МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Київський національний університет будівництва і архітектури

ПРОГРАМУВАННЯ РОБОТОТЕХНІЧНИХ ІНФОРМАЦІЙНИХ СИСТЕМ

Методичні вказівки
до виконання самостійної роботи та індивідуального завдання
для здобувачів першого (бакалаврського) рівня вищої освіти
спеціальності F6 «Інформаційні системи і технології»

Київ 2025

УДК 004.41
П78

Укладачі: Д. О. Міщук, канд. техн. наук, доцент;
В. П. Рашківський, канд. техн. наук, доцент

Рецензент О. О. Терентєв, д-р техн. наук, професор

Відповідальний за випуск В. П. Рашківський, канд. техн. наук, доцент

Затверджено на засіданні кафедри будівельних машин, протокол №10 від 27 травня 2025 року.

В авторській редакції.

Програмування робототехнічних інформаційних систем : П78 методичні вказівки до виконання самостійної роботи та індивідуального завдання / Д. О. Міщук, В.П. Рашківський. – Київ : КНУБА, 2025. – 88 с.

Містять завдання та методику виконання індивідуального завдання з курсу програмування робототехнічних інформаційних систем.

Призначено для здобувачі першого (бакалаврського) рівня вищої освіти спеціальності F6 «Інформаційні системи і технології».

ЗМІСТ

ЗАГАЛЬНІ ПОЛОЖЕННЯ

Метою курсу є вивчення основ робототехнічних інформаційних систем, засобів комунікації між роботами, людиною та способів програмування логіки функціонування інформаційної системи робота.

За результатами навчання здобувачі будуть:

знати:

– види, класифікацію і основні технічні параметри робототехніних систем;

– основні відомості про конструкції інформаційцних робототехнічних систем;

– компоненти виконавчої та інформаціної систем робота;

вміти:

– підбирати роботів для виконання поставленої задачі;

– програмувати інформаційні системи роботів;

– створювати алгоритми управління та здійснювати їхню реалізацію.

Індивідуальна робота виконується самостійно здобувачем за власним вибором теми проєкту. Обсяг роботи повинен бути до 10 – 12 сторінок тексту розміром 14 пт і одинарним міжрядковим інтервалом з рисунками або таблицями. Якщо наводиться алгоритм або фрагмент коду, тоді треба пояснити його роботу. В роботі обов'язково потрібно вказати список використаних джерел. В списку джерел рекомендовано робити посилання на оригінальні роботи, дослідження, монографії. Кількість джерел повинно бути більше одного.

Основні розділи роботи повинні розкривати такі питання:

– загальний огляд існуючої проблеми та існуючих рішень або власні міркування з приводу того, як це рішення може бути реалізовано;

– детальний опис одного або декількох рішень, алгоритмів або програмного коду;

– власний висновок щодо переваг і недоліків розглянутого рішення та можливостей його реалізації.

ЗАВДАННЯ № 1. Дослідити програми управління роботими в симуляторі CoppeliaSim

Мета роботи: ознайомитися із симулятором роботів CoppeliaSim та змоделювати візуально поведінку колісного робота з диференціальним приводом в даному симуляторі.

Основні теоретичні відомості

CoppeliaSim – це програмне забезпечення для віртуальної симуляції роботів, розроблене компанією Coppelia Robotics. У цій роботі буде детально розглянуто інтерфейс та можливості CoppeliaSim, структуру та особливості використання.

CoppeliaSim – це інтегроване середовище для симуляції робототехнічних систем, що дозволяє моделювати, тестувати та відлагоджувати роботів у віртуальному середовищі перед їх реальним використанням. Воно надає широкі можливості для моделювання різних типів роботів, включаючи маніпулятори, мобільні роботи, дрони тощо. Coppelia Robotics офіційно надає безкоштовну версію CoppeliaSim для навчання студентів.

Інтерфейс CoppeliaSim показано на рис. 1.1.

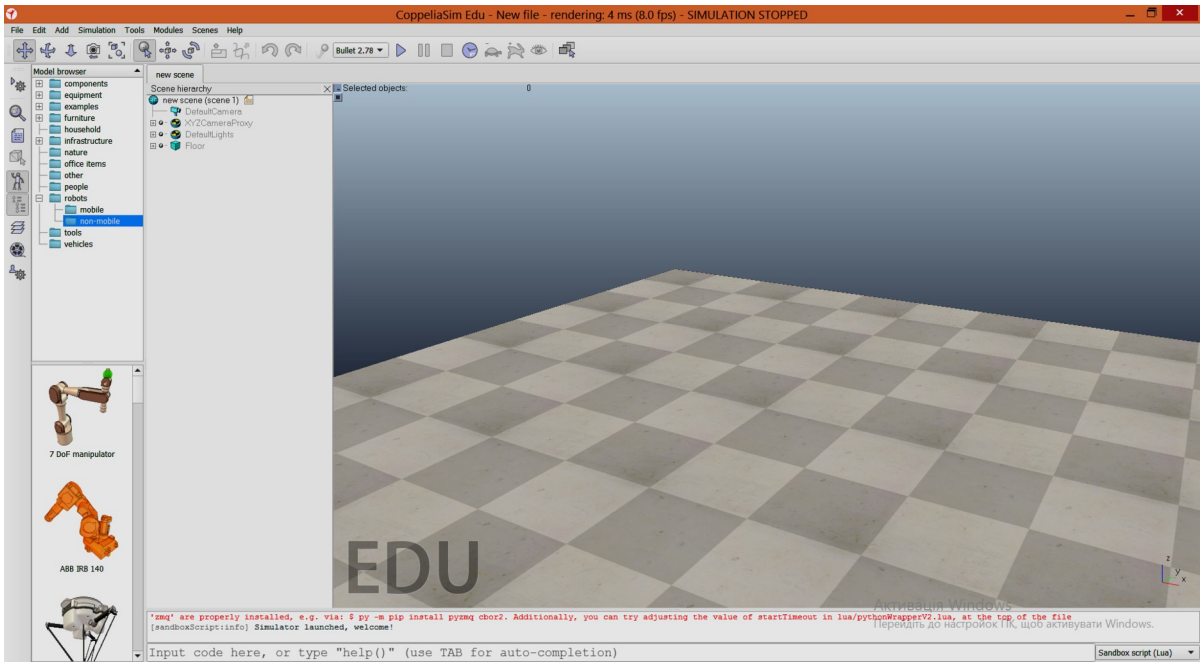


Рис. 1.1. Інтерфейс програмного середовища CoppeliaSim

В CoppeliaSim можна створювати об’єкти, змінювати їх фізичні властивості та взаємодіяти з ними, окрім цього даний симулятор

дозволяє створювати віртуальні середовища для симуляції робототехнічних систем з підтримкою аспектів реальної фізичної природи.

Інтерфейс CoppeliaSim складається з програмної панелі, меню навігації, панелі інструментів, вікна готових моделей, вікна дерева сцени, вікна сцени, вікна консолі, командного рядка. Програмна панель відображає тип ліцензії CoppeliaSim, назву файлу сцени, яка зараз відображається, час, використаний для одного проходу візуалізації (одного проходу відображення), і поточний стан симулятора (стан симуляції або тип активного режиму редагування). CoppeliaSim підтримує такі файли розширення: «*.ttt» (файли сцени) і «*.ttm» (файли моделей).

Меню навігації дозволяє отримати доступ майже до всіх функцій симулятора. У більшості випадків елементи на панелі меню активують діалогове вікно. Вміст панелі меню є контекстно-залежним (тобто залежить від поточного стану симулятора). До більшості функцій панелі меню також можна отримати доступ через спливаюче вікно, яке запускається подвійним натисканням на піктограму в ієрархічному поданні сцени або натисканням кнопки на панелі інструментів.

Панель інструментів містить функції, які часто застосовуються (наприклад, зміна режиму навігації, вибір іншої сторінки тощо). Доступ до функцій панелі інструментів можна отримати через командний рядок або спливаюче меню. На рис. 1.2 та 1.3 показано функції панелі інструментів.

Переглядач моделей (браузер моделей) відображається за замовчуванням, але його можна перемикати за допомогою відповідної кнопки на панелі інструментів. Переглядач відображає у своїй верхній частині структуру каталогів моделей CoppeliaSim, а в нижній частині – мініатюри моделей, які містяться у вибраній теці (рис. 1.4). Мініатюри можна перетягувати в сцену, щоб автоматично завантажити відповідну модель.

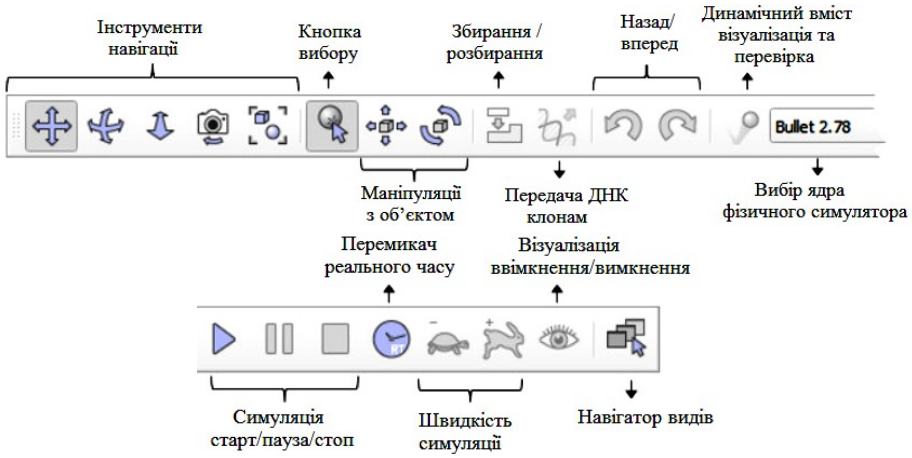


Рис. 1.2. Функції панелі інструментів

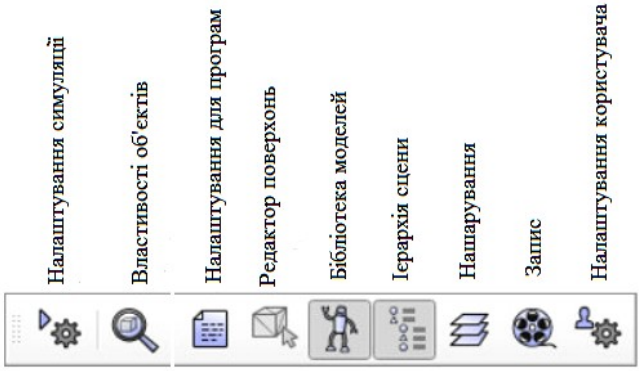


Рис. 1.3. Функції панелі інструментів

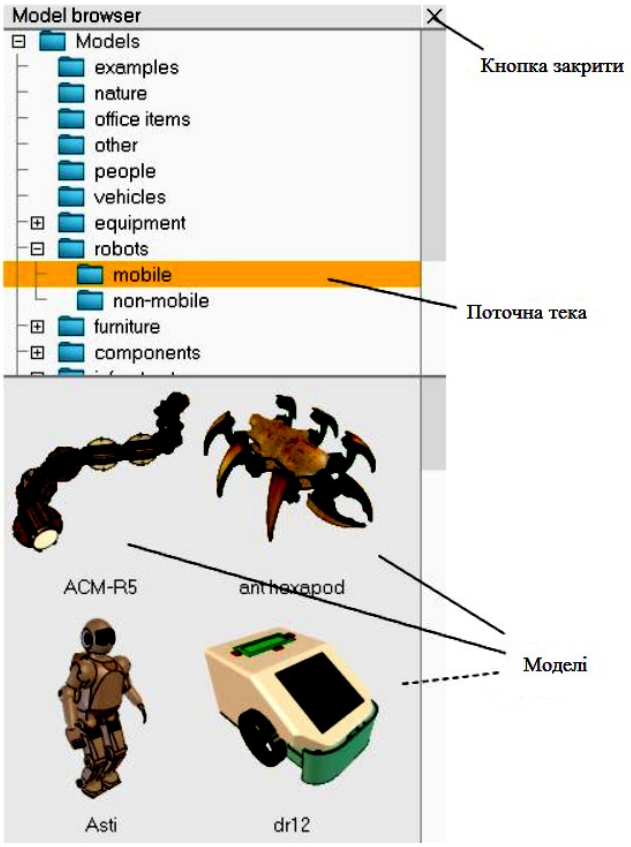


Рис. 1.4. Переглядач моделей

Дерево сцени (рис. 1.5) є видимим за умовчанням, але його можна перемикати за допомогою відповідної кнопки на панелі інструментів. Воно відображає вміст сцени (тобто всі об’єкти, що складають сцену). Оскільки об’єкти сцени побудовані в ієрархічну структуру, ієрархія сцени відображає дерево цієї ієрархії, а окремі елементи можна розгорнути або згорнути. Подвійне натискання піктограми відкриває/закриває діалогове вікно властивостей, пов’язане з піктограмою. Подвійний клік на псевдонімі об’єкта дозволяє редагувати його. Колесо миші, а також перетягування смуг прокручування перегляду ієрархії сцени дозволяє зміщувати вміст вгору/вниз або вліво/вправо. Об’єкти в ієрархії сцени можна перетягувати на інший об’єкт, щоб створити зв’язок «батько-нащадок». Ієрархія сцен відображатиме інший вміст, якщо симулятор перебуває в режимі редагування.

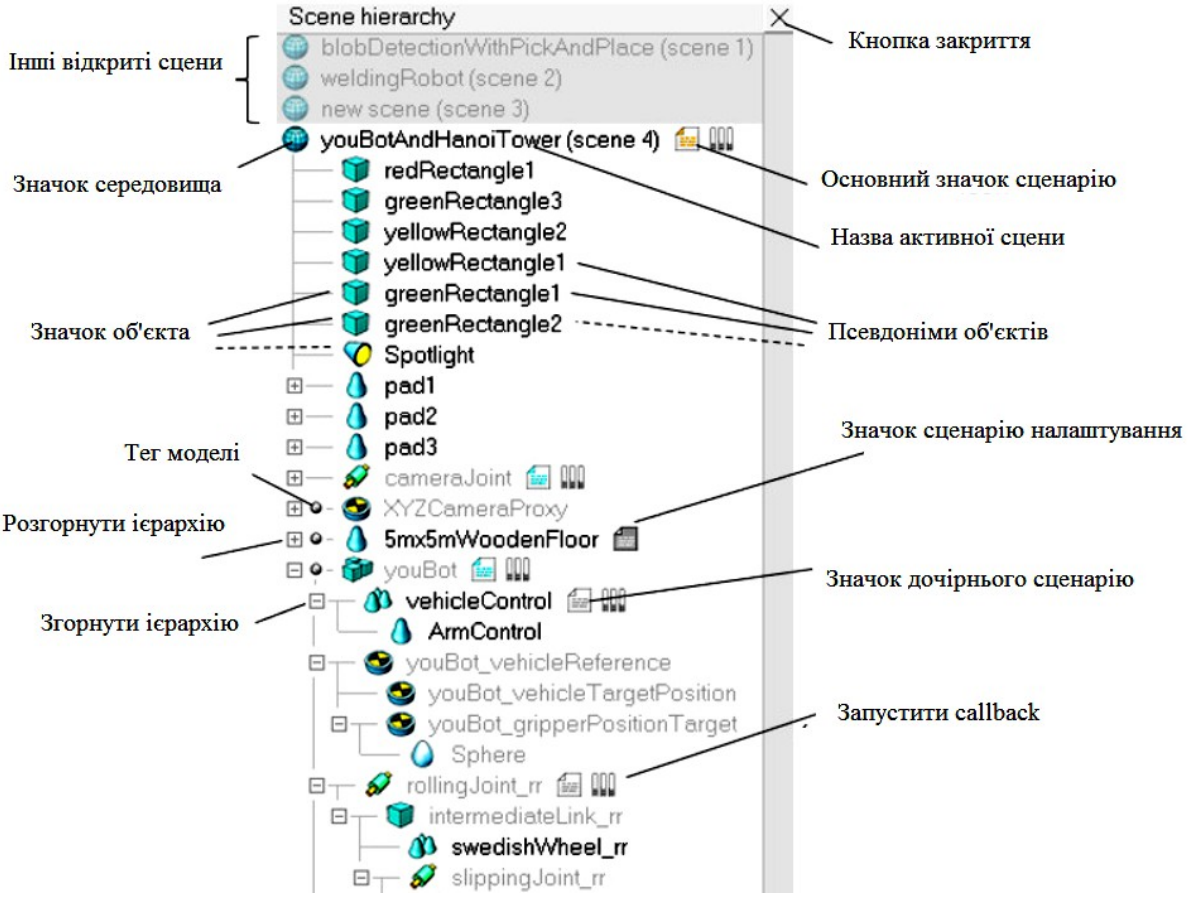


Рис. 1.5. Вид дерева сцени

Кожна сцена може містити до 8 видів сцени. Вид сцени можна розглядати як контейнер для представлень – вигляд зверху, знизу, збоку, перспектива та ін.

На кожному виді сцени може бути необмежена кількість представлень об’єктів. Представлення використовується для відображення сцени (містить середовище та об’єкти), видимої крізь видимий об’єкт (наприклад, камери, графіки або датчики зору).

Інформаційний текст відображає інформацію, пов’язану з поточним вибором об’єкта/елемента та поточними станами або параметрами моделювання. Відображення тексту можна перемикати за допомогою однієї з двох маленьких кнопок у верхній лівій частині сторінки. Іншу кнопку можна використовувати для перемикання білого фону, що забезпечує кращий контраст залежно від кольору фону сцени.

В консолі відображається інформація, пов’язана з виконуваними функціями, командами, а також відображається інформація про помилки від інтерпретаторів сценаріїв. Зі сценарію користувач також може виводити рядки в рядок стану або на консоль за допомогою функції sim.addLog. Рядок стану за замовчуванням відображає лише два рядки, але його розмір можна змінити за допомогою маркера горизонтального розділення.

Командний рядок дозволяє вводити текст з процедурами в рядок CoppeliaSim та виконувати код Python або Lua на льоту. Код можна запустити в сценарії пісочниці або будь-якому іншому активному сценарії в CoppeliaSim.

Користувацькі інтерфейси користувача – це визначений користувачем набір інтерфейсів, які можна використовувати для відображення інформації (тексту, зображень тощо) або користувацького діалогу, що дозволяє взаємодіяти з користувачем в налаштований спосіб.

Спливаючі меню – це меню, які з’являються після натискання правої кнопки миші. Щоб активувати спливаюче меню, переконайтеся, що миша не рухається під час кліку, інакше може бути активовано режим обертання камери. Кожна поверхня у вікні програми (наприклад, перегляд ієрархії сцени, сторінка, перегляд тощо) може запускати інше спливаюче меню (контекстно-залежне). Вміст спливаючих меню також може змінюватися залежно від поточного стану симуляції або режиму редагування. До більшості функцій спливаючого меню також можна

отримати доступ через командний рядок меню, за винятком пунктів меню перегляду, які з'являються лише, коли спливаюче меню активовано на представленні або сторінці виду.

Завантаження стандартного робота на поточну сцену. Стандартно CoppeliaSim відкриває порожню сцену без робота. Щоб додати на таку сцену стандартного колісного робота з бібліотеки роботів, виберемо в бібліотеці моделей теку **robots.mobile** (рис. 1.6) та знайдемо стандартну модель **pioneer p3dx** (рис. 1.7). Натиснувши на робота та перемістивши його на сцену, буде завантажено даного робота на поточну сцену.

Під час додавання на сцену стандартного робота зазвичай також до цього робота буде додано стандартну програму його поведінки, виконану розробниками даного програмного середовища. В CoppeliaSim можна додавати власні цифрові моделі роботів, які спроєктовані в CAD системах, але в такому випадку потрібно буде налаштувати ступені рухомості і датчики для виконання управління і програмування. Стандартні цифрові моделі вже містять налаштовані параметри із відомими фізичними якостями.

