження отримане, дрон спускається поряд з об'єктом пошуку та відкриває маніпулятор, а інші дрони завершують місію.

Почнемо з того, що додамо до стандартних повідомлень MAVLink нові повідомлення, що стосуються рятувальної місії. Це може бути п'ять повідомлень:

– RESQUE_MISSION_START – генерується на старті місії;

– RESQUE_TARGET_CANDIDATE (з полями, що включають координати та MFTP URL та унікальним ID) – генеруються, коли камера знайшла кандидата для цілі пошуку;

– RESQUE_TARGET_CANDIDATE_APPROVE – генеруються, коли з наземної станції прийшло підтвердження, що пошук успішний;

– RESQUE_TARGET_CANDIDATE_REJECT – генерується, коли оператор позначив результати пошуку як несправжнє розпізнавання;

– RESQUE_TARGET_REACHED – коли один дрон почав операцію порятунку, а інші можуть закінчити пошук.

Після цього потрібно перегенерувати класи повідомлень та імплементувати поведінку.

Поведінка визначається класом, у якому визначається стан і вказуються реакції на повідомлення (Рис. 5).

Тобто тут ми бачимо, що під час старту пошукової місії, якщо дрон не зайнятий, викликається метод start_search, що ініціює старт місії. Відповідне MAVLink-повідомлення передається на цей же автопілот (і буде відпрацьоване як частина стандартної поведінки).

У режимі пошуку періодично сканується зображення з камери, і якщо знайдений кандидат, то місія призупиняється і на наземну станцію передається зображення.

Отримавши підтвердження того, що пошук вдалий, дрон переходить у режим посадки біля цілі, та посилає броадкаст-сигнал, що пошукову ціль знайдено (Рис. 6).

Усі інші дрони бачать це повідомлення і призупиняють роботу, якщо немає іншої активної пошукової місії.

```
class ResqueMission(BaseBehavior):

   state: 'IDLE' | 'SEARCH' | 'APPROVE' | 'RESCUE' | 'RETURN' | 'LAND'

   @on_message(RESQUE_MISSION_START, when='IDLE')
   async def  start_search(self, message: mavlink_pb2.SEARCH_AREA):
       self.state = 'SEARCH'
       await self.ctx.send_mavlink(my_system_id, 1,
MAV_CMD_DO_SET_MISSION_CURRENT(1,1) )
       self._context.log_info("ResqueMission: start")

   @periodic(duraction = 1, when='SEARCH')
   async def classify_resque_target(self):
       image = await self.ctx.camera_image()
       found = check_for_person(image)
       if found:
           self.state = 'APPROVE'
           await self.ctx.send_mavlink(my_system_id, 1, MAV_CMD_CONDITION_DELAY, 1)
           image_uri = await ctx.upload_image(image)
           self.ctx.send_mavlink(GROUND_STATION, 1, RESQUE_CANDIDATE_FOUND,
...ctx.pos, image_uri)
```

Рис. 5. Клас, що задає поведінку

```
   @on_message(RESQUE_CANDIDATE_APPROVED, when='APPROVE')
   async def  resque_candidate_approved(self, message:
   mavlink_pb2.CAMERA_IMAGE_CAPTURED):
       self._context.log_info("ResqueMission: resque_approved")
       self.state = 'RESCUE'
       await self.ctx.send_mavlink(my_system_id, 1, SET_POSITION_TARGET_LOCAL_NED,
          ...correct_resque_target(ctx.pos))
       await self.ctx.send_mavlink(my_system_id, 1, MAV_CMD_DO_RALLY_LAND)
       self.ctx.send_mavlink(BROADCAST, 1, RESQUE_TARGET_REACHED, ctx.pos)
```

Рис. 6. Після підтвердження успішного пошуку дрон переходить у режим посадки біля цілі та надсилає широкомовний сигнал про її виявлення

Зазначимо, що один агент може описуватись кількома скриптами поведінки, які можуть описувати різні компоненти однієї системи. Так, наприклад, у розробці систем машинного зору, як правило, окремо моделюється камера.

Висновки

Високорівневі системи моделювання мультиагентних систем відіграють ключову роль у прискоренні циклу розробки та впровадження програмного забезпечення для автономних мультиагентних місій. Вони дозволяють створювати складні симуляційні середовища, які можуть точно відтворювати як фізичні умови, так і

поведінкові аспекти агентів, що взаємодіють між собою. Це, у свою чергу, сприяє ефективному тестуванню, налагодженню та вдосконаленню алгоритмів управління автономними системами без необхідності безпосереднього використання реального обладнання на ранніх етапах розробки.

Оскільки різні завдання вимагають акцента на різних аспектах моделювання, гнучкість симуляційних платформ стає критично важливою. Деякі сценарії потребують точного моделювання фізичних законів, таких як аеродинаміка або динаміка руху транспортних засобів, тоді як інші зосереджуються на комунікаційних аспектах або ухваленні рішень агентами в складному середовищі.

У цьому контексті інтерфейсно-орієнтований підхід до розробки мультиагентних систем може значно полегшити адаптацію різних середовищ моделювання до специфічних інтерфейсів поведінки агентів. Такий підхід передбачає чітке розділення між моделями середовища та алгоритмами поведінки, що дозволяє легко змінювати або вдосконалювати окремі компоненти системи без необхідності повного перероблення всієї інфраструктури. Крім того, це сприяє масштабованості та модульності симуляційних платформ, що робить їх придатними для широкого спектру застосувань – від дослідження стратегій кооперації роботів до тестування автономного транспорту та безпілотних літальних апаратів.