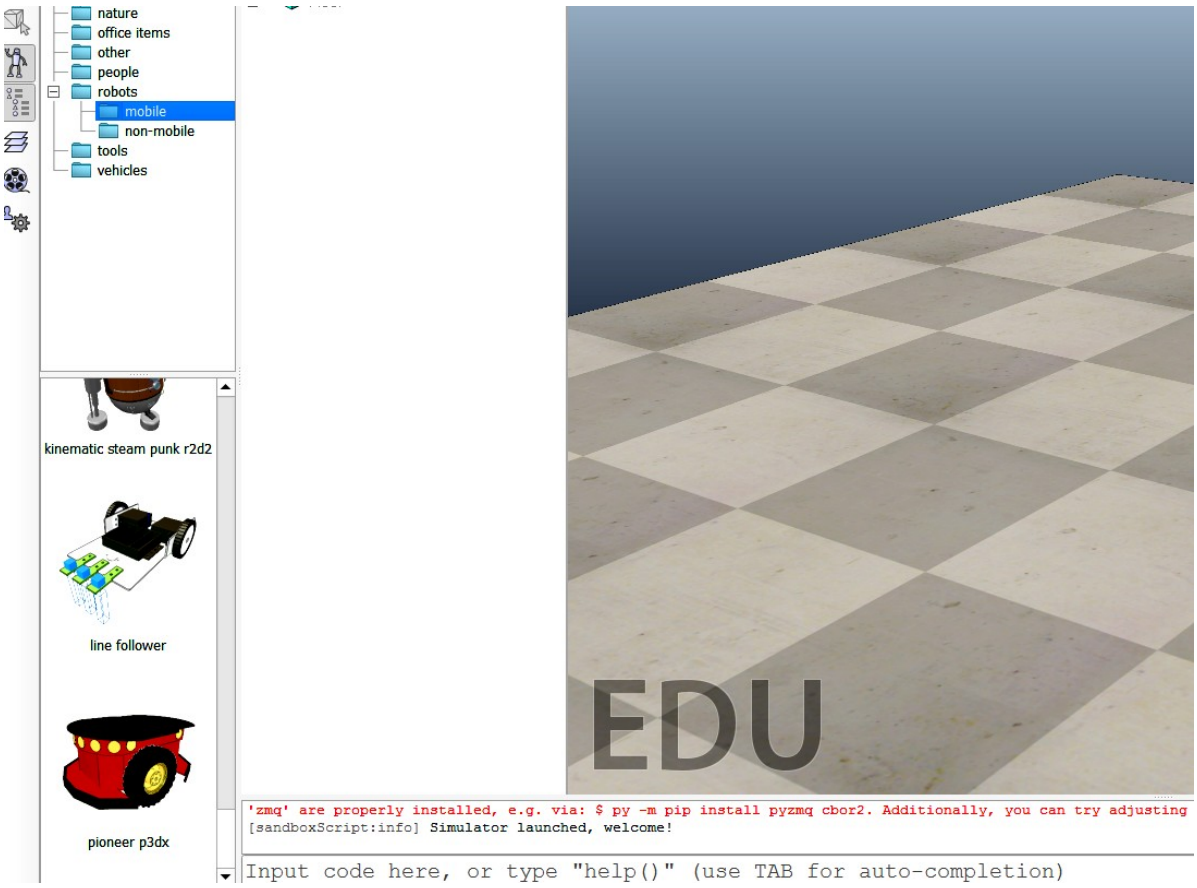


Рис. 1.6. Порядок додавання робота на сцену

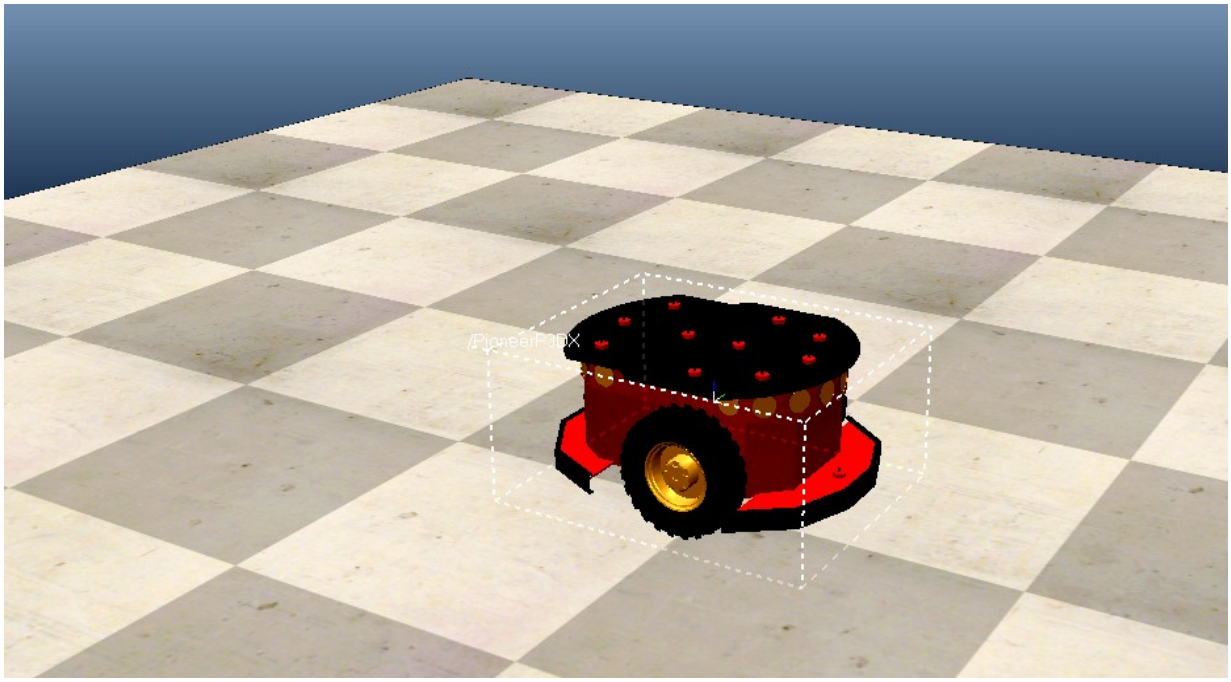


Рис. 1.7. Головна сцена з роботом pioneer p3dx

Для перегляду стандартної програми управління доданого робота перейдіть до дерева моделей, знайдіть назву робота в дереві моделей і двічі натисніть на піктограму документа (див. рис. 1.8). Це призведе до відкриття нового вікна з кодом програми управління для даного робота на мові Lua (рис. 1.9).

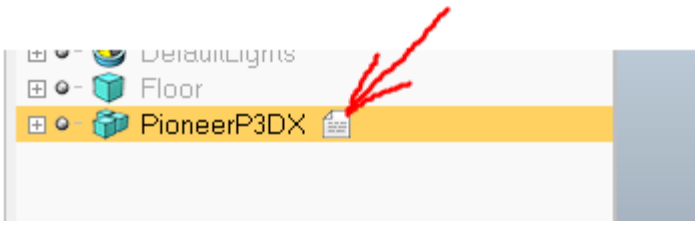
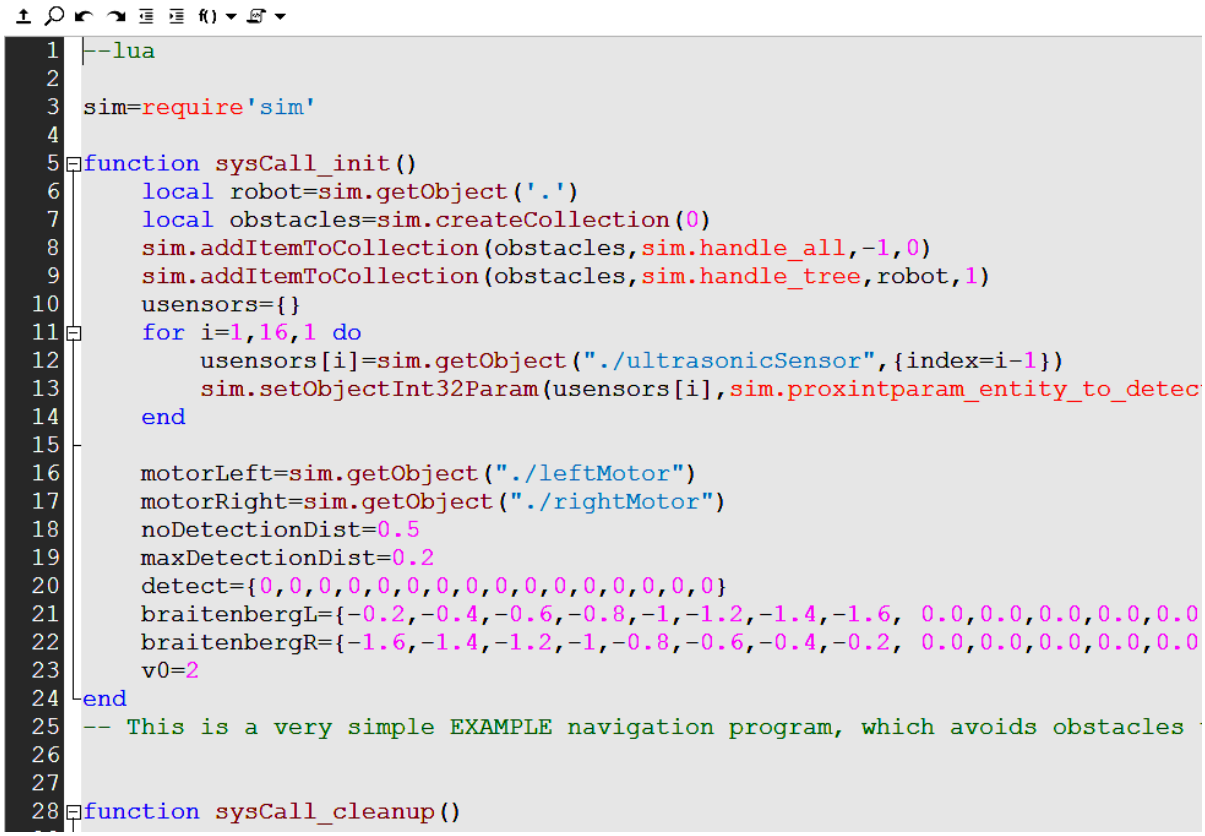


Рис. 1.8. Піктограма зі скриптом для робота



```lua
--lua

sim=require'sim'

function sysCall_init()
    local robot=sim.getObject('.')
    local obstacles=sim.createCollection(0)
    sim.addItemToCollection(obstacles,sim.handle_all,-1,0)
    sim.addItemToCollection(obstacles,sim.handle_tree,robot,1)
    usensors={}
    for i=1,16,1 do
        usensors[i]=sim.getObject("./ultrasonicSensor",{index=i-1})
        sim.setObjectInt32Param(usensors[i],sim.proxintparam_entity_to_detec
    end

    motorLeft=sim.getObject("./leftMotor")
    motorRight=sim.getObject("./rightMotor")
    noDetectionDist=0.5
    maxDetectionDist=0.2
    detect={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
    braitenbergL={-0.2,-0.4,-0.6,-0.8,-1,-1.2,-1.4,-1.6, 0.0,0.0,0.0,0.0,0.0
    braitenbergR={-1.6,-1.4,-1.2,-1,-0.8,-0.6,-0.4,-0.2, 0.0,0.0,0.0,0.0,0.0
    v0=2
end
-- This is a very simple EXAMPLE navigation program, which avoids obstacles


function sysCall_cleanup()
```

Рис. 1.9. Вікно скрипту з програмою керування для стандартного робота

Підключення сторонньої управляючої програми на інших мовах програмування для даного симулятора може здійснюватися через вебсокети та інструменти API (див. офіційну документацію). CoppeliaSim має вбудовану підпримку Python, але для цього на комп'ютері, де буде запущено симулятор, потрібно додатково глобально встановити бібліотеки pyzmq та cbor2 для Python. Інший спосіб підключення Python – це застосування веб-сервера з відповідним API, яке надає CoppeliaSim разом з програмою симулятора (в каталозі LegasyRemoteAPI).

Підключення через LegasyRemoteAPI:

1. Створіть власний каталог для коду на Python. В створеному каталозі зробіть каталог api – це хороша практика написання прикладних інтерфейсів.

2. Перейдіть за вказаним шляхом (**диск та каталог із встановленою програмою**\CoppeliaRobotics\CoppeliaSimEdu\programming\legacyRemote Api\remoteApiBindings\python\python) та скопіюйте всі файли з даного каталогу у власний каталог api.

3. Перейдіть за вказаним шляхом (**диск та каталог із встановленою програмою**\CoppeliaRobotics\CoppeliaSimEdu\programming\legacyRemote Api\remoteApiBindings\lib\lib) та відповідно до вашої операційної системи скопіюйте файл з назвою **remoteApi** до вашого власного каталогу api з програмою. Таким чином структура файлів у вашому проєкті повинна бути схожою, як на рис. 1.10.

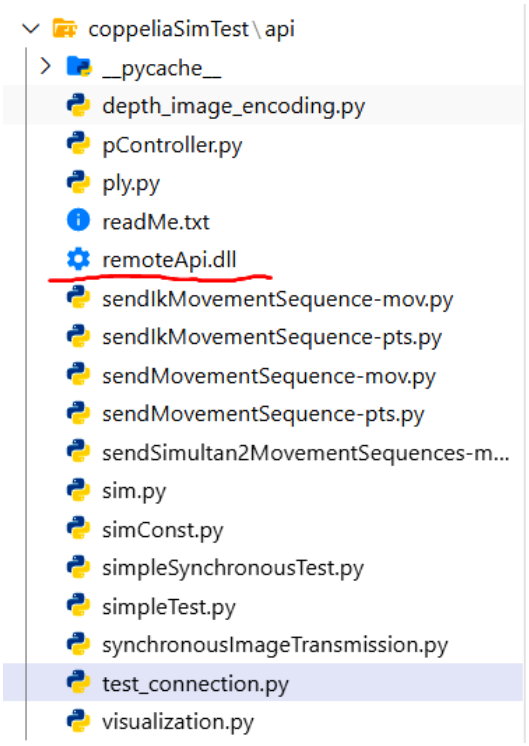


Рис. 1.10. Структура файлів в робочому каталозі з api

4. Тепер можна створювати власні скрипти та запускати їх в CoppeliaSim. Файли можна створювати як всередині каталогу api для вашого проєкту, так і зовні в інших каталогах. Для запуску власних скриптів, за даного способу, попередньо потрібно ініціювати запуск сервера командою **simRemoteApi.start(19999)**. Для цього в програмі CoppeliaSim додайте до головної сцени Floor скрипт **Add → Associated Child Script → Lua,** як показано на рис. 1.11.

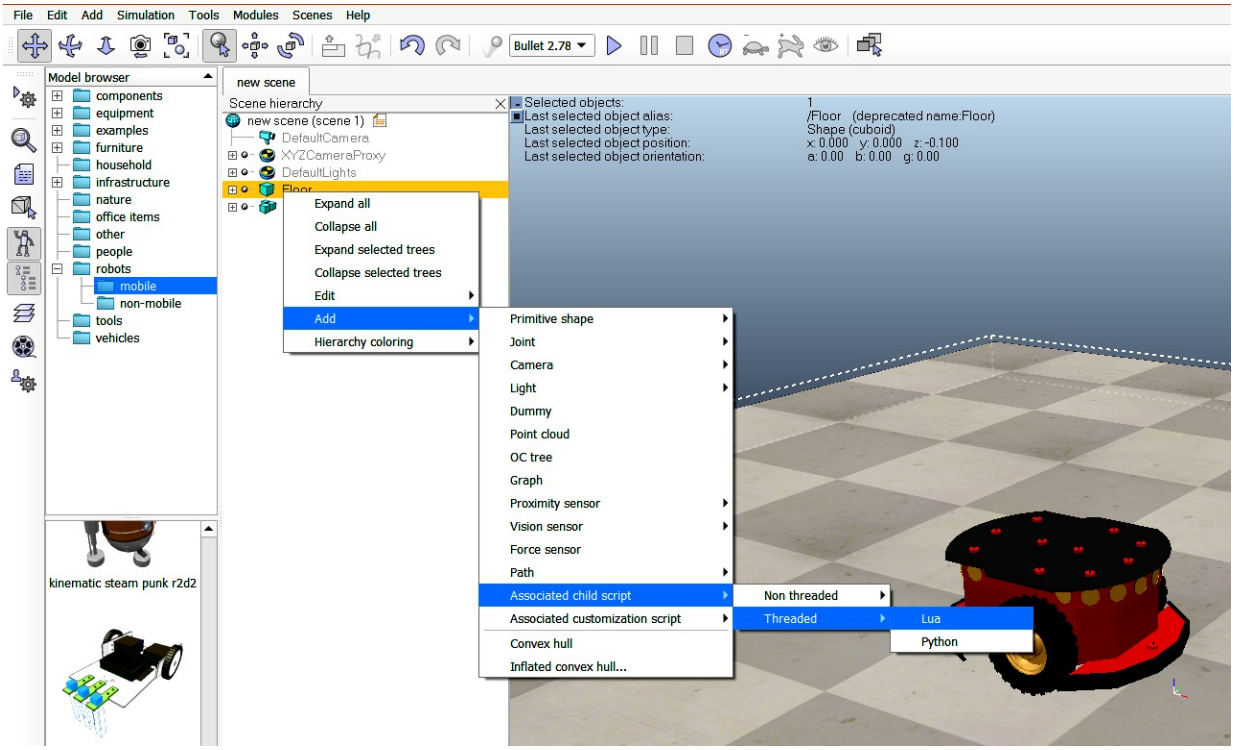


Рис. 1.11. Ініціалізація скрипту для вибраного об’єкта

Далі потрібно відкрити створений скрипт для сцени Floor та додати інструкцію ініціалізації сервера, як показано на рис. 1.12.

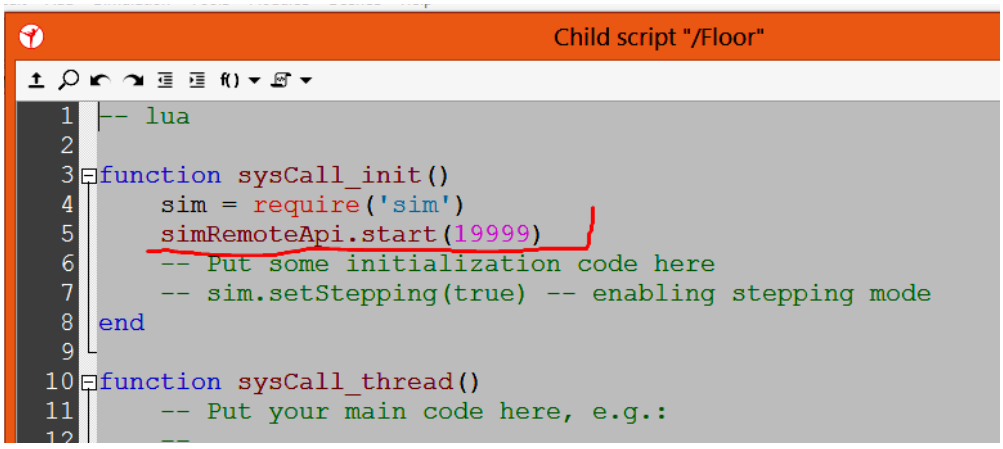


Рис. 1.12. Ініціалізація сервера simRemoteApi.start(19999)

Параметр **19999** вказує на порт, на якому буде запущений сервер віддаленого API CoppeliaSim.

Створіть власний каталог для тестування коду управління робота. Наприклад, в даному проєкті, оскільки планується працювати з роботом PioneerP3DX, створимо відповідний каталог (структуру

проєкту див. на рис. 1.13). Перед початком роботи рекомендується (необов'язково для цієї роботи) ознайомитися з конструкцією робота PioneerP3DX за наступним посиланням: https://www.inf.ufrgs.br/~prestes/Courses/Robotics/manual_pio-neer.pdf.