Завдяки такій організації можна не лише скоротити час розробки програмного забезпечення, а й покращити його якість, забезпечивши ефективне тестування віртуальних агентів у різних умовах ще до їхнього розгортання у реальному світі.

References

1. P. Andon, A. Doroshenko, V. Akylovsky, P. Ivanenko, O. Yatsenko, Formal and adaptive methods of constructing high-performance parallel programs. Kyiv: Naukova Dumka, 2023. [in Ukrainian]
2. A. Doroshenko, O. Yatsenko, Formal and adaptive methods for automation of parallel programs construction: emerging research and opportunities. Hershey: IGI Global, 2021. doi: 10.4018/978-1-5225-9384-3
3. Rahozin D. V., Doroshenko A. Yu. Modelling videocard memory performance for LLM neural networks, in: Problems in programming 2–3 (2024) 37–44. doi: 10.15407/pp2024.02-03.037 [in Ukrainian]
4. Doroshenko A. Yu., Okonsky I. V., Zhereb K. A., Beketov O. G. Using modeling tools to determine optimal parameters of program execution on video graphics accelerators, in: Problems in programming 2 (2013) 23–31. [in Ukrainian]
5. M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, A. Y. Ng, ROS: an open-source Robot Operating System, in: Open-Source Software workshop of the International Conference on Robotics and Automation (ICRA) (2009) 1–6. Accessed: 04.03.2025. http://www.robotics.stanford.edu/~ang/papers/icraoss09-ROS.pdf
6. S. Macenski, T. Foote, B. Gerkey, C. Lalancette, W. Woodall, Robot Operating System 2: design, architecture, and uses in the wild, in: Science Robotics (2022) 7. doi: 10.1126/scirobotics.abm6074
7. ROS. Accessed: 04.03.2025. https://ros.org/
8. N. Koenig, A. Howard, Design and use paradigms for Gazebo, an open-source multi-robot simulator, in: 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566) (2004), vol. 3, 2149–2154, doi: 10.1109/IROS.2004.1389727.
9. Matlab ROS toolbox. Accessed: 04.03.2025. https://www.mathworks.com/products/ros.html
10. S. Shah, D. Dey, C. Lovett, A. Kapoor, AirSim: high-fidelity visual and physical simulation for autonomous vehicles, in: ArXiv (2017) doi: 10.48550/arXiv.1705.05065
11. ros-sharp. Accessed: 04.03.2025. https://github.com/siemens/ros-sharp/
12. Unity-Robotics-Hub. Accessed: 04.03.2025. https://github.com/Unity-Technologies/Unity-Robotics-Hub/
13. J. Arthur, V.–P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, D. Lange, in: arXiv preprint arXiv:1809.02627 (2020) doi: 10.48550/arXiv.1809.02627
14. CLOiSim: Multi-Robot Simulator. Accessed: 04.03.2025. https://github.com/lge-ros2/cloisim

15. Y. Song, S. Naji, E. Kaufmann, A. Loquercio, D. Scaramuzza, Flightmare: a flexible quadrotor simulator, in: Conference on Robot Learning (CoRL) (2020) 1–11. Accessed: 04.03.2025. https://rpg.ifi.uzh.ch/docs/CoRL20_Yunlong.pdf
16. M. Fernandez-Cortizas, M. Molina, P. Arias-Perez, R. Perez-Segui, D. Perez-Saura, P. Campoy, Aerostack2: a software framework for developing multi-robot aerial systems, in: ArXiv (2023). doi: 10.48550/arXiv.2303.18237.
17. F. L. Bellifemine, G. Caire, D. Greenwood, Developing multi-agent systems with JADE (Wiley Series in Agent Technology), John Wiley & Sons, 2007.
18. Jade. Accessed: 04.03.2025. https://jade.tilab.com/
19. P. Pierpaoli, T. T. Doan, J. Romberg, M. Egerstedt, Sequencing of multi-robot behaviors using reinforcement learning, in: Control Theory and Technology 19 (2021) 529–537. doi: 10.1007/s11768-021-00069-5
20. C. Street, B. Lacerda, M. Staniaszek, M. Mühlig, N. Hawes, Context-aware modelling for multi-robot systems under uncertainty, in: 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS'22). International Foundation for Autonomous Agents and Multiagent Systems (2022) 1228–1236. Accessed: 04.03.2025. https://www.ifaamas.org/Proceedings/aamas2022/pdfs/p1228.pdf

Одержано: 05.03.2025
Внутрішня рецензія отримана: 11.03.2025
Зовнішня рецензія отримана: 10.03.2025

Про авторів:

[1]Шевченко Руслан Сергійович,
кандидат технічних наук,
старший науковий співробітник.
https://orcid.org/0000-0002-1554-2019.

[1,2]Дорошенко Анатолій Юхимович,
доктор фізико-математичних наук,
професор, провідний науковий співробітник.
http://orcid.org/0000-0002-8435-1451.

[1,2]Лесик Валентин Олександрович,
інженер,
аспірант КПІ ім. Сікорського.
http://orcid.org/0000-0002-8307-5707.

[2]Савчук Олена Володимирівна,
кандидат технічних наук,
доцент кафедри інформаційних систем та технологій.
http://orcid.org/0000-0003-3176-7952.

[1]Яценко Олена Анатоліївна,
кандидат фізико-математичних наук,
старший науковий співробітник.
http://orcid.org/0000-0002-4700-6704.

Місце роботи авторів:

[1] Інститут програмних систем
НАН України,
тел. +38-067-407-32-33
E-mail: doroshenkoanatoliy2@gmail.com,
ruslan@shevchenko.kiev.ua,
oayat@ukr.net
Сайт: https://iss.nas.gov.ua

[2] Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»,
Сайт: https://ist.kpi.ua