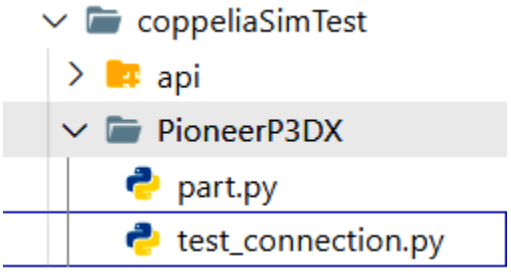


Рис. 1.13. Структура стартового проєкту для тестування робота PioneerP3DX

Створимо в каталозі PioneerP3DX файл **test_connecton.py** для тестування підключення по API сервера CoppeliaSim з наступним вмістом:

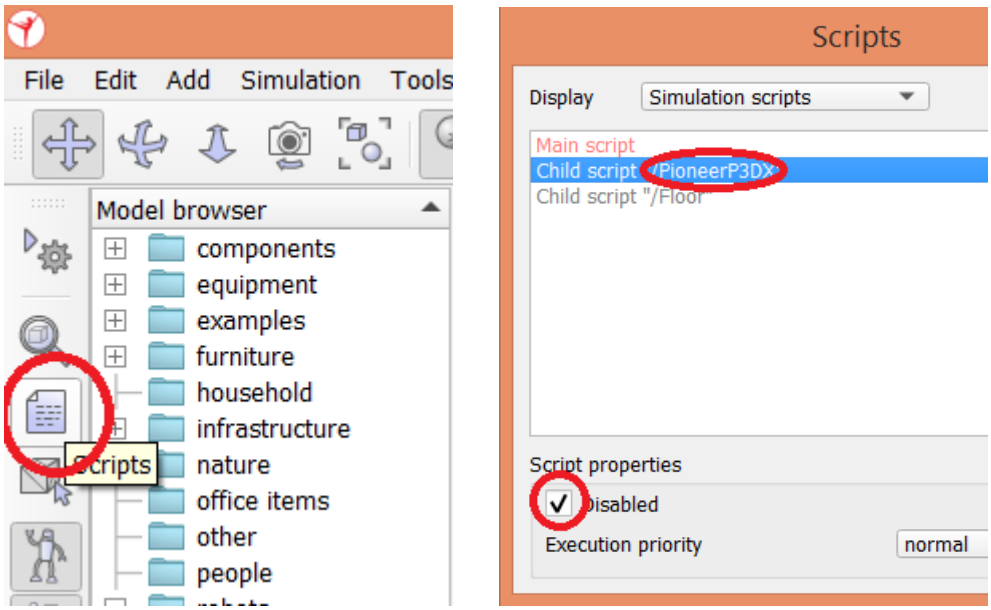
```
# Make sure to have the server side running in CoppeliaSim:
# to be executed just once, at simulation start:
# simRemoteApi.start(19999)
# then start simulation, and run this program.
# IMPORTANT: for each successful call to simxStart, there
# should be a corresponding call to simxFinish at the end!

from sys import path
import time
path.append('..')
import api.sim as sim
print ('Program started')
sim.simxFinish(-1)
clientID = sim.simxStart('127.0.0.1', 19999, True, True,
5000, 5)
if clientID != -1:
    print ('Connected to remote API server')
else:
    print ('Failed connecting to remote API server')
time.sleep(1)
error_code, motorLeft = sim.simxGetObjectHandle(clientID,
'./PioneerP3DX/leftMotor', sim.simx_opmode_oneshot_wait)
error_code, motorRight = sim.simxGetObjectHandle(clientID,
'./PioneerP3DX/rightMotor', sim.simx_opmode_oneshot_wait)

error_code = sim.simxSetJointTargetVelocity(clientID,
motorLeft, 0.2, sim.simx_opmode_oneshot_wait)
```

```
error_code = sim.simxSetJointTargetVelocity(clientID,
motorRight, -0.2, sim.simx_opmode_oneshot_wait)
print ('Program ended')
```

Перед початком симуляції в CoppeliaSim необхідно вимкнути стандартний код для робота, бо управління буде здійснюватися через віддалений доступ. Для цього в програмі CoppeliaSim активуйте вкладку Scripts на боковому меню (див. рис. 1.14, *а*) та відкрийте вікно зі скриптами для даної сцени, вибравши PioneerP3DX, деактивуйте його (див. рис. 1.14, *б*).



а *б*

Рис. 1.14. Режим деактивації скриптів сцени

Спочатку запустіть симулятор в меню програми, натиснувши відповідну активність в головному меню, а далі запустіть ваш скрипт на Python запуску команд на роботі. Якщо все зроблено правильно, програма повинна запустити переміщення робота, що відобразиться в симуляторі.

В запропонованому коді Python конструкція функції **simxStart('127.0.0.1', 19999, True, True, 5000, 5)** запустить підключення до сервера віддаленого API у CoppeliaSim. Параметри запуску цієї функції наступні:

- **'127.0.0.1'** – це IP-адреса, за якою відбувається підключення до сервера. У цьому випадку **'127.0.0.1'** вказує на локальний комп'ютер, на якому працює CoppeliaSim. Якщо CoppeliaSim працює на іншому комп'ютері, потрібно буде вказати його IP-адресу;

- **19999 –** це номер порту, на якому працює сервер віддаленого API у CoppeliaSim. За замовчуванням це 19999, але його можна змінити у налаштуваннях CoppeliaSim;
- **True** – це параметр, який вказує на автоматичний запуск симуляції у CoppeliaSim, якщо вона ще не запущена. Якщо це значення **False**, тоді симуляція не буде автоматично запущена;
- **True** – це параметр, який вказує на очікування з'єднання, якщо воно ще не встановлене. Якщо це значення **False**, функція **simxStart** буде намагатися підключитися без очікування з'єднання;
- **5000** – це таймаут очікування з'єднання в мілісекундах. Якщо з'єднання не буде встановлене протягом цього часу, функція **simxStart** поверне помилку;
- **5** – це кількість спроб підключення. Якщо підключення не вдалось, функція **simxStart** буде намагатися підключитися ще 5 разів перед поверненням помилки.

Розглянемо тепер інструкцію **error_code, motorLeft = sim.simxGetObjectHandle (clientID, './PioneerP3DX/leftMotor', sim.simx_ opmode_oneshot_wait)**. Функція **simxGetObjectHandle** призначена для пошуку конкретного об'єкта за його повним іменем. В даному випадку ім'я об'єкта вказується у виді рядка './PioneerP3DX/leftMotor' (придивіться до сцени на симуляторі і знайдіть це значення рядка). Інші параметри даної інструкції наступні:

- **error_code –** це змінна, в яку буде збережено код помилки, який повертається функцією **simxGetObjectHandle**. Якщо процедура запуску пройшла успішно, цей параметр буде мати значення 0, інакше буде повернуто значення, відмінне від нуля, що вказує на помилку;
- **motorLeft –** це власна змінна, в яку буде збережено дескриптор об'єкта (наприклад, мотора лівого колеса) у CoppeliaSim. Якщо процедура успішна, цей параметр буде мати значення, яке можна використовувати для подальшої роботи з цим об'єктом;
- **Sim.simxGetObjectHandle –** це основна функція в даній процедурі, яка розраховує дескриптор об'єкта за його іменем на сцені CoppeliaSim;
- **clientID** – це ідентифікатор клієнта, який був створений під час підключення до сервера віддаленого API і використовується для взаємодії із сервером;

- **'./PioneerP3DX/leftMotor'** – це ім’я об’єкта сцени, для якого потрібно отримати дескриптор. В цьому випадку це ім’я мотора лівого колеса у сцені CoppeliaSim;
- **sim.simx_opmode_oneshot_wait –** це режим, який вказує, що функція має чекати доти, доки процедура не буде завершена. Це означає, що вона буде блокувати виконання коду доти, доки не буде отримано результат від сервера.

Для запуску мотора робота розглянемо такий код: **error_code = sim.simxSetJointTargetVelocity (clientID, motorLeft, 0.2, sim.simx_opmode _oneshot_wait)**. Функція **simxSetJointTargetVelocity** призначена для встановлення цільової швидкості обертання для з’єднаної з нею кінематичної пари (наприклад, мотора) у CoppeliaSim. Параметри виразу:

- **error_code**: – це змінна, в яку буде збережено код помилки, який повертається функцією **simxSetJointTargetVelocity**. Якщо процедура пройшла успішно, цей параметр буде мати значення 0, інакше буде повернуто значення, відмінне від нуля, що вказує на помилку;
- **sim.simxSetJointTargetVelocity** – це функція, яка запускається для встановлення цільової швидкості обертання для кінематичної пари в CoppeliaSim;
- **clientID** – ідентифікатор клієнта;
- **motorLeft** – це дескриптор об’єкта (наприклад, мотора лівого колеса), для якого потрібно встановити цільову швидкість обертання;
- **0.2** – це значення цільової швидкості обертання, яке встановлюється для кінематичної пари. У цьому випадку 0.2 означає, що мотор повинен обертатися зі швидкістю 0.2 одиниці часу вперед (якщо буде від’ємне значення, обертання буде назад по ходу руху робота).
- **sim.simx_opmode_oneshot_wait** – це режим, який вказує, що функція має чекати доти, доки функції не завершиться. Це означає, що вона буде блокувати виконання коду доти, доки не буде отриманий результат від сервера.

Для роботи із сенсорами розглянемо таку частину коду:

```
error,visl=sim.simxGetObjectHandle(clientID,'Left_sensor',
sim.simx_opmode_oneshot_wait)
error,visr=sim.simxGetObjectHandle(clientID,'Right_sensor',
sim.simx_opmode_oneshot_wait)
```

```
while 1:

error,result_l,data_l=sim.simxReadVisionSensor(clientID,vis
l, sim.simx_opmode_oneshot_wait)

error,result_r,data_r=sim.simxReadVisionSensor(clientID,vis
r, sim.simx_opmode_oneshot_wait)
    # print(V1)
    # print(data_l[0])
    # print(data_r[0])
   if (data_l[0][11] > 0.4) and (data_r[0][11] > 0.4):
      error = sim.simxSetJointTargetVelocity(clientID,
         Left_wheel, w1, sim.simx_opmode_oneshot_wait)
      error = sim.simxSetJointTargetVelocity(clientID,
         Right_wheel, w2, sim.simx_opmode_oneshot_wait)
  elif (data_l[0][11] < 0.4) and (data_r[0][11] > 0.4):
      error = sim.simxSetJointTargetVelocity(clientID,
         Left_wheel, 0, sim.simx_opmode_oneshot_wait)
      error = sim.simxSetJointTargetVelocity(clientID,
         Right_wheel, w2*2, sim.simx_opmode_oneshot_wait)
  elif (data_l[0][11] > 0.4) and (data_r[0][11] < 0.4):
      error = sim.simxSetJointTargetVelocity(clientID,
         Left_wheel, w1*2, sim.simx_opmode_oneshot_wait)
      error = sim.simxSetJointTargetVelocity(clientID,
         Right_wheel, 0, sim.simx_opmode_oneshot_wait)
```

ПОСЛІДОВНІСТЬ ВИКОНАННЯ РОБОТИ

1. Встановити симулятор CoppeliaSim та ознайомитися з інтерфейсом даного симулятора.

2. Ознайомитися з меню навігації та довідником для цього симулятора.

3. Створити першу сцену та додати на сцену робота.

4. Дослідити характеристики вибраного робота за його інструкцією.

5. Запрограмувати робота на його переміщення.

6. Розробити власний API для взаємодії з роботом для функцій руху вперед, назад, ліворуч, праворуч, зупинки.

7. Дослідити самостійно послідовність роботи з датчиками в симуляторі CoppeliaSim.

8. Дати відповідь на запитання:

– що таке лідар?
– які види датчиків використовуються для вимірювання відстані?
– як в CopeliaSim отримати інформацію з датчика робота?

ЗАВДАННЯ № 2. Ознайомитися із засобами програмування комп'ютерного зору роботів

Мета роботи: ознайомитися з бібліотекою комп'ютерного зору OpenCV для отримання даних із зображень в інформаційно-вимірювальних системах.

Основні теоретичні відомості

OpenCV – це бібліотека з відкритим кодом для розробки додатків комп'ютерного зору. Бібліотеку можна використовувати як в академічних, так і в комерційних цілях під ліцензією BSD, що дозволяє вільно застосовувати, поширювати і адаптовувати програмне забезпечення.

Починаючи з версії бібліотеки 2.2, в бібліотеці замість універсальних модулів cxcore, cvaux, highGUI та інших створено набір компактних модулів із вужчою спеціалізацією:

- opencv_core – основна функціональність. Включає базові структури, обчислення (математичні функції, генератори випадкових чисел) і лінійну алгебру, DFT , DCT , введення/виведення для XML і YAML і т. д;
- opencv_imgproc – обробка зображень (фільтрація, геометричні перетворення, перетворення колірних просторів тощо);
- opencv_highgui – простий UI, введення/виведення зображень та відео;
- opencv_ml – моделі машинного навчання (SVM, дерева рішень, навчання зі стимулюванням тощо);
- opencv_features2d – розпізнавання та опис плоских примітивів (SURF, FAST та інші, включаючи спеціалізований фреймворк);
- opencv_video – аналіз руху та відстеження об'єктів (оптичний потік, шаблони руху, усунення фону);
- opencv_objdetect – виявлення об'єктів на зображенні (знаходження осіб за допомогою алгоритму Віоли-Джонса , розпізнавання людей HOG і т. д.);
- opencv_calib3d – калібрування камери, пошук стереовідповідності та елементи обробки тривимірних даних;
- opencv_flann – бібліотека швидкого пошуку найближчих сусідів (FLANN 1.5) та обгортки OpenCV;

- opencv_contrib – супутній код, який ще не готовий для застосування;
- opencv_legacy – застарілий код, збережений для зворотної сумісності;
- opencv_gpu – прискорення деяких функцій OpenCV за рахунок CUDA, створено за підтримки NVidia.

На офіційному веб-сайті OpenCV за адресою http://opencv.org/ можна знайти останню версію бібліотеки, онлайнову документацію, що описує інтерфейс прикладного програмування (Application Programming Interface, API), та багато інших корисних ресурсів щодо OpenCV.

Для початку роботи з OpenCV потрібно встановити бібліотеку. Під час роботи з Python можна застосувати бібліотеки OpenCV, які було розроблено спеціально для даної мови програмування. Для виконання даної роботи створимо проєкт для роботи з OpenCV:

1. В будь-якому місці створимо каталог проєкту з назвою «python-cv».
2. В створеному каталозі створити і активувати віртуальне середовище Python:
```
>>> python -m venv venv
>>> .\venv\Scripts\activate
```
3. В активованому віртуальному середовищі встановити бібліотеку OpenCV для Python
```
>>> pip install opencv-contrib-python
```
альтернатива
```
>>> pip install opencv-python
```
4. Перевіряємо встановлені бібліотеки
```
>>> pip list
```

Альтернативна бібліотека opencv-python є основною бібліотекою opencv_core і не містить додаткової функціональності, а запропонована як основна, opencv-contrib-python – містить, окрім базового ядра бібліотеки, ще додаткові функціональності, які можуть бути використані у відкритому доступі.

Для перевірки коректності підключення OpenCV необхідно завантажити тестове зображення (https://bit.ly/3oWSyLg) до

відповідного каталогу img створеного проєкту «python-cv» та завантажити наступний код:

```
#read.py
import cv2 as cv
img = cv.imread('img/cat.jpg')
cv.imshow('Cat', img)
cv.waitKey(0)
```

Результатом виконання програми на екрані монітору повинно у окремому вікні відобразитися тестове зображення.

В даному коді спочатку виконується імпортування потрібної бібліотеки та оголошення класів і функцій, які потрібно використовувати. Всі класи і функції в API OpenCV стандартно визначаються в просторі імен cv2. Запуск функції cv.imread() зчитує зображення з файлу, декодує його і виділяє пам’ять. Для відображення зображень на екрані монітору можна застосувати декілька функцій, проте найбільш проста у роботі – функція cv.imshow(), якій в якості параметрів необхідно передати назву вікна (перший параметр) та посилання на зображення (другий параметр). Оскільки вікно консолі, а отже і відображення картинки завершиться, коли головний потік програми досягне кінця головної функції, в код додано додаткову функцію cv.waitKey(0), яка запустить процес очікування основного потоку доти, доки користувач не натисне на будь-яку клавішу, перш ніж завершити програму.

Досить часто зображення можуть мати занадто великий або малий розмір, що не достатньо для цього коректного відображення на екрані монітора. Для масштабування та непропорційної зміни розмірів зображень в OpenCV є спеціальна функціональність. Розглянемо функцію для зміни розміру зображення:

```
def rescaleFrame(frame, scale = 0.75):
    height = int(frame.shape[0] * scale)
    width = int(frame.shape[1] * scale)
    dimentions = (width, height)
    return cv.resize(frame, dimentions, interpolation =
                                        cv.INTER_AREA)
```

Застосуємо дану функцію до тестового зображення:

```
img_scale = rescaleFrame(img, 0.5)
```

Функція cv.resize() застосовується для безпосередньої зміни розміру зображення. В якості параметру дана функція приймає змінну

алгоритму інтерполяції – interpolation = cv2.INTER_XXX, де XXX може приймати наступні значення: NEAREST (інтерполяція найближчого сусіда); LINEAR (білінійна інтерполяція); CUBIC (бікубічна інтерполяція); AREA (перерахунок із використанням відношення площ пікселів); LANCZOS4 (інтерполяція Lanczoc за околицями 8×8). Окрім алгоритму інтерполяції, дана функція приймає на вхід початкове зображення та розмір нового фрейму у виді кортежу значень ширини та висоти.

Для створення власного зображення на екрані монітора запустимо наступний код:

```
import cv2 as cv
import numpy as np
blank = np.zeros((500, 500, 3), dtype='uint8')
cv.imshow('Blank', blank)
blank[:] = 0,255,0
cv.imshow('Cgeen', blank)
cv.waitKey(0)
```

В результаті виконання буде створено матрицю, розміром 500 на 500. Створена матриця може буде конвертована в набір пікселів та відображена на екрані (за стандартом всі пікселі дорівнюють 0, тому відображатиметься темний колір). Також можна змінити код пікселів та зафарбувати їх в зелений колір і отримати таке зображення (див. рис. 2.1).

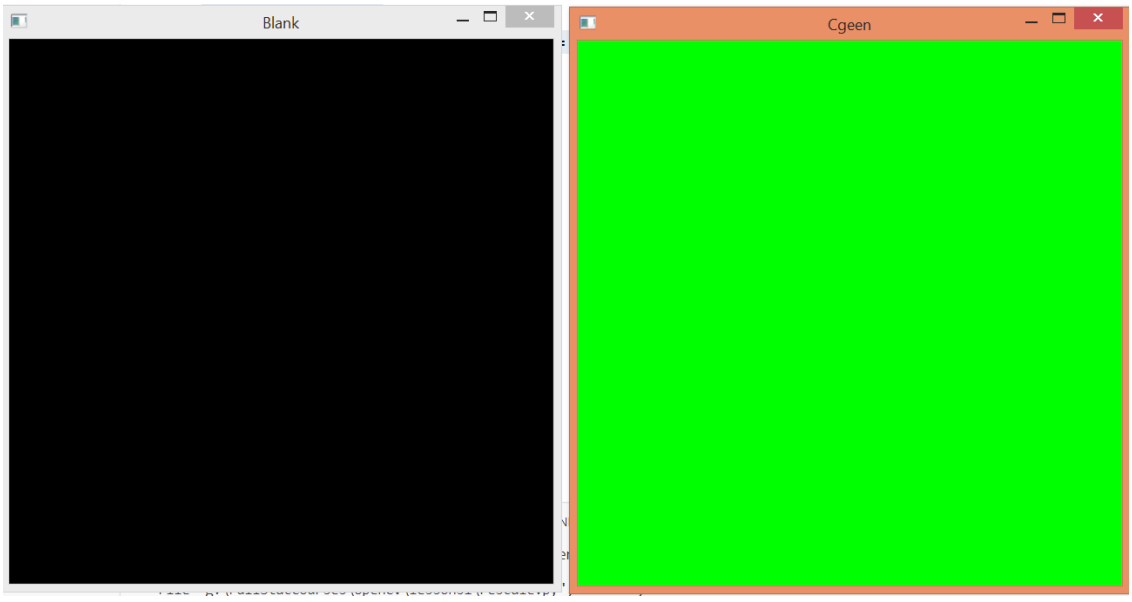


Рис. 2.1. Екран створених пікселів

OpenCV підтримує RGB формат каналів кольорового зображення (рис. 2.2) і може здійснювати конвертування в інші формати каналів, наприклад HSV. *HSV* (також *HSB*) – колірна модель, заснована на трьох характеристиках кольору: колірному тоні (Hue), насиченості (Saturation) і значенні кольору (Value), який також називають яскравістю (Brightness).

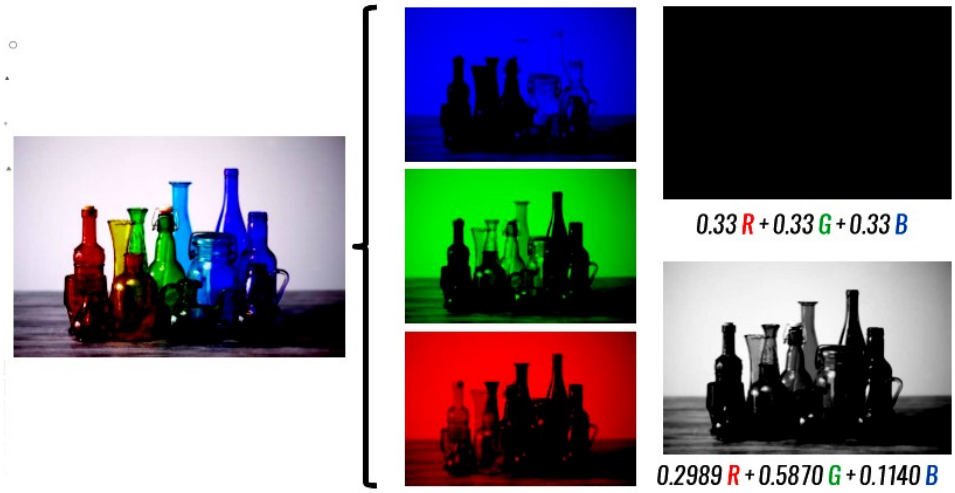


Рис. 2.2. Приклад RGB формату кольорового зображення

Найпростіший спосіб зобразити HSV в тривимірний простір – скористатися циліндричною системою координат (рис. 2.3). Тут координата H визначається полярним кутом, S – радіус-вектором, а V – Z-координатою. Тобто відтінок змінюється під час руху вздовж кола циліндра, насиченість – вздовж радіуса, а яскравість – вздовж висоти.

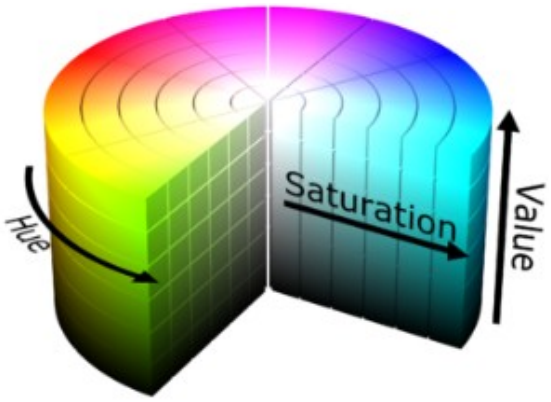


Рис. 2.3. Приклад формування HSV формату кольорового зображення

Процедура blank[:] = 0,255,0 дозволила заповнити всі значення змінної blank однаковими пікселями з кортежу (0, 255, 0). Можна змінити таку поведінку програми та вказати діапазон значень, які необхідно заповнити потрібними пікселями:

```
blank[200:300, 300:400] = 0,255,0
```

Така поведінка призведе до зафарбовування лише прямокутної ділянки, розміром 300 на 400 пікселів, у зелений колір з початком в координатах 200 пікселів по X та 300 пікселів по Y на загальному темному екрані монітора.

Щоб накреслити прямокутник з початку координат (0, 0) з висотою і шириною 250 пікселів, кольором граней – зеленим і товщиною ліній – 2 пікселі, застосуємо наступний код:

```
blank = np.zeros((500, 500, 3), dtype='uint8')
cv.rectangle(blank, (0,0), (255,255), (0, 250, 0), thickness=2)
cv.imshow('Rectangle', blank)
cv.waitKey(0)
```

Якщо параметру «thickness» передати ціле число, тоді цей параметр буде відтворювати товщину границі прямокутника, а якщо даному параметру передати значення cv.FILLE або -1, тоді прямокутник буде зафарбований повністю.

Для зміни кольору картинки з кольорового на сірий застосовуємо наступний код:

```
img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
cv.imshow("Cat2", img_gray)
```

Даний фрагмент коду демонструє, як OpenCV інтерпретує RGB формат. Як видно, палітра каналів має послідовність BGR.

Розглянемо наступні команди:

– `cv2.flip()` – перевертає 2D-масив навколо вертикальної, горизонтальної чи обох осей. Може набувати значень: 1 – переворот навколо осі y, 0 – переворот навколо осі z, –1 – переворот по обох осях;

– `b, g, r = cv2.split()` – розділити зображення за каналами;

– `cv2.merge()` – об’єднати канали зображення;

– `cv2.blur(image, (n, n))` – розмиття зображення на основі згортки з ядром n-n. Піксель у центрі матриці встановлюється рівним середньому значенню решти пікселів. Чим більше ядро згортки, тобто n-n, тим більше розмиття;

– `cv2.GaussianBlur(image, (n, n), s)` – розмиття за Гаусом. Схоже на попереднє розмиття, проте замість простого середнього використовується зважене, тобто чим ближче пікселі до

центрального, тим більший вплив вони мають на середнє значення (n – ядро згортки, s – стандартне відхилення ядра Гауса);

– `cv2.putText(image, text, org, font, fontScale, color[, thickness[, lineType]])` – накладення тексту на зображення (image – зображення; text – рядок тексту; org – координати нижнього лівого кута текстового рядка x та y; font – шрифт, наприклад, FONT_HERSHEY_SIMPLEX, FONT_HERSEY_PLAIN і т. д.; fontScale – масштабний коефіцієнт шрифту, який множиться на базовий розмір; color – колір рядка, thickness – товщина; lineType – необов'язковий параметр, який визначає тип лінії);

– `cv2.line((image, org, color[, thickness[, lineType[, shift]]])` – накреслити лінію на зображенні (image – зображення; text – рядок тексту; org – координати; color – колір; thickness – товщина; lineType – необов'язковий параметр, що визначає тип лінії; shift – колір);

– `cv2.threshold(image, thr, max, TypeThr)` – функція бінаризації. Для кожного пікселя застосовується те саме порогове значення (image – вихідне зображення, яке має бути зображенням у градаціях сірого; thr – це граничне значення, яке використовується для класифікації значень пікселів; max – це максимальне значення, яке присвоюється значенням пікселів, що перевищує граничне значення. OpenCV надає різні типи порогових значень, які задаються четвертим параметром функції: cv2.THRESH_BINARY, cv2.THRESH_BINARY_INV, cv2.THRESH_ TRUNC, cv2.THRESH_TOZERO, cv2.THRESH_TOZERO_INV);

– `cv2.findContours(image, mode, method)` – функція виділення контурів на зображенні (image – вихідне зображення; mode – один із чотирьох режимів групування знайдених контурів: CV_RETR_LIST – видає всі контури без групування; CV_RETR_EXTERNAL – видає лише крайні зовнішні контури; CV_RETR_CCOMP – групує контури у дворівневу ієрархію. На верхньому рівні – зовнішні контури об'єкта. На другому рівні – контури отворів, якщо такі є. Решта контурів потрапляють на верхній рівень; CV_RETR_TREE – групує контури у багаторівневу ієрархію; method – один із трьох методів упаковки контурів:

CV_CHAIN_APPROX_NONE – упаковка відсутня та всі контури зберігаються у вигляді відрізків, що складаються з двох пікселів; CV_CHAIN_APPROX_SIMPLE – склеює всі горизонтальні, вертикальні та діагональні контури; CV_CHAIN_APPROX_TC89_L1, CV_CHAIN_APPROX_TC89_KCOS – застосовує до контурів метод упаковки (апроксимації) Teh-Chin);

– `cv2.Canny(image, T_lower, T_upper, aperture_size, L2Gradient)` – фільтр, який використовується для виявлення країв зображення (`image` - зображення, до якого буде застосовано фільтр Canny; T_lower – нижнє порогове значення в гістерезисі; T_upper – верхнє порогове значення в гістерезисі; aperture_size – розмір діафрагми фільтра Sobel; L2Gradient – логічний параметр, який використовується для більшої точності під час обчислення Edge Gradient);

– `cv2.createCLAHE()` – алгоритм адаптивної корекції гістограм з обмеженим контрастом. Зображення поділяється на невеликі блоки і гістограма вирівнюється для кожного з них, після чого для об'єднання кожного блоку застосовується білінійна інтерполяція для усунення штучних меж. Застосовується для покращення контрастності зображень. Має два параметри: clipLimit (встановлює поріг для обмеження контрасту, за замовчуванням 40) та tileGridSize (встановлює кількість плиток у рядку та стовпці, за замовчуванням 8×8).

ПОСЛІДОВНІСТЬ ВИКОНАННЯ РОБОТИ

1. Ознайомитися з основними функціями бібліотеки OpenCV.
2. Завантажити файли з директорії https://bit.ly/41YtoKW та дослідити наступний код фільтрації зображень:

```
import cv2 as cv
import numpy as np

def task_1():
    image_filename = r'week1/img/hearts 1.png'
    img = cv.imread(image_filename)
    cv.imshow('Origin_1', img)
    cv.waitKey(0)
```

```
    th, img = cv.threshold(img, 17, 255, cv.THRESH_BINARY)
    cv.imshow('Processed_1', img)
    cv.waitKey(0)
def task_2():
    image_filename = r'week1/img/hearts 2.png'
    img = cv.imread(image_filename, 0)
    cv.imshow('Origin_2', img)
    cv.waitKey(0)
    img  =  cv.adaptiveThreshold(img,
    255,
    cv.ADAPTIVE_THRESH_GAUSSIAN_C,
    cv.THRESH_BINARY, 125, 2.8)
    img = cv.morphologyEx(img, cv.MORPH_CLOSE, np.ones( (5,
    5), dtype=np.uint8) )
    cv.imshow('Processed_2', img)
    cv.waitKey(0)
def task_3():
    image_filename = r'week1/img/hearts 3.png'
    img = cv.imread(image_filename)
    cv.imshow('Origin_3', img)
    cv.waitKey(0)
    cv.imshow('Origin_3', img[:,:,1])
    cv.waitKey(0)
    th, img = cv.threshold(img[:, :, 1], 17,
    255, cv.THRESH_BINARY+cv.THRESH_OTSU)
    cv.imshow('Processed_3', img)
    cv.waitKey(0)
def task_4():
    image_filename = r'week1/img/hearts 4.png'
    img = cv.imread(image_filename)
    cv.imshow('Origin_4', img)
    cv.waitKey(0)
    img_hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)
    cv.imshow('Intermedite_4 ', img_hsv[:,:,0])
    cv.waitKey(0)
    img = 255 - cv.inRange(img_hsv[:,:,0], 25, 65)
    cv.imshow('Processed_4', img)
    cv.waitKey(0)
def task_5():
    image_filename = r'week1/img/hearts 5.png'
    img = cv.imread(image_filename, 1)
```

```
    cv.imshow('Origin_5', img)
    cv.waitKey(0)
    kernel = np.ones((3,3), dtype=np.uint8)
    kernel[0, [0, -1]] = 0
    kernel[-1, [0, -1]] = 0
    img = cv.dilate(img, kernel)
    cv.imshow('Intermedite_5', img)
    cv.waitKey(0)
    img = np.mean(img, axis=2).astype(np.uint8)
    img = cv.adaptiveThreshold(img, 255,
cv.ADAPTIVE_THRESH_MEAN_C, cv.THRESH_BINARY, 155, 1.8)
    kernel = np.ones((5,5), dtype=np.uint8)
    kernel[0, [0, -1]] = 0
    kernel[-1, [0, -1]] = 0
    img = cv.dilate(img, kernel)
    img = cv.morphologyEx(img, cv.MORPH_CLOSE, kernel)
    cv.imshow('Processed_5', img)
    cv.waitKey(0)
def task_6():
    image_filename = r'week1/img/hearts 6.png'
    img = cv.imread(image_filename)
    cv.imshow('Origin_6', img)
    cv.waitKey(0)
    img = cv.dilate(img, np.ones((3,3), dtype=np.uint8))
    img = np.mean(img, axis=2).astype(np.uint8)
    img = cv.adaptiveThreshold(img, 255,
cv.ADAPTIVE_THRESH_MEAN_C, cv.THRESH_BINARY, 155, 2.5)
    img = cv.morphologyEx(img, cv.MORPH_CLOSE,
np.ones((9,9), dtype=np.uint8))
    cv.imshow('Processed_6', img)
    cv.waitKey(0)
def task_7():
    image_filename = r'week1/img/hearts 7.png'
    img = cv.imread(image_filename)
    cv.imshow('Origin_7', img)
    cv.waitKey(0)
    img = cv.dilate(img, np.ones((3,3), dtype=np.uint8))
    img = np.mean(img, axis=2).astype(np.uint8)
    img = cv.adaptiveThreshold(img, 255,
cv.ADAPTIVE_THRESH_MEAN_C, cv.THRESH_BINARY, 155, 2)
```

```
    img = cv.morphologyEx(img, cv.MORPH_CLOSE,
np.ones((9,9), dtype=np.uint8))
    cv.imshow('Processed_7', img)
                                                cv.waitKey(0)
def task_8():
    image_filename = r'week1/img/hearts 8.png'
    img = cv.imread(image_filename)
    cv.imshow('Origin_8', img)
    cv.waitKey(0)
    img = np.mean(img, axis=2).astype(np.uint8)
    img = cv.dilate(img, np.ones((5, 5), dtype=np.uint8),
iteration=3)
    img = cv.erode(img, np.ones((5, 5), dtype=np.uint8),
iteration=3)
    cv.imshow('Origin', img)
    cv.waitKey(0)
    img = cv.adaptiveThreshold(img, 255,
cv.ADAPTIVE_THRESH_MEAN_C, cv.THRESH_BINARY, 155, 5.0)
    cv.imshow('Processed_8', img)
    cv.waitKey(0)
def task_9():
    image_filename = r'week1/img/hearts 9.png'
    img = cv.imread(image_filename)
    cv.imshow('Origin_9', img)
    cv.waitKey(0)
    img = np.mean(img, axis=2).astype(np.uint8)
    img = cv.threshold(img, 254, 255, cv.THRESH_BINARY_INV)
[1]
    cv.imshow('Processed_9', img)
    cv.waitKey(0)
def task_10():
    image_filename = r'week1/img/hearts 10.png'
    img = cv.imread(image_filename)
    cv.imshow('Origin_10', img)
    cv.waitKey(0)
    img = np.absdiff(img[:,1:,:], img[:, :-1,:])
    cv.imshow('Processed_10', img)
    cv.waitKey(0)
if __name__ == '__main__':
    task_1()
```

3. Надати коментарі до коду вище та зробити висновки про його роботу.

4. Дослідити інструменти з бібліотеки dlib: get_frontal_face_detector() – функція для детектування осіб. Для виявлення осіб dlib використовує методи HOG (Histogram of Oriented Gradients) та SVM (Support Vector Mashines); shape_predictor() – являє собою інструмент, який вибирає область зображення, що містить певний об'єкт та виводить набір точок, що визначають положення об'єкта. В якості аргументу необхідно вказати використовувану модель shape_predictor_68_face_landmarks.dat.

5. Для роботи з бібліотекою детектування обличчя на зображенні потрібно встановити модуль dlib. Оскільки під час встановлення даного модуля можуть виникати труднощі, пропонується виконати дану частину роботи в Google Colab, де даний модуль є встановленим за замовчуванням. Посилання на підготовлену лабораторію: https://colab.research.google.com/drive/1yQhRID8i24TRdJUKwwKSsLOf4Rlb2QWA?usp=sharing

6. Приклад фрагмента коду python для детектування особливих точок особи з використанням бібліотеки dlib:

```
#імпорт бібліотеки dlib
import dlib
```

Вибір детекторів:

```
# детектор обличчя на зображенні
detector = dlib.get_frontal_face_detector()
```

Функція detector вміє автоматично визначати координати границь обличчя на зображенні, а для функції predictor потрібна тестова навчальна вибірка, наприклад, «shape_predictor_68_face_landmarks.dat», яка дозволяє визначати 68 точок контуру обличчя на зображенні. Вказана тестова вибірка є стандартною вибірною від розробників бібліотеки dlib і завантажується окремо.

Визначення контуру обличчя на фотографії:

```
image = cv2.imread(file)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
faces = detector(image_rgb)
for face in faces:
  x1 = face.left()
  y1 = face.top()
  x2 = face.right()
  y2 = face.bottom()
   cv2.rectangle(image_rgb, (x1, y1), (x2, y2), (0, 255,
0), 3)
plt.imshow(image_rgb)
plt.show()

# Шлях для збереження завантаженого файлу
model_path = 'shape_predictor_68_face_landmarks.dat.bz2'

# Завантаження моделі
if not os.path.isfile(model_path):
    import urllib.request
    url =
'http://dlib.net/files/shape_predictor_68_face_landmarks.
dat.bz2'
    urllib.request.urlretrieve(url, model_path)
# Розпакування моделі
model_unzipped_path =
'shape_predictor_68_face_landmarks.dat'
if not os.path.isfile(model_unzipped_path):
  with bz2.open(model_path, 'rb') as source,
open(model_unzipped_path, 'wb') as dest:
  dest.write(source.read())
```

Приклад коду на мові Python для пошуку та відображення на зображенні особливих точок обличчя:

```
import cv2
from google.colab.patches import cv2_imshow

# Код для завантаження та обробки зображення
```

```
# Детектування обличчя
faces = detector(image_rgb)
# Перебір результатів детектування
for face in faces:
    x, y, w, h = face.left(), face.top(), face.width(),
face.height()
    # Визначення ключових точок обличчя
    landmarks = predictor(image_rgb, face)
    # Перебір ключових точок і малювання їх на
зображенні
    for i in range(68):  # 68 - кількість ключових точок
для 68-ти точкової моделі
        x_point = landmarks.part(i).x
        y_point = landmarks.part(i).y
        cv2.circle(image, (x_point, y_point), 1, (0,
255, 0), -1)  # Малюємо точку зеленим колом

# Відображення зображення з виділеними ключовими точками
обличчя
cv2_imshow(image)  # Використовуйте cv2.imshow якщо не
використовуєте Google Colab
cv2.waitKey(0)
cv2.destroyAllWindows()
```

7. Дати відповідь на запитання:

– що таке піксель?

– якими параметрами описується піксель?

– що таке формат RGB і які альтернативи існують?

– як зображення зберігається в електронному вигляді?

– яким чином для зображення встановлюють колір?

– які формати для зберігання зображень існують?

– як кольорове зображення перетворити на сіре?

– що вібудеться із зображення, якщо в кожному з його пікселів змінити значення червоного каналу на 1?

– що означає термін «вирівнювання кольору»?

ЗАВДАННЯ № 3. Дослідити IoT та елементи Smart Home

Мета роботи: ознайомитися з ключовими термінами IoT («Інтернет речей») та основами розробки технологій IoT.

Основні теоретичні відомості

Інтернет речей (Internet of Things, скорочено IoT) – це глобальна мережа підключених до Інтернету речей пристроїв, оснащених сенсорами, датчиками, засобами передавання сигналів. Ці цифрові пристрої можуть сприймати датчиками різноманітні сигнали з навколишнього світу, вступати у взаємодію з іншими пристроями, обмінюватися даними з метою віддаленого моніторингу за станом об’єктів, аналізу зібраних даних і прийняття на їх основі рішень.

Інтернет речей об’єднує реальні речі в віртуальні системи, здатні вирішувати абсолютно різні завдання. Ключова ідея — з’єднати між собою всі об’єкти, які можна з’єднати, підключити їх до мережі для збирання даних і прийняття рішень на їх основі. Наприклад, відкрити гаражні двері, включити кавоварку або кондиціонер, виключити світло тощо. IoT з технологічної точки зору – це, по суті, мережа мереж, що складається з унікально ідентифікованих об’єктів (по факту «речей»), які можуть взаємодіяти між собою через IP-підключення без втручання людини.

Smart технологія – це процес взаємодії об’єктів з оточуючим середовищем, що наділяє цю систему здатністю адаптації до нових умов, саморозвитку та самонавчання, ефективного досягнення цілей. На основі розвитку Smart-технологій останнім часом стали виникати нові поняття: Smart-міста, Smart-країни, Smart-освіта, Smart-економіка, і це найближчим часом призведе до створення Smart-суспільства. В основі цього «розумного суспільства» лежить розвиток «суспільства знань», цифрових технологій, усього того, що приведе до цифрової ери розвитку цивілізації.

Ключовими поняття IoT є:

- «Інтернет речей» представляє мережу зв’язаних через інтернет об’єктів, здатних збирати дані і обмінюватися даними, які надходять із вбудованих сервісів.
- «Пристрої IoT» входять до системи інтернету речей і представляють будь-які автономні пристрої, підключені до Інтернету,

якими можна керувати дистанційно.

- «Екосистема IoT» включає всі компоненти, які дозволяють бізнесу, урядам і користувачам приєднувати свої пристрої IoT, включаючи пульти управління, панелі інструментів, мережі, шлюзи, аналітику, зберігання даних і безпеку.
- «Фізичний рівень» представляє апаратне забезпечення, яке використовується в IoT пристроях, включаючи сенсори та мережеве обладнання. Відповідає за передачу даних, зібраних у фізичному шарі, до різних пристроїв.
- «Рівень додатків» включає протоколи та інтерфейси, які використовують пристрої для ідентифікації та зв'язку між собою.
- «Пульти управління» дозволяють людям використовувати IoT-пристрої, з'єднуючись з ними і контролюючи їх за допомогою панелі інструментів – наприклад, за допомогою мобільних додатків. До пультів управління відносяться смартфони, планшети, ПК, розумні годинники, телевізори і нетрадиційні пульти.
- «Панелі інструментів» забезпечують відображення інформації про екосистему IoT для користувачів, дозволяючи нею керувати (як правило, дистанційно).
- «Аналітичний фактор» представляє програмні системи, які аналізують дані, отримані від IoT-пристроїв. Аналітика використовується у великій кількості сценаріїв – наприклад, для прогнозування технічного обслуговування.

Промисловий Інтернет речей (Industrial IoT, IIoT) – це один з найбільш великих сегментів Інтернету речей з точки зору кількості підключених пристроїв і ступеня корисності цих сервісів для виробництва і автоматизації підприємств. Цей сегмент традиційно служить операційно-технологічною базою. Сюди входять апаратні і програмні засоби моніторингу фізичних пристроїв. Традиційні завдання інформаційних технологій вирішуються інакше, ніж операційно-технологічні завдання. Операційні технології (OT) зосереджені на оцінці продуктивності, часу безвідмовної роботи, зборі даних і відповідної реакції в режимі реального часу, а також безпеки систем. Інформаційні технології спрямовані на безпеку, групування, сервіси та надання даних.

До екосистеми Інтернету речей відносяться усі засоби, сервіси і технології, які використовуються в Інтернеті речей (рис. 3.1), зокрема:

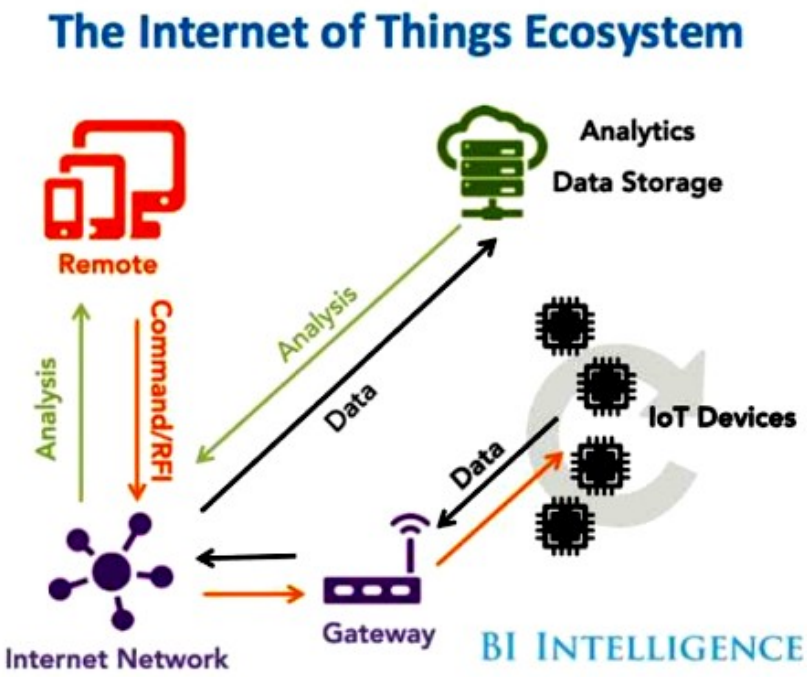


Рис. 3.1. Екосистема Інтернету речей

- sensors (розумні датчики/виконавчі механізми): вбудовані системи, операційні системи реального часу, джерела безперебійного живлення, мікро-електромеханічні системи (МЕМС);
- системи зв'язку з датчиками: зона охоплення бездротових персональних мереж становить від 0 см до 100 м. Для обміну даними між датчиками застосовуються низькошвидкісні малопотужні інформаційні канали, які часто побудовані не на протоколі IP;
- локальні обчислювальні мережі (LAN): зазвичай це системи обміну даними на основі протоколу IP, наприклад, 802.11 Wi-Fi-мережу для швидкого радіозв'язку, часто це пірингові або зіркоподібні мережі;
- агрегатори, маршрутизатори (routers), шлюзи (gateways), пограничні пристрої (Edge Device): постачальники вбудованих систем, самі бюджетні складові (процесори, динамічна оперативна пам'ять і система зберігання даних), виробники модулів, виробники пасивних компонентів, виробники тонких клієнтів, виробники стільникових і бездротових радіосистем, постачальники міжплатформового програмного забезпечення, розробники інфраструктури туманних

обчислень, інструментарій для граничної аналітики, безпеку граничних пристроїв, системи управління сертифікатами;

- глобальна обчислювальна мережа: оператори стільникового зв'язку, оператори супутникового зв'язку, оператори малопотужних глобальних мереж (Low- Power Wide-Area Network, LPWAN). Зазвичай застосовуються транспортні протоколи Інтернету для IoT і мережевих пристроїв (MQTT, CoAP і навіть HTTP);
- хмара: інфраструктура в якості постачальника послуг, платформа в якості постачальника послуг, розробники баз даних, постачальники послуг потокової і пакетної обробки даних, інструменти для аналізу даних, програмне забезпечення в якості постачальника послуг, постачальники озер даних, оператори програмно-визначених мереж / програмно-визначених периметрів, сервіси машинного навчання;
- сервіси аналізу даних: величезні масиви інформації передаються в хмару. Робота з великими обсягами даних і отримання з них користі – це завдання, що вимагає комплексної обробки подій, аналітики і прийомів машинного навчання;
- безпека (security): під час зведення всіх елементів архітектури воєдино постають питання кібербезпеки. Безпека стосується кожного компонента: від датчиків фізичних величин до ЦПУ і цифрового апаратного забезпечення, систем радіозв'язку і самих протоколів передачі даних. На кожному рівні необхідно забезпечити безпеку, достовірність і цілісність. У цьому ланцюзі не повинно бути слабких ланок, оскільки Інтернет речей стане головною мішенню для атак хакерів в світі.

Всі датчики і пристрої IoT повинні бути пов'язані в єдину мережу для обміну інформацією між собою. Рішення про вибір типу зв'язку ґрунтується на вирішенні конкретного завдання.

Основними технологіями комунікації між пристроями є:

- Wi-fi – одна з найпоширеніших бездротових технологій передачі даних. Відповідає стандарту IEEE 802.11 (n / b / g) і функціонує на частотах 2,4 ГГц і 5 ГГц, залежно від використаного обладнання. Забезпечує високу швидкість передачі даних, але має ряд недоліків: високе енергоспоживання; в діапазоні 2,4 ГГц безліч пристроїв і пересічних технологій (Bluetooth); різні експлуатаційні

обмеження для різних країн; слабка стійкість до взлому стандарту шифрування WEP (нові стандарти WPA і WPA2 мають більшу надійність).

- Bluetooth дозволяє обмінюватися інформацією пристроїв, що знаходяться в радіусі 10 метрів (нові версії пристроїв працюють на відстані до 16 метрів). Має істотний недолік в безпеці, а також на якість зв'язку сильно впливають зовнішні фактори. До переваг можна віднести низьке енергоспоживання і постійне поліпшення стандарту.
- Технології стільникового зв'язку (2G, 3G, 4G, 5G) мають велике покриття і досить високу швидкість передачі даних (для мереж з технологією 3G/4G) і дозволяє охопити більшу територію під час реалізації проєктів без необхідності пошуку окремої точки доступу до мережі. Основною проблемою вважається обов'язкова прив'язка до стільникового оператора, можливе навантаження на вишки стільникового оператора (або їхня недоступність), високе енергоспоживання під час передачі даних (особливо у разі слабкого покриття мережі).
- LoRaWan – відкритий протокол для високоємних (до 1 млн пристроїв в одній мережі) мереж з великим радіусом дії і низьким енергоспоживанням. Протокол забезпечує двосторонній зв'язок з шифруванням для всіх класів пристроїв. Архітектура протоколу розроблялася в тому числі і для того, щоб легко знайти мобільні об'єкти для відстеження пересувань. Це найбільш швидко зростаючий напрям додатків Інтернету речей. Основним недоліком є швидкість передачі даних (до 5 КБ/с), проте цього цілком достатньо для передачі даних з датчиків.
- Local Area Network (LAN) – комп'ютерна мережа, зазвичай покриває відносно невелику територію. Сполучення здійснюється за допомогою мережевого кабелю. Має високу швидкість передачі даних, високу стійкість до перешкод, певну адресацію всередині мережі. Недоліками є відсутність маршрутизаторів, фізичної доступності для підключення пристрою до мережі кабелем.

Наведені технології не є вичерпним описом всіх існуючих рішень, але є основними у роботі з інтернет-речами.

Оскільки архітектура IoT передбачає наявність таких функціональних рівнів, як мережа датчиків, шлюзів, керування,

додатків, тоді виникає необхідність в протоколах для забезпечення взаємодії цих пристроїв один з одним і верхніми рівнями. Стандартні прикладні протоколи не підходять через їхню непристосованість до умов мережі IoT. Наприклад, датчик зазвичай з невеликою пам'яттю і вимірює фізичні параметри в режимі реального часу, найчастіше в умовах низького енергозабезпечення. Результати вимірювань повинні оброблятися та передаватися на сервер, а обсяг інформації, що формується одним сенсорним вузлом, може бути як дуже маленьким, так і дуже великим. Для реалізації таких задач архітектура IoT повинна підтримувати парадигму багато джерел – багато одержувачів з динамічним масштабуванням трафіку даних.

Шаблон проєктування передачі повідомлень, який називається «видавець-передплатник» (publisher-subscriber або pub/sub), є одним з базових в топології мережі IoT (рис. 3.2).

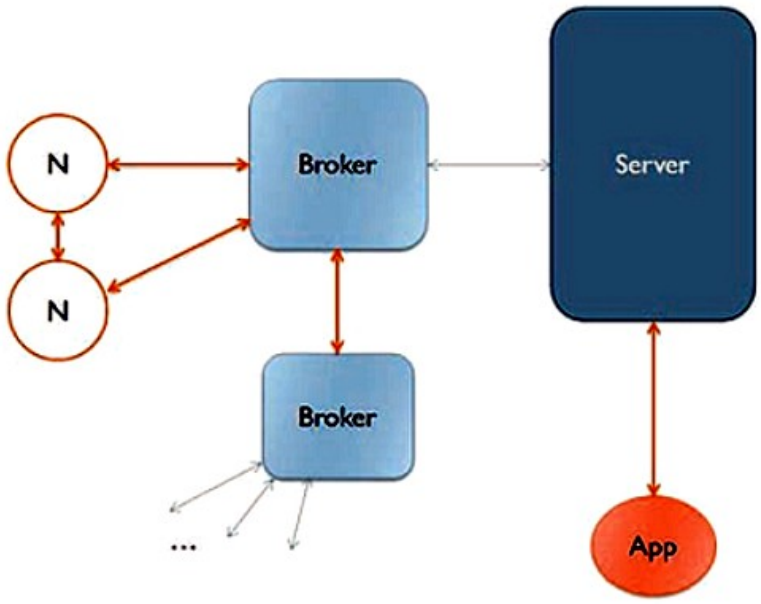


Рис. 3.2. Топологія publisher-subscriber

В такій схемі вводиться поняття видавця – джерела інформації та передплатника – її отримувача. Термін «підписка» пов'язаний з певною процедурою, яку використовують учасники шаблону, для надання інформації передплатником від конкретного видавця, а також упорядкування збору інформації – параметрів швидкості тощо. У цьому випадку розглядається ситуація, коли сенсорний вузол (на рис. 3.2 – Node) генерує інформацію з кількох датчиків і направляє її відповідно до параметрів підписки або за запитом, або самостійно з

певним інтервалом або після якоїсь події на сервері. Оскільки датчики зазвичай досить примітивні, завдання зводяться до постійної передачі інформації по контрольованому параметру, тому з'являється необхідність об'єднувати датчики у вузли, оснащені мікроконтролерами, які відповідають за зчитування даних і надсилання їх за заданими визначеними алгоритмами далі на сервері. Також для взаємодії клієнта із системою необхідний клієнтський додаток (на рис. 3.2 – Application), який встановлений на персональному пристрої. Така топологія також розрахована на підключення брокера – це сервер, який отримує інформацію від видавців і передає її відповідним передплатникам, а в складних системах може виконувати різні дії, пов'язані з аналізом та обробкою даних, які на його надійшли. Брокер може встановлювати пріоритети повідомлень і формує черговість передачі повідомлень. Таким чином брокер організовує реєстр повідомлень, їх зберігання та фільтрацію. У разі недостатнього ресурсу канал зв'язку або якщо одержувач недоступний під час надсилання даних, дані повинні бути збережені доти, доки вони не будуть отримані.

На ділянці мережі між сенсорними вузлами виконуватися ряд завдань, наприклад, розподіл інформації між сенсорними вузлами для тимчасового зберігання або перенаправлення. Для забезпечення зв'язку між сенсорними вузлами/датчиками використовується протокол DDS (Data Distribution Service) (рис. 3.3).

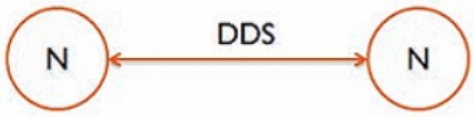


Рис. 3.3. Протокол DDS між вузлами (Node)

Протокол DDS розподіляє дані між пристроями і реалізує прямий шинний зв'язок між пристроями з урахуванням даних реляційної моделі. Протокол DDS реалізує багатоадресну систему за допомогою UDP. Цей протокол орієнтований на шаблон «видавець–передплатник», під час цієї передачі повідомлення створено по шині з використанням методу «запит–відповідь». Процедури, що виконуються протоколом, задаються трьома класами (Entity Class, WaitSet Class, Condition Class, Publisher Class, DataWriter Class, Subscriber Class, DataReader Class, ReadCondition Class, QueryCondition

Class та інші). Протокол DDS реалізує дві процедури – читання та запис, використовуючи відповідні класи.

Процедура читання (Read) запускається на всіх доступних пристроях. Дані не видаляються з локального кеша DDS в результаті цієї процедури і можуть бути знову прочитані за вказівкою спеціальних параметрів. Отримання даних виконується трьома способами:

- опитування – програма (зазвичай тривало) запитує DDS для отримання нових даних або інформації про зміну стану. Інтервал опитування залежить від програми та даних;
- списки очікування (WaitSets) – додаток реєструє в DDS списки очікування та чекає доти, доки одна з переданих подій не відбудеться;
- слухачі (Listeners) – додаток реєструє в DDS (у класах, де описані події) спеціальні класи-слухачі, які будуть інформовані під час цих подій.

На ділянці Node – Broker реалізується кілька завдань, наприклад, таких як реєстрація сенсорного вузла, конфігурація і налаштування вузлів, передача та розподіл інформації (рис. 3.4). У цьому сегменті мережі можуть використовуватися два наступні протоколи.

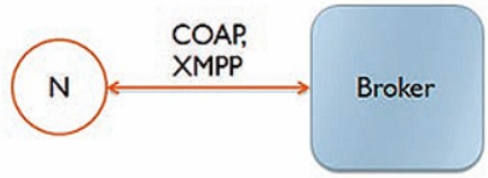


Рис. 3.4. Сегмент мережі від вузла (Node) до брокера (Broker)

В невеликих персональних мережах використовується протокол XMPP (Extensible Messaging and Presence Protocol) – масштабований протокол обміну повідомленнями та інформацією про присутність. XMPP забезпечує простий спосіб адресації пристроїв. Для ідентифікації користувачів використовують ідентифікатори JID. У протоколі XMPP використовується текстовий формат XML. Як транспорт використовується протокол TCP. XMPP підтримує різні комунікаційні моделі (запит-відповідь, публікація-підписка та інші). Адресація XMPP особливо зручна у випадку, коли дані передаються між віддаленими, потенційно незалежними точками, наприклад, у разі

двох взаємодії абонентів. За допомогою XMPP, наприклад, можливе підключення домашнього термостата до веб-сервера для отримання доступу до нього з телефона. Сильними сторонами цього протоколу є його безпека і масштабованість.

Для мереж з обмеженими ресурсами та низьким енергоспоживанням рекомендують протокол COAP (Constrained Application Protocol) – це спеціалізований протокол передачі, створений для мережі та пристроїв з обмеженими ресурсами, M2M-додатків тощо. COAP можна розглядати як додаток до HTTP, але, на відміну від HTTP, COAP орієнтований на використання у пристроях з певними обмеженнями. COAP використовує транспортний протокол UDP. Основними видами запитів, які використовує протокол COAP є запити-відповіді: GET, PUT, HEAD, POST, DELETE, CONNECT.

На ділянці між брокером та сервером відбувається збір та агрегація даних; організація пакетів даних та їх черговість, розподіл та зберігання інформації. MQTT є найпоширенішим протоколом зв'язку в IoT між брокером і сервером. Взагалі в IoT використовується безліч протоколів на базі TCP/IP, але MQTT є, де-факто, стандартом галузі, бо був розроблений саме для використання в Інтернеті речей. Ключовими поняттями для MQTT є брокер та клієнт. В MQTT обмін даними між пристроями виконується за принципом «видавець–передплатник», що дозволяє пристроям надсилати та отримувати дані у процесі виникнення певної події. MQTT – бінарний протокол обміну повідомленнями з використанням TCP.

Для мереж, які використовують обладнання різних платформ і допускають застосування простого протоколу надсилання повідомлень, можна використовувати STOMP – Simple (або Streaming) Text Oriented Message Protocol – простий протокол обміну повідомленнями, що забезпечує взаємодію з багатьма мовами, платформами та брокерами. Цей протокол підходить під шаблон «видавець–передплатник» та за допомогою повідомлень SEND, SUBSCRIBE, UNSUBSCRIBE, BEGIN, COMMIT, ABORT, ACK, NACK, DISCONNECT організовує зв'язок з брокером за методом «запит–відповідь».

На ділянці між сервером і додатком користувача вирішуються завдання, пов'язані із взаємодією користувача та системи: отримання

інформації з сервера (можливо за участю сервера-посередника, вся інформація може бути розподілена), конфігурація параметрів користувача (частоти отримання інформації, активація/дезактивація датчиків та вузлів тощо) та ін.

Для розподіленого обчислювального середовища, для веб-сервісів використовується протокол SOAP, бо в ньому реалізовано механізм доступу RPC (Remote Procedure Call), який відповідає віддаленому виклику функцій.

SOAP (Simple Object Access Protocol) – протокол обміну структурованими та довільними повідомленнями у форматі XML у розподіленому обчислювальному середовищі. SOAP використовує базову модель з'єднання, що забезпечує узгоджену передачу повідомлень від відправника до одержувача, який допускає наявність посередників, що можуть обробляти частину повідомлень або додавати до нього додаткові елементи. Може бути застосований в архітектурах, де потрібні складні запити. SOAP підтримує два механізми доступу – SOAP RPC та SOAP Message. SOAP RPC є простим протоколом «запит–відповідь», який базується на об'єкті Call. Цей об'єкт (і деякі низькорівневі методи для створення та надсилання повідомлень) використовується для синхронного віддаленого виклику процедури за допомогою XML. SOAP Message – це протокол для надсилання та обробки SOAP-повідомлень, який можна використовувати для асинхронних комунікацій, та має на увазі негайну або відкладену відповідь на запит. Протокол SOAP Message базується на об'єкті Message.

SOAP завдяки декількох запитів (Get; SOAPAction, SOAPAction-Response), які мають структуру запит-відповідь, протокол може використовуватися з будь-яким протоколом прикладного рівня: SMTP, FTP, HTTP, HTTPS.

Найбільш поширеним контролером, який використовують для реалізації простих систем IoT, є ESP32 (рис. 3.5).

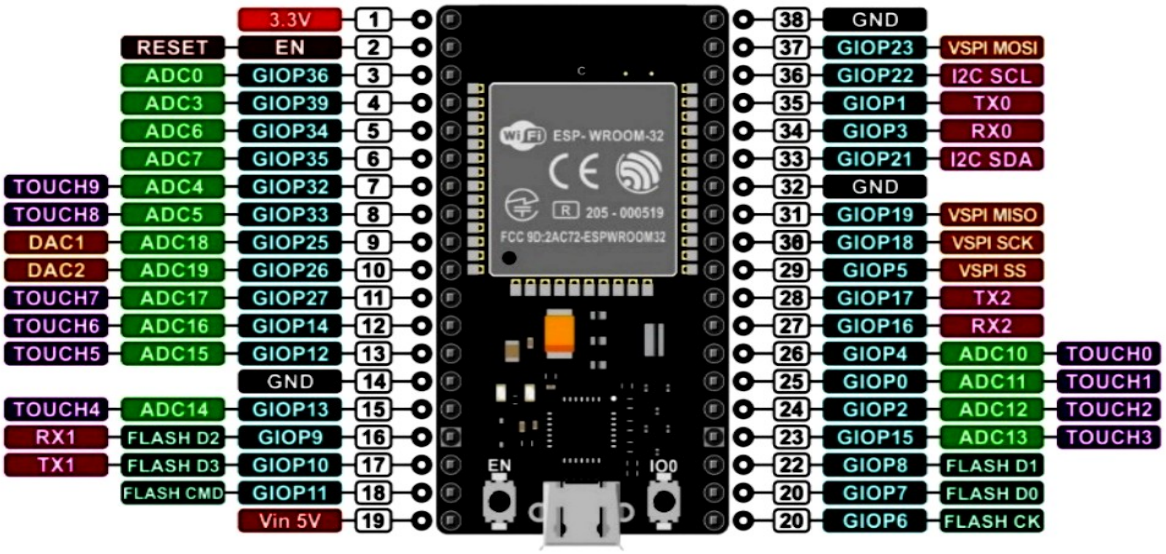


Рис. 3.5. Розташування та призначення входів/виходів ESP32

Цей контролер має 36 вбудованих входів/виходів загального призначення (GPIO). Всі GPIO можуть бути налаштованими як для читання цифрового сигналу, так і для його генерації. Деякі з них мають додаткові функції, наприклад, підключені до вбудованого АЦП або до апаратних інтерфейсів передачі даних (I2C, UART, SPI), чи мають можливість використовуватися як сенсорні елементи. Також наявні службові піни для виводу живлення та заземлення. При цьому частина GPIO мають обмеження у використанні. GPIO з 34 по 39 можуть працювати лише на вхід, а використання GPIO з 6 по 11 і 0, 1 та 3 рекомендується лише за їхнім призначенням, яке прописане у документації, тому що вони підключені безпосередньо до пам’яті модуля і можуть викликати нестабільність його роботи. Робота з цифровими входами/виходами найпростіша, тому з неї і починається ознайомлення з функціоналом модуля.

Розглянемо реалізацію програми, яка буде вмикати/вимикати світлодіод, який підключено до піна 14 через струмообмежувальний резистор номіналом 100 Ом.

Код на мові С: (https://wokwi.com/projects/363890819796403201)

```
#define LED 14
int TIME_DELAY = 500;
void setup() {
  Serial.begin(115200);
  Serial.println("Hello, ESP32!");
  pinMode(LED, OUTPUT);
}
```

```
void loop() {
  digitalWrite(LED, HIGH);
  delay(TIME_DELAY);
  digitalWrite(LED, LOW);
  delay(TIME_DELAY);
}
```

Код на мові Python (https://wokwi.com/projects/363893376952571905)

```
from machine import Pin
from time import sleep
ledPin = 14
TIME_DELAY = 2
led = Pin(ledPin, Pin.OUT)
print("Hello, ESP32!")
while True:
    led.value(1)
    sleep(TIME_DELAY)
    led.value(0)
    sleep(TIME_DELAY)
```

Дослідимо особливості реалізації системи моніторингу температури та вологості повітря на модулі IoT типу ESP32, датчику температури та вологості DHT22 (рис. 3.6) та хмарного сервісу для моніторингу даних hivemq.com. Виконаємо прототипування даного проєкту у симуляторі Wokwi. Створюємо новий проєкт з модулем ESP32, додаємо датчик температури, приєднуємо його до D15 (GPIO15) ESP32. З лівої сторони вікна проєкту додаємо код бібліотеки для DHT22, ініціалізуємо датчик та зчитуємо значення температури та вологості:

https://wokwi.com/projects/322577683855704658

```
import network
import time
from machine import Pin
import dht
import ujson
from umqtt.simple import MQTTClient
# MQTT Server Parameters
MQTT_CLIENT_ID = "micropython-weather-demo"
MQTT_BROKER    = "broker.mqttdashboard.com"
MQTT_USER      = ""
MQTT_PASSWORD  = ""
MQTT_TOPIC     = "wokwi-weather"
sensor = dht.DHT22(Pin(15))
print("Connecting to WiFi", end="")
```

```
sta_if = network.WLAN(network.STA_IF)
sta_if.active(True)
sta_if.connect('Wokwi-GUEST', '')
while not sta_if.isconnected():
  print(".", end="")
  time.sleep(0.1)
print(" Connected!")
print("Connecting to MQTT server... ", end="")
client = MQTTClient(MQTT_CLIENT_ID, MQTT_BROKER,
user=MQTT_USER, password=MQTT_PASSWORD)
client.connect()
print("Connected!")
prev_weather = ""
while True:
  print("Measuring weather conditions... ", end="")
  sensor.measure()
  message = ujson.dumps({
    "temp": sensor.temperature(),
    "humidity": sensor.humidity(),  })
  if message != prev_weather:
    print("Updated!")
              print("Reporting   to   MQTT   topic   {}:
{}".format(MQTT_TOPIC, message))
    client.publish(MQTT_TOPIC, message)
    prev_weather = message
  else:
    print("No change")
  time.sleep(1)
```

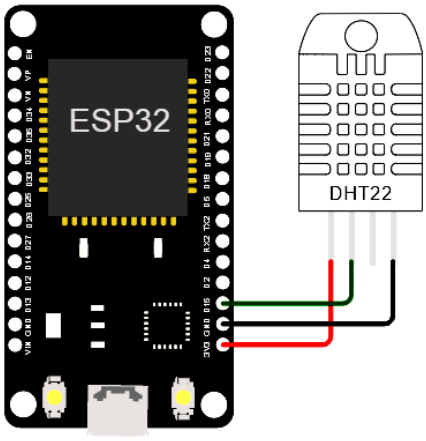


Рис. 3.6. Схема підключення датчика вологи

Проєкт управління модульним світлодіодом (рис. 3.7) також може бути реалізовано через ESP32 та віддалений сервер:
https://wokwi.com/projects/315787266233467457

```
import network
import ujson
```

```
from neopixel import NeoPixel
from machine import Pin
from time import sleep
from umqtt.simple import MQTTClient
pixels = NeoPixel(Pin(13), 16)
pixels.fill((0, 0, 0))
pixels.write()
def mqtt_message(topic, msg):
  print("Incoming message:", msg)
  try:
    msg = ujson.loads(msg)
    pixels.fill((msg[0], msg[1], msg[2]))
    pixels.write()
  except Exception as e:
    print("Error:", e)
print("Connecting to WiFi...", end="")
import network
wifi = network.WLAN(network.STA_IF)
wifi.active(True)
wifi.connect("Wokwi-GUEST", "")
while not wifi.isconnected():
  sleep(0.5)
  print(".", end="")
print("Done")
print("Connecting to MQTT...")
client = MQTTClient("wokwi1", "broker.hivemq.com")
client.set_callback(mqtt_message)
client.connect()
client.subscribe("wokwi")
print("Connected!")
while True:
  client.wait_msg()
```

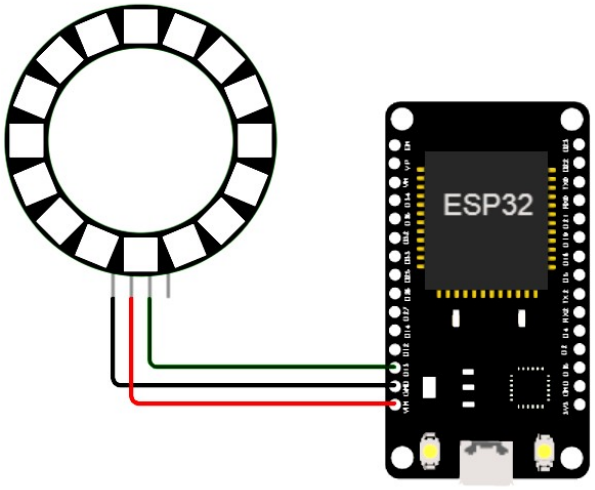


Рис. 3.7. Схема підключення модульного світлодіоду

ПОСЛІДОВНІСТЬ ВИКОНАННЯ РОБОТИ

1. Ознайомитися з основними термінами та поняттями технології IoT.

2. Проаналізувати код в даній роботі та зробити висновки щодо його роботи.

3. Спробувати реалізувати через сервіс Wokwi систему керування реле модулем ESP32 та брокером BLYNK.IO. Для прикладу можна використати наступний код на мові Сі для програмування ESP32 з підключення до сервісу BLYNK.IO. Схему підключення подано на рис. 3.8. Обов'язково потрібно буде підключити бібліотеку Blynk.

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>
#define BLYNK_TEMPLATE_ID "ВАШ ID З BLYNK.IO "
#define BLYNK_TEMPLATE_NAME "ВАША НАЗВА ПРОЕКТУ"
#define BLYNK_AUTH_TOKEN "ВАШ ТОКЕН З BLINK.IO"
char auth[] = "ВАШ ТОКЕН З BLINK.IO ";
char ssid[] = "Wokwi-GUEST";
char pass[] = "";
BlynkTimer timer;
#define RELEY1 12
#define RELEY2 14
int SW_state = 0; //kondisi lampu mati
BLYNK_WRITE(V1){
  SW_state = param.asInt();
  if(SW_state == 1){
    digitalWrite(RELEY1, HIGH);
    Serial.println("Lampu telah dihidupkan");
    Blynk.virtualWrite(V1, HIGH);
  }else{
    digitalWrite(RELEY1, LOW);
    Serial.println("Lampu telah dimatikan");
    Blynk.virtualWrite(V1, LOW);
  }
}
void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  pinMode(RELEY1, OUTPUT);
  Blynk.begin(auth, ssid, pass);
```

```
}
void loop() {
  // put your main code here, to run repeatedly:
  Blynk.run();
  timer.run();
}
```

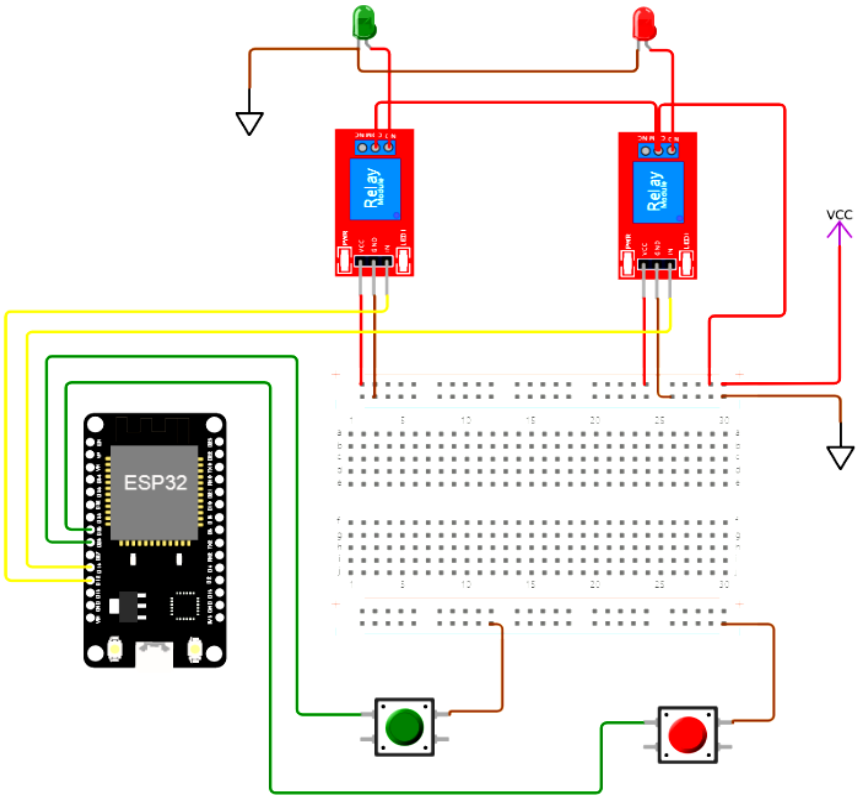


Рис. 3.8. Схема підключення реле до модуля ESP32

ЗАВДАННЯ № 4. Вивчення веб-фреймворку Django

Мета роботи: вивчення основних можливостей фреймворку Django для розробки веб-застосунків з архітектурою MVC та REST.

Основні теоретичні відомості

Django – це Python-фреймворк з відкритим кодом для швидкої розробки веб-систем з архітектурою, схожою на «Модель–Вид–Контролер» (MVC). Роль «контролера» класичної моделі MVC в Django виконує «вид» (view), який називається «шаблон» (template). Таким чином шаблон MVC розробники Django називають MTV («Модель Шаблон Вид»). Однією з основних переваг для розробника Django є відсутність потреби створювати контролери та сторінки для адміністративної частини сайту, оскільки в збірці є вбудований модуль для керування вмістом, який можна долучити до будь-якого сайту, написаного на Django, і якими можна керувати відразу декількома сайтами на одному сервері.

В Django є адміністративний модуль, який дає змогу створювати, змінювати і вилучати будь-які об’єкти наповнення сайту, фіксуючи всі дії, та надає інтерфейс для керування обліковими записами користувачів і групами (з призначенням прав). В Django також внесені засоби для системи коментарів і «статичні сторінки», які можна використовувати без необхідності писати додаткові контролери та відображення.

В Django зручна система шаблонів, яка передбачає наявність окремої мови для їх опису. Вона є достатньо простою, містить оператори циклу, умови, засоби форматування даних.

Мова шаблонів в Django виконує функцію відображення даних.

В Django є власний веб-сервер для розробки і налагоджування. Він автоматично відслідковує зміни у файлах програмного коду і перезапускається, що дуже зручно під час розробки проєкту.

Створимо новий каталог для проєкту Django та рекомендується активувати в створеному каталозі віртуальне середовище python. Далі встановимо Django через `pip python` в створеному віртуальному середовищі:

```
$ pip install django
```

Створимо новий проєкт Django:

```
$ django-admin startproject djangorest
```

Виконавши дану команду Django в поточному каталозі, створимо новий проєкт з назвою, яка була вказана в `django-admin`.

Переходимо в даний каталог проєкту та запускаємо створений проєкт:

```
$ python manage.py runserver
```

Вказана команда запускає тестовий локальний сервер за адресою http://127.0.0.1:8000.

Для виходу із серверного процесу потрібно виконати `Ctrl-C`.

Дослідимо структуру Django проєкту.

В каталі з проєктом буде наступний перелік файлів:

```
| - pythonprogect
| |- pyhonprogect (головний каталог проекту)
| |-|- - _pycache_
| | |- - _init_.py
| | |- - asgi.py
| | |- - settings.py
| | |- - urls.py
| | |- - wsgi.py
| |- manage.py
```

В головному каталозі проєкту знаходяться файли з основними налаштуваннями всього проєкту. У файлі `settings.py` розміщено конфігурації створеного проєкту, в `urls.py` – налаштування URL диспетчера. Файл `wsgi.py` – це службовий модуль, який робить аплікацією WSGI і виступає «посередником» між веб-сервером і проєктом.

Додаток Django входить до складу проєкту і реалізує функціональність одного з розділів проєкту Django. Кількість додатків в проєкті необмежена. Фізично додаток є пакетом, каталог якого знаходиться в каталозі проєкту поряд з головним каталогом. Ім’я пакета є ім’ям програми, а сам пакет називається пакетом додатку. Пакет додатку формується Django у процесі створення програми. Спочатку додаток містить наступні модулі:

– каталог `migrations`, де Django зберігає файли для відстеження змін, створених у файлі models.py, щоб підтримувати синхронізацію бази даних та моделей models.py;

– admin.py – це файл конфігурації для вбудованого застосунку Django під назвою Django Admin;

– apps.py – це файл конфігурації самого застосунку;

– models.py – це файл, де визначено об'єкти веб-застосунку. Django автоматично переводить моделі в таблиці бази даних;

– tests.py – це файл, який використовується для написання модульних тестів для застосунку;

– views.py – це файл, в якому оброблюється цикл запитів/відповідей веб-застосунку.

Щоб додаток успішно працював, потрібно, по-перше, виконати його прив'язку до інтернет-адреси, а по-друге, вказати його в списку активних додатків, що знаходиться в модулі settings головного пакета проєкту. Тільки після цього додаток стане активним.

Наприклад, створемо новий додаток:

```
$ python manage.py startapp hello_app
```

Виконання цієї команди створить додаток під назвою hello_blog. Щоб Django завантажив новий додаток, потрібно додати його назву в список Installed Apps в файлі settings.py:

```
# pythonprogect/settings.py
INSTALLED_APPS = [
'django.contrib.admin',
'django.contrib.auth',
'django.contrib.contenttypes',
'django.contrib.sessions',
'django.contrib.messages',
'django.contrib.staticfiles',
'hello_app'
]
```

Також можна замість 'hello_app' вказати повне ім'я конфігураційного файлу 'hello_app.apps.HelloAppConfig', тим самим вказавши посилання на автоматично створений клас HelloAppConfig з файлу apps.py, який знаходиться в створеній директорії додатка hello_app.

Частина допоміжних модулів Django реалізована також у вигляді додатків (вбудовані додатки). Таким вбудованим додатком є, зокрема, підсистема, що реалізує розмежування доступу. Щоб Django отримав доступ до створеного додатку hello_app, коли хтось відвідує URL

головної сторінки додатку, потрібно визначити URL, який надаватиме Django доступ до шаблону сторінки додатка hello_app. Для цього відредагуємо файл urls.py в головному каталозі проєкту. Це має виглядати наступним чином:

```
# pythonproget/urls.py
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
path('admin/', admin.site.urls),
path(r'hello/', include('hello_app.urls'))
]
```

Далі переходимо в каталог додатку (в нас це hello_app) та створюємо файл з назвою urls.py, де вказуємо простір URL-адрес для додатку hello_app, наприклад так:

```
# hello_app/urls.py
from django.urls import path
from . import views
urlpatterns = [
    path("", views.index, name="index"),
    path("brian", views.brian, name="brian"),
    path("<str:name>", views.greet, name="greet"),
]
```

В даному прикладі буде застосовано завантаження логіки додатку з файлу views.py поточного каталогу. Якщо користувач в URL адресі після hello (наприклад, http://127.0.0.1:8000/hello/) не вказуватиме інших адрес, тоді буде завантажено функцію index з views.py, якщо в адресі буде вказано hello/brian, тоді буде завантажено функцію brian з views.py, а у всіх інших випадках, коли користувач вказуватиме будь-що після hello/, буде завантажено функцію greet з views.py. Як видно з прикладу, паттерн <str:name> дозволяє зчитувати будь-який рядок тексту.

У файлі views.py створимо наступні функції контролера для логіки контролера:

```
from django.shortcuts import render
from django.http import HttpResponse
def index(request):
```

```
    return render(request, "hello/index.html")
def brian(request):
    return HttpResponse("<h1>Hello Brian</h1>")
def greet(request, name):
    return render(request, "hello/greet.html", {
        "name" : name.capitalize()
    })
```

Кожна окрема функція, що вказана в контролері `views.py`, завантажуватиме окремо логіку для заданого додатку. Наприклад, функція `index(request)` буде завантажувати цілу сторінку `index.html` з каталогу `hello_app/templates/hello`, а функція `brian(request)` передаватиме Http-відповідь з тегом `<h1>Hello Brian</h1>`, функція `greet(request, name)` здійсюватиме завантаження шаблону так само, як і функція `index(request)`, проте в заданий шаблон буде передаватися контекст у вигляді словника з єдиним параметром ключа `"name"` та значенням `name.capitalize()`, де параметр `name` буде передано даній функції автоматично з URL рядка. У всіх функціях-контролерах request-параметр є параметром за замовчуванням, які функції отримуватимуть автоматично.

Django автоматично шукає шаблони в директорії `templates` всередині додатку, тому для роботи з шаблонами обов'язково потрібно створити такий каталог в директорії нового додатку. Для того, щоб в шаблонах застосовувати параметри, які Django передає у функції-контролери, потрібно вказувати спеціальні літерали шаблонізації, наприклад, в шаблоні `greet.html`, щоб скористатися параметром `"name"`, потрібно назву даного параметру вказувати в фігурних дужках:

```
 <!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
       <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Hello</title>
</head>
<body>
```

```
    <h1>Hello, {{ name }}!</h1>
</body>
</html>
```

Логіка `if-else` та циклів в шаблоні підключається також через фігурні дужки, наприклад:

```
{% if newyear %}
        <h1>YES</h1>>
{% else %}
        <h1>NO</h1>
{% endif %}
```

або

```
{% for task in tasks %}
        <li>{{ task }}</li>
{% endfor %}
```

Підключення стилів до шаблонів в Django виконується через файли статичного завантаження. Для цього потрібно в директорії додатку створити каталог з назвою `static`, а у документі шаблону вказати його підключення, наприклад так:

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
  <head>
     <title>Is it New Year?</title>
     <link rel="stylesheet" href="{% static
     'newyear/styles.css' %}">
  </head>
  <body></body>
</html>
```

В даній роботі буде розглянуто побудову простого сервісу REST API для ведення нотаток. Більше інформації про REST API за посиланням: https://www.restapitutorial.com/.

Перед виконанням роботи рекомендується ознайомитися з офіційним посібником Django https://docs.djangoproject.com/en/1.11/intro/ та Django Rest Framework https://www.django-rest-framework.org/tutorial/quickstart/.

Для реалізації функціоналу REST в Django-проєкті, встановимо Django Rest Framework:

```
$ pip install djangorestframework
```

Підключимо створений застосунок для використання у проєкті. Для цього у файлі settings.py необхідно додати рядок налаштувань:

```
# Application definition
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
                                                'hello_app',
                                            'rest_framework',
]
```

В каталозі з проєктом створимо окремий додаток для нотаток:

```
$                       cd                          djangorest
$ python manage.py startapp todolist_app
```

Додаємо створений додаток у файл settings.py:

```
# Application definition
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'hello_app',
                                            'rest_framework',
                        'todolist_app.apps.TodolistAppConfig',
]
```

Створимо моделі для таблиці з бази даних. Для цього відредагуємо файл models.py в каталозі додатку todolist_app, описавши структуру даних в базі даних:

```
from django.db import models
class Task(models.Model):
      name  =  models.CharField(max_length=200,  blank=True)
         description = models.TextField(max_length=1000,
         blank=True)
          completed   =   models.BooleanField(default=False)
    date_created = models.DateField(auto_now_add=True)
    due_date = models.DateField(null=True, blank=True)
    date_modified = models.DateField(auto_now=True)
    PRIORITY = (
        ('h', 'High'),
```

```
        ('m', 'Medium'),
        ('l', 'Low'),
        ('n', 'None')
    )
    priority = models.CharField(max_length=1, choices =
         PRIORITY, default = 'n')
    def __str__(self):
        return "{}".format(self.name)
```

Після того, як були внесені зміни до файлу `models.py`, необхідно виконати створення таблиць (міграцію) в базі даних:

```
$ python manage.py makemigrations
```

далі

```
$ python manage.py migrate
```

Виконання даних команд повинно створити таблицю в стандартній базі даних Django-проєкту (за замовчуванням це база SQLite).

Далі, щоб працювати з Django REST, необхідно створити файл відображення даних та файл серилізатор даних.

Серіалізатор (https://www.django-rest-framework.org/api-guide/serializers/) це компонент, який дозволяє перетворювати складні об'єкти, отримані з бази даних, на прості та зрозумілі форми, наприклад, у формати JSON або XML. Створимо файл `todolist_app/serializers.py` з таким вмістом:

```
from rest_framework import serializers
from .models import Task
class TaskSerializer(serializers.ModelSerializer):
    class Meta:
        model = Task
        fields =('id', 'name', 'description', 'completed',
                 'date_created', 'date_modified',
'due_date',
                                                 'priority')
                     read_only_fields   =   ('date_created',
'date_modified')
```

Створимо декілька відображень (views), які дозволять:

- створити нотатку – POST-запит;
- видалити нотатку – DELETE-запит;
- оновити нотатку – PUT-запит;
- переглянути одну або декілька нотаток – GET-запит.

У файлі todolist_app/views.py створимо два відображення для створення та перегляду нотаток:

```
from rest_framework import generics
from .serializers import TaskSerializer
from .models import Task

class TaskCreateView(generics.ListCreateAPIView):
    queryset = Task.objects.all()
    serializer_class = TaskSerializer

class TaskDetailsView(generics.RetrieveAPIView):
    queryset = Task.objects.all()
    serializer_class = TaskSerializer
```

Пов'яжимо відображення з відповідними URL, для цього необхідно створити та відредагувати файл todolist_app/urls.py:

```
from django.conf.urls import url, include
from rest_framework.urlpatterns import
format_suffix_patterns
from .views import TaskCreateView, TaskDetailsView

urlpatterns = {
   path(r'', TaskCreateView.as_view(), name="create"),
   path(r'tasks/', TaskDetailsView.as_view(),
name="detail"),
}
urlpatterns = format_suffix_patterns(urlpatterns)
```

Далі необхідно цей файл включити до списку всіх маршрутів pythonprogect/urls.py:

```
from django.urls import path, include
from django.contrib import admin

urlpatterns = [
    path('admin/', admin.site.urls),
    path(r'api/todolist/', include('todolist_app.urls')),
]
```

Запустимо сервер та перевіримо коректність його роботи:

```
$ python manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
February 25, 2017 - 13:48:20
Django version 1.10.5, using settings 'djangorest.settings'
```

```
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Перейшовши за адресою http://127.0.0.1:8000/api/todolist/, бачимо відображення наступної сторінки (рис. 4.1):

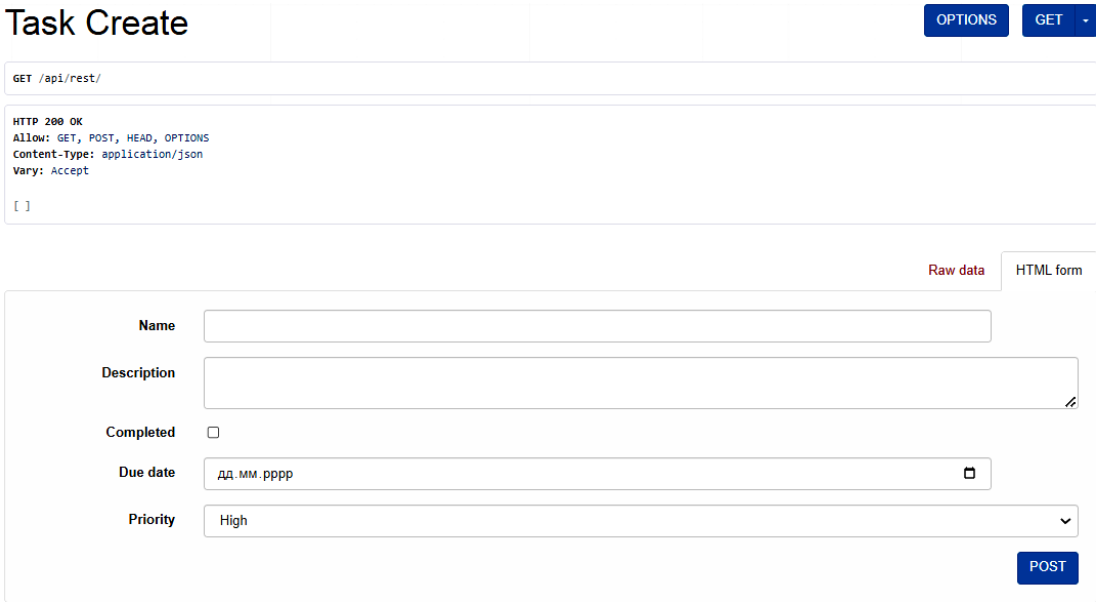


Рис. 4.1. Сторінка api/todolist

Оскільки в базі даних ще немає жодного запису, дані не відображаються, проте є поля для їхнього додавання. Додайте новий запис, для цього достатньо заповнити зазначені поля веб-форми і натиснути на посилання POST (рис. 4.2):

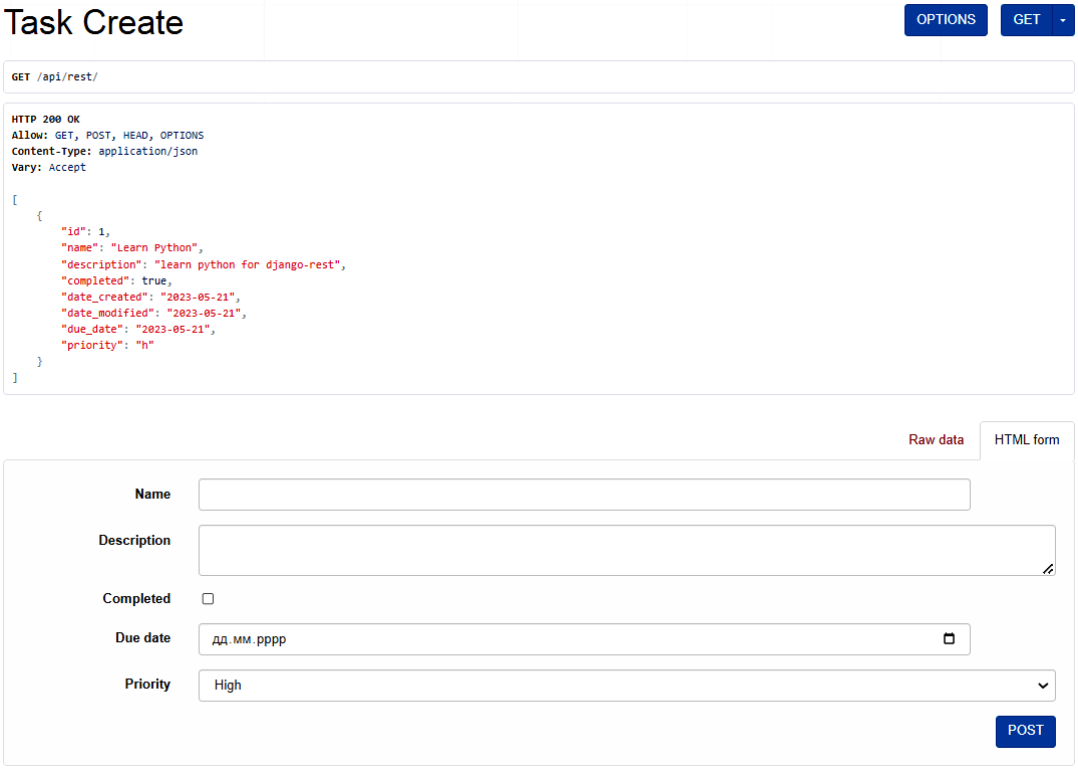


Рис. 4.2. Сторінка api/todolist після внесення першого запису

Після завантаження даних до бази даних, якщо звернутися за адресою http://127.0.0.1:8000/api/todolist/tasks, буде завантажена сторінка з всіма даними з бази даних застосунку (рис. 4.3):

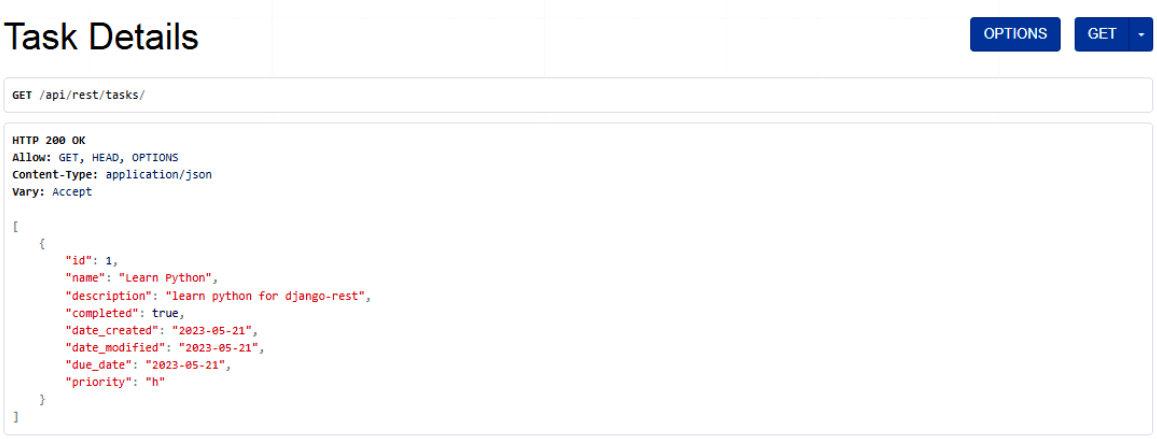


Рис. 4.3. Сторінка api/todolist/tasks

Додамо можливість об'єднання нотаток у групи, назвавши їх «Список завдань». Для цього потрібно додати модель для списку завдань `todolist_app/models.py`:

```
class Tasklist(models.Model):
    name = models.CharField(max_length=200, db_index=True)
    def __str__(self):
        return "{}".format(self.name)
```

Далі необхідно модифікувати модель відображення, показавши, що завдання входить у конкретний список завдань (тобто відношення один до багатьох – один список задач включає безліч завдань). Така поведінка відтворюється за допомогою зовнішнього ключа (про роботу із зовнішніми ключами можна почитати за наступним посиланням https://docs.djangoproject.com/en/1.10/topics/db/examples/many_to_one/ :

```
    class Task(models.Model):
     # ...
  tasklist = models.ForeignKey(Tasklist,
related_name='tasks', on_delete=models.CASCADE, null=True)
     # ...
```

Також необхідно додати серілізатор до списку завдань:

```
from .models import Tasklist

class TasklistSerializer(serializers.ModelSerializer):
    tasks = serializers.StringRelatedField(many=True)

    class Meta:
```

```
        model = Tasklist
        fields = ('name', 'tasks')
```

Для фіксування всіх зміни у БД необхідно оновити «міграції» наступними командами:

```
$ python manage.py makemigrations
$ python manage.py migrate
```

Далі визначимося зі списком методів, які надаватимемо користувачеві:

- /todolists – отримання всіх списків завдань або створення нового списку (GET та POST методи);
- /todolists/<list_id> – редагування або видалення списку завдань з ідентифікатором <list_id> (GET, PUT, DELETE);
- /todolists/<list_id>/tasks – перегляд завдань або створення нового завдання у списку задач з ідентифікатором list_id(GET та POST);
- /todolists/<list_id>/tasks/<task_id> – редагування або видалення завдання task_id у списку задач з ідентифікатором list_id (GET, PUT, DELETE).

Відобразимо список методів у маршрутах todolists_app/urls.py:

```
from django.urls import re_path
from rest_framework.urlpatterns import
format_suffix_patterns
from .views import TasklistCreateView, TasklistDetailsView,
TaskCreateView, TaskDetailsView
from rest_framework.authtoken.views import
obtain_auth_token

urlpatterns = [
    re_path(r'^todolists/$', TasklistCreateView.as_view(),
name="lists"),
    re_path(r'^todolists/(?P<pk>[0-9]+)/$',
TasklistDetailsView.as_view(), name="list-detail"),
    re_path(r'^todolists/(?P<list_id>[0-9]+)/tasks$',
TaskCreateView.as_view(), name="tasks"),
    re_path(r'^todolists/(?P<list_id>[0-9]+)/tasks/(?
P<pk>[0-9]+)$', TaskDetailsView.as_view(), name="task-
detail"),
]
```

```
urlpatterns = format_suffix_patterns(urlpatterns)
```

Логіка функцій відображення потрібних завдань в файлі views.py:

```
class TasklistCreateView(generics.ListCreateAPIView):
    queryset = Tasklist.objects.all()
    serializer_class = TasklistSerializer

class TasklistDetailsView
(generics.RetrieveUpdateDestroyAPIView):
    queryset = Tasklist.objects.all()
    serializer_class = TasklistSerializer

class TaskCreateView(generics.ListCreateAPIView):
    serializer_class = TaskSerializer
    def get_queryset(self):
        queryset = Task.objects.all()
        list_id = self.kwargs.get('list_id', None)
        if list_id is not None:
            queryset = queryset.filter(tasklist_id =
list_id)
        return queryset
    def perform_create(self, serializer):
        list_id = self.kwargs.get('list_id', None)
        try:
            tasklist = Tasklist.objects.get(pk=list_id)
        except Tasklist.DoesNotExist:
            raise NotFound()
        serializer.save(tasklist=tasklist)

class
TaskDetailsView(generics.RetrieveUpdateDestroyAPIView):
    serializer_class = TaskSerializer
    def get_queryset(self):
        queryset = Task.objects.all()
        list_id = self.kwargs.get('list_id', None)
        if list_id is not None:
            queryset = queryset.filter(tasklist_id =
list_id)
        return queryset
```

ПОСЛІДОВНІСТЬ ВИКОНАННЯ РОБОТИ

1. Ознайомитися з веб-фреймворком Django на прикладі коду, наведеного в роботі.

2. Дослідити можливості створення веб-застосунків.

3. Дослідити роботу REST на прикладі коду в даній роботі.

4. Розробити власний веб-застосунок з можливістю підтримки API засобами REST для відображення даних в JSON форматі.

5. Зробити висновки щодо роботи.

6. Самостійно знайти відповіді на наступні запитання:

– що таке JSON формат?
– які основні методи в REST?
– що таке «модель» в контексті django?
– що таке «міграції» та як це пов’язано з базами даних?
– як django передає дані в бази даних?

ЗАВДАННЯ № 5. Вивчення принципів локалізації на основі фільтра Калмана

Мета роботи: познайомитися з поняттям «локалізація» в робототехніці та дослідити алгоритм фільтра Калмана. Навчитися визначати параметри фільтра.

Основні теоретичні відомості

Локалізація – це процес визначення положення робота в робочому середовищі (рис. 5.1). Формальна постановка задачі локалізації полягає в наступному: нехай є робот, який знаходиться в невідомому середовищі і потрібно визначити його точне географічне положення або положення в просторі. Середовище може бути відомим або невідомим, а робот може мати доступ до різних датчиків, таких як датчики відстані, візуальні камери, акселерометри, гіроскопи. Для вирішення задачі локалізації використовують різні алгоритми обробки даних від датчиків робота.

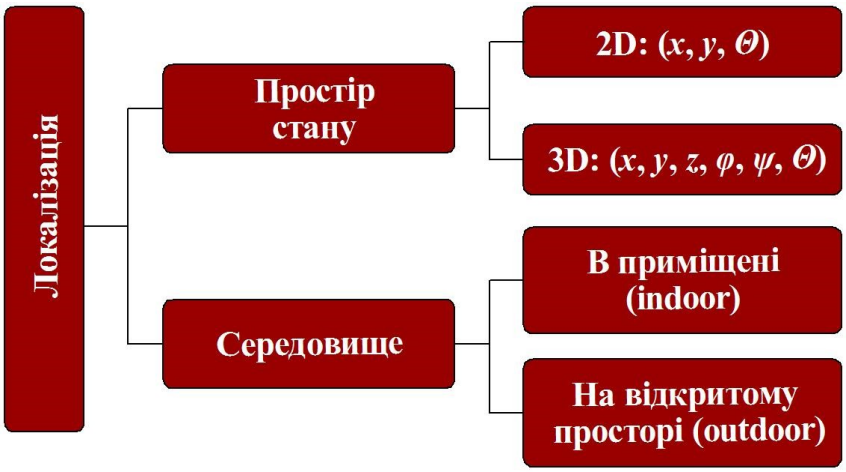


Рис. 5.1. Принципи поняття локалізації

Основні методи локалізації включають:

- **візуальну локалізацію –** при цьому робот використовує візуальні дані, такі як зображення або відеопотік, для визначення свого положення в середовищі. Цей підхід може використовувати методи комп'ютерного зору та принципи і моделі лазерного сканування;
- **використання датчиків розташування**, таких як GPS, акселерометри або гіроскопи, для визначення свого географічного положення або орієнтації в просторі;
- **симуляцію та картографування**, що використовує інформацію з відомої карти середовища разом з даними датчиків для вирішення задачі локалізації. Цей підхід може вимагати попереднього сканування середовища або побудови карт за допомогою датчиків робота;
- **фільтрацію та статистичні методи**, які використовують алгоритми фільтрації, такі як фільтр Калмана або частково обмежені фільтри, для обробки даних з датчиків та визначення свого положення в реальному часі;
- **використання маркерів або точок відліку**, зокрема QR-коди або розмітку на підлозі, для визначення положення та орієнтації.

Задача локалізації може бути сформульована як проблема статистичного оцінювання, де мета полягає в тому, щоб на основі отриманих даних з датчиків робота визначити ймовірність його поточного положення. Формально, задача локалізації може бути виражена так:

- **модель робота** визначає стан робота шляхом параметрів, які описують його географічне положення або положення в просторі;
- **модель середовища** визначає характеристики середовища, такі як об'єкти, перешкоди, відомі точки відліку;
- **модель датчиків** визначає співвідношення між реальним станом робота та даними, які отримуються від датчиків;
- **функція вірогідності** визначає ймовірності отримання певного набору даних в умовах певного положення робота;

• **оцінка стану** визначає найбільш вірогідний стан робота на основі отриманих даних з датчиків.

Ймовірнісна постановка задачі локалізації в робототехніці полягає в наступному. Введемо просторову систему координат, в якій робот може визначити своє положення. Нехай x_t – це стан робота в момент часу t, включаючи його географічні координати або положення в просторі. Робот може здійснювати керуючі дії, які змінюють його стан. Ці дії можуть включати переміщення в просторі або обертання. Динаміку стану робота x_t описується за допомогою рекурентного виразу:

$$x_t = f(x_{t-1}, u_t, w_t) \quad , \tag{5.1}$$

де u_t – керуючий вплив на момент часу t, а w_t – випадкова помилка або шум, пов'язаний з рухом.

Робот отримує спостереження про своє середовище за допомогою своїх датчиків. Нехай Z_t – це спостереження робота в момент часу t. Введемо модель спостереження $P(Z_t \mid X_t)$, яка визначає ймовірність отримання спостереження Z_t за умови, що робот перебуває в стані X_t. Визначимо початковий стан робота X_0 та ймовірність $P(X_0)$. Тоді метою задачі локалізації є оцінка або передбачення стану робота X_t на основі його керуючих впливів u_t та отриманих спостережень Z_t.

Розв'язок задачі локалізації може бути виконано за допомогою різних алгоритмів, таких як фільтр Калмана, фільтра частинок, методу Монте-Карло тощо. Основною метою цих алгоритмів є оцінка стану робота на основі отриманих даних з датчиків та моделей середовища.

Дослідимо метод фільтрації Калмана на простій одномірній задачі.

Розглянемо рух робота в одномірному просторі (рис. 5.2). Позначимо за x_k величину, яка вимірюється, а потім фільтрується. Це може бути координата, швидкість або прискорення робота.

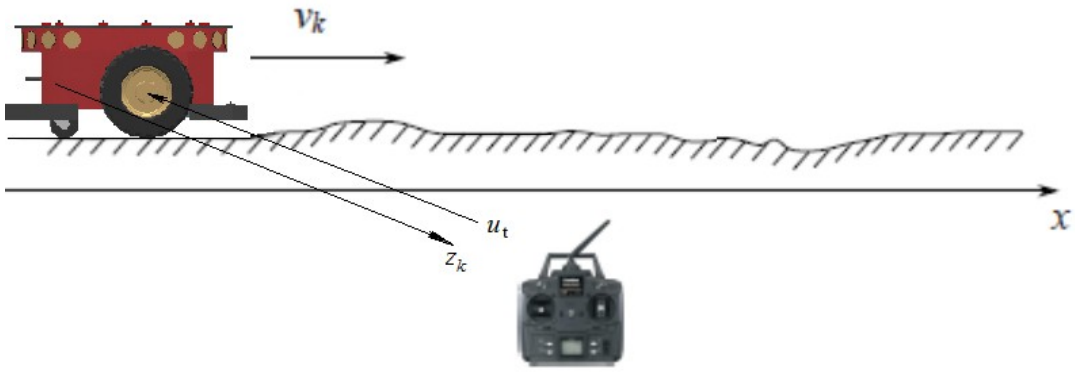


Рис. 5.2. Схема переміщення робота в одномірному просторі

Дослідимо випадок, коли робот може переміщуватися лише вперед і назад. Зміна лінійної координати робота в такому випадку змінюватимуться за законом:

$$x_{k+1} = x_k + vdt, \tag{5.2}$$

де v – швидкість переміщення платформи робота.

На робота в реальних умовах діють зовнішні збурення (вітер, нерівність дороги тощо), тому його справжня швидкість відрізнятиметься від розрахункової, а отже до правої частини рівняння (12.2) додається випадкова величина ξ_k:

$$x_{k+1} = x_k + vdt + \xi_k. \tag{5.3}$$

З іншого боку на роботі встановлено датчик, який вимірює справжню координату x_k положення робота з помилкою η_k, яка також є випадковою величиною. У результаті з датчика буде отримано помилкові дані:

$$z_k = x_k + \eta_k. \tag{5.4}$$

Знаючи хибні покази датчика, знайдемо найкраще наближення $x_k^{опт}$ для істинної координати положення робота x_k.

Позначимо керування системою $u_k = vdt$, тоді рівняння для координати та показання датчика виглядатимуть так:

$$x_{k+1} = x_k + u_k + \xi_k; \tag{5.5}$$

$$z_k = x_k + \eta_k, \tag{5.6}$$

де u_k – це відома величина, яка контролює еволюцію системи; ξ_k, η_k – це похибка моделі та датчика, відповідно, які є випадковими величинами, а їхні закони розподілу не залежать від часу (від кроку ітерації k, а їхні середні значення дорівнюють нулю, тобто $E(\xi_k) = E(\eta_k) = 0$).

Закон розподілу випадкових величин може бути невідомий, проте можуть бути відомі їхні дисперсії $\sigma^2(\xi_k)$ і $\sigma^2(\eta_k)$. Зауважимо, що дисперсії не залежать від k, а всі випадкові похибки є незалежними одна від одної.

Завдання фільтрації – це не задача згладжування даних із датчика, а визначення найближчого значення до реальної координати x_k. Якщо на κ-ом кроці знайдено відфільтроване значення з датчика $x_k^{\text{опт}}$, що добре наближає справжню координату системи x_k, тоді на κ+1 кроці система еволюціонує згідно із законом руху і датчик покаже щось близьке до $x_k^{\text{опт}} + u_{\text{k}}$. З іншого боку, на кроці κ+1 буде відоме неточне показання сенсора z_{k+1}. Ідея фільтра Калмана полягає в тому, щоб отримати найкраще наближення до істинної координати x_{k+1}, а тому потрібно вибрати золоту середину між значенням z_{k+1} неточного датчика та $(x_k^{\text{опт}} + u_{\text{k}})$ – прогнозом того, що очікується побачити. Показанням датчика дамо вагу K, а для передбаченого значення залишиться вага (1 - K), тоді:

$$x_{k+1}^{\text{опт}} = K z_{k+1} + (1 - K)(x_k^{\text{опт}} + u_k). \qquad (5.7)$$

Коефіцієнт K називають коефіцієнтом Калмана. Цей коефіцієнт залежить від кроку ітерації, тому правильним буде запис K_{k+1}. Коефіцієнт Калмана вибирається так, щоб оптимальне значення координати $x_{k+1}^{\text{опт}}$ було найближче до істинної координати x_{k+1}. Таким чином, якщо датчик дуже точний, тоді більша довіра його показанням і значення z_{k+1} буде мати більшу вагу (K близько до одиниці). Якщо ж датчик, навпаки, зовсім неточний, тоді більша довіра на теоретичне значення $x_k^{\text{опт}} + u_k$.

В загальному випадку, щоб знайти точне значення коефіцієнта Калмана, потрібно мінімізувати похибку:

$$e_{k+1} = x_{k+1} - x_k^{\text{опт}}, \qquad (5.8)$$

Вираз (12.8) можна записати так:

$$e_{k+1} = (1 - K)(e_k + \xi_k) - K\eta_{k+1}. \qquad (5.9)$$

Мінімізуватимемо середнє значення від квадрата похибки:

$$E(e^2_{k+1}) \to \min. \qquad (5.10)$$

Розпишемо останній вираз:

$$E(e^2_{k+1}) = (1 - K)^2 (E(e^2_k) + \sigma^2_\xi) + K^2 \sigma^2_\eta. \qquad (5.11)$$

Вираз (12.1) набуває мінімального значення, коли прирівнюємо похідну до нуля, тобто:

$$K_{k+1} = \frac{Ee_k^2 + \sigma_\xi^2}{Ee_k^2 + \sigma_\xi^2 + \sigma_\eta^2} . \quad (5.12)$$

Дослідіть наступний код в Python з тестом фільтра Калмана:

```
import numpy as np
import matplotlib.pyplot as plt

N = 100  # number of samples
a = 0.1  # acceleration
sigmaPsi = 1
sigmaEta = 50
k = np.arange(1, N+1)
x = np.zeros(N)
z = np.zeros(N)
x[0] = 0
z[0] = x[0] + np.random.normal(0, sigmaEta)
for t in range(1, N):
    x[t] = x[t-1] + a*t + np.random.normal(0, sigmaPsi)
    z[t] = x[t] + np.random.normal(0, sigmaEta)

# Kalman filter
xOpt = np.zeros(N)
eOpt = np.zeros(N)
xOpt[0] = z[0]
eOpt[0] = sigmaEta

for t in range(1, N):
    eOpt[t] = np.sqrt((sigmaEta**2) * (eOpt[t-1]**2 +
sigmaPsi**2) / (sigmaEta**2 + eOpt[t-1]**2 + sigmaPsi**2))
    K = (eOpt[t]**2) / sigmaEta**2
    xOpt[t] = (xOpt[t-1] + a*t) * (1 - K) + K * z[t]

# Plotting
plt.plot(k, xOpt, label='xOpt')
plt.plot(k, z, label='z')
plt.plot(k, x, label='x')
plt.xlabel('Sample')
plt.ylabel('Value')
plt.legend()
plt.show()
```

ПОСЛІДОВНІСТЬ ВИКОНАННЯ РОБОТИ

1. Ознайомитися з основними теоретичними відомостями щодо локалізації та фільтрації.

2. Ознайомитися з алгоритмом фільтра Калмана.

3. Ознайомитися самостійно з іншими видами фільтрів та скласти звіт з описом відомих алгоритмів фільтрації сигналів.

4. Самостійно знайти відповіді на наступні запитання:

- що таке дисперсія випадкової величини?
- що таке математичне очікування випадкової величини?
- в чому суть теореми Байєса?
- що таке Марківський процес?
- як застосувати фільтр Калмана для багатомірної задачі?

ЗАВДАННЯ № 6. Дослідити алгоритм пошуку в глибину

Мета роботи: ознайомитися з алгоритмом пошуку в глибину (DFS), вивчити принципи роботи алгоритму, проаналізувати переваги та недоліки, розглянути приклади застосування.

Основні теоретичні відомості

Алгоритм пошуку в глибину (DFS) – це алгоритм обходу зв'язних структур даних (граф, дерево, зображення тощо), який досліджує один напрямок розгалуження структури на її максимальній довжині (в глибину), перш ніж переходити до наступної вершини (початку) розгалуженої структури. DFS починається з початкової вершини і повністю досліджує одну із сусідніх вершин з усіма можливими її розгалуженнями. Потім алгоритм переходить до сусідніх недосліджених вершин і так доти, доки не дослідить всі вершини, які можна досягти з початкової вершини. Якщо структуру даних представляти у вигляді графа, тоді DFS обходить вершини графа в порядку LIFO (останній прийшов – перший вийшов, стек).

Алгоритм пошуку в глибину працює наступним чином:

1. Приймають початкову точку входу до структури даних, яка і буде вершиною графа зв'язку для досліджуваної структури даних.
2. Помістити початкову вершину графа на вершину стека.
3. Взяти верхній елемент стека і додати його до списку пройдених.
4. Створити список суміжних вершин від досліджуваної вершини і додати ті вершини, яких немає у списку «пройдених» до стеку.
5. Повторювати кроки 3 і 4 доти, доки стек не стане порожнім.

Розглянемо на прикладі, як працює алгоритм пошуку у глибину, застосовуючи неорієнтований граф із п'ятьма вершинами (рис. 6.1).

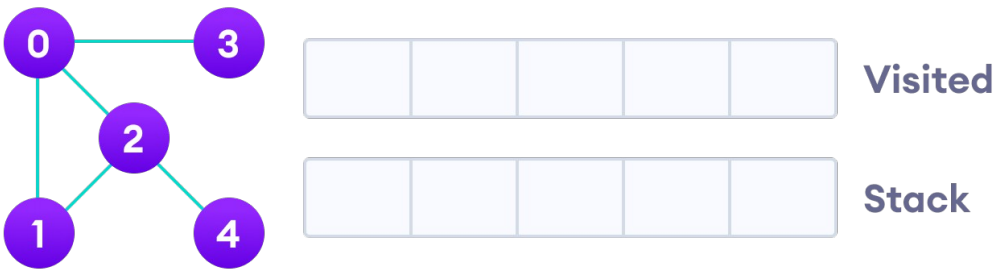


Рис. 6.1. Неорієнтований граф із п'ятьма вершинами

Розглянемо алгоритм DFS на графічному прикладі для графа на рис. 6.1 і почнемо з вершини «0», помістивши її в список «пройдені» (на зображенні «Visited»), а її суміжні вершини 1, 2, 3 – у стек.

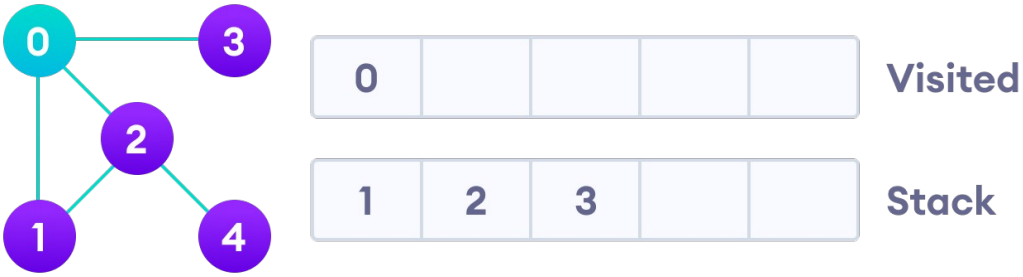


Рис. 6.2. Крок ініціалізації алгоритму

На наступному кроці зі стеку зверху береться останній елемент, тобто елемент «1», та переміщується до списку «Пройдені» та досліджують сусідні вершини з даною. Оскільки сусідньою вершиною до вершини «1» є вершина «2», тоді дана вершина повинна бути внесена до стеку, проте ця вершина вже є на стеці. Переходять до обробки вершини «2» (рис. 6.3).

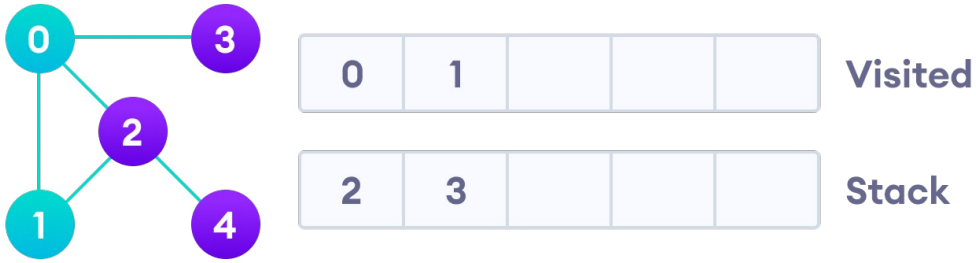


Рис. 6.3. Крок обходу елементів на вершині стеку

Вершина «2» суміжна непройденій вершині «4», отже додаємо її вгору стека і проходимо її (рис. 6.4).

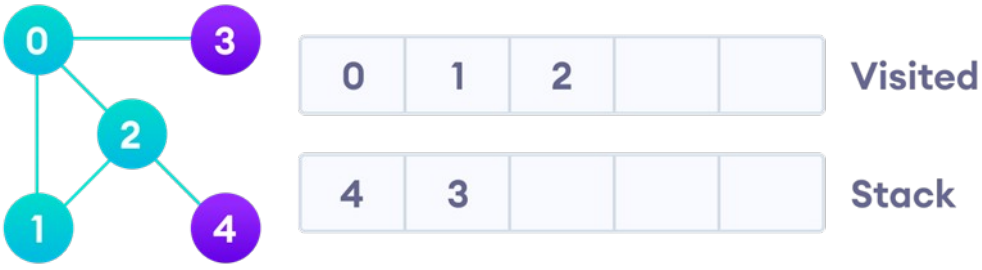


Рис. 6.4. Крок переміщення вершини «4» до початку стеку

Додаємо вершину «4» до списку «Пройдені» після проходження (рис. 6.5).

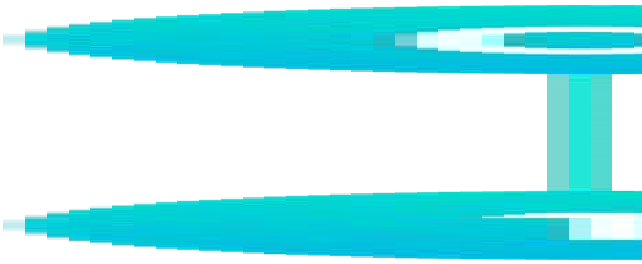


Рис. 6.5. Крок переміщення вершини «4» до списку пройдених

Після того, як буде пройдемо останній елемент (вершина «3»), в стеку не залишиться непройдених суміжних вершин, і таким чином обхід графа у глибину буде завершено (рис. 6.6).

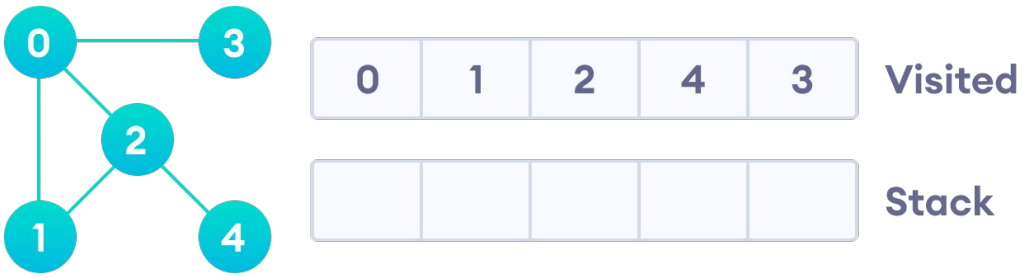


Рис. 6.6. Крок завершення обходу в глибину

Нижче наведено приклад реального коду алгоритму на Python пошуку в глибину:

```
def dfs(graph, start, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    print(start)
    for next in graph[start] - visited:
        dfs(graph, next, visited)
    return visited

graph = {'0': set(['1', '2']),
         '1': set(['0', '3', '4']),
         '2': set(['0']),
         '3': set(['1']),
         '4': set(['2', '3'])}
dfs(graph, '0')
```

ПОСЛІДОВНІСТЬ ВИКОНАННЯ РОБОТИ

1. Ознайомитися з алгоритмом обходу в глибину.

2. Як виглядатимуть реалізації структур даних на мові Python, які необхідні для розв’язку задач обходу в глибину.

3. Реалізуйте програму, яка на вхід першим параметром приймає ім’я файлу, наприклад: ‘test.jpg’ (рис. 6.7) та видає кількість силуетів на картинці. Для «обходу» силуету застосуйте власний алгоритм DFS.

Рис. 6.7. Зображення для обробки

4. Зробіть висновки по роботі із зазначенням, де може бути корисним алгоритм DFS.

ЗАВДАННЯ № 7. Дослідити алгоритм пошуку в ширину

Мета роботи: ознайомитися з алгоритмом пошуку в ширину (BFS), вивчити принципи роботи алгоритму, проаналізувати переваги та недоліки, розглянути приклади застосування.

Основні теоретичні відомості

Алгоритм пошуку в ширину (BFS) використовується для пошуку найкоротшого шляху на незваженому графі або обходу всіх вершин графа в порядку поступового збільшення відстані від початкової вершини.

Основні кроки алгоритму BFS:

- Визначити початкову точку, від якої необхідно почати обхід всіх інших точок.
- Починаючи з початкової вершини, помістити її в чергу (структуру за принципом перший зайшов, перший вийшов).
- Позначити початкову вершину як відвідану та дослідити всі сусідні вершини.
- Для кожної сусідньої вершини сусіда, який ще не був відвіданий, позначте його як відвіданого та додайте до черги.
- Повторюйте кроки 3 – 4 дотри, доки черга не спорожніє.

Пропонуємо дослідити самостійно алгоритм BFS на наступному прикладі:

```
from collections import deque

def bfs(graph, start):
    visited = set()
    queue = deque([start])
    visited.add(start)

    while queue:
        vertex = queue.popleft()
        print(vertex)
        for neighbor in graph[vertex]:
            if neighbor not in visited:
                visited.add(neighbor)
                queue.append(neighbor)
graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D', 'E'],
    'C': ['A', 'F'],
    'D': ['B'],
    'E': ['B', 'F'],
    'F': ['C', 'E']
}
bfs(graph, 'A')
```

Математична морфологія (ММ) – теорія і техніка аналізу та обробки геометричних структур, заснована на теорії множин, топології та випадкових функціях, яка застосовується для цифрової обробки зображень в комп’ютерному зорі.

Основні функції математичної морфології включають в себе ерозію, розширення, відкриття та закриття, а також операції тонкої обробки та скелетизації і виконуються з використанням структурних елементів, які представляють собою невеликі форми або вікна, використовувані для сканування зображення (див. рис. 7.1):

– ерозія зменшує об’єкти на зображенні шляхом видалення пікселів на їх межах. Вона застосовується шляхом сканування зображення з використанням структурного елемента та видалення центрального пікселя, якщо не всі пікселі структурного елемента відповідають пікселям об’єкта;

– розширення збільшує об’єкти на зображенні шляхом додавання пікселів до їх меж. Воно застосовується шляхом сканування зображення з використанням структурного елемента та додавання центрального пікселя, якщо принаймні один піксель структурного елемента відповідає пікселям об’єкта;

– відкриття – це послідовне застосування ерозії, за якою слідує розширення. Воно може використовуватися для видалення дрібних деталей та шумів, зберігаючи основні структури об’єктів;

– закриття – це послідовне застосування розширення, за яким слідує ерозія. Воно може використовуватися для заповнення невеликих отворів в об’єктах;

– тонка обробка використовується для зменшення товщини об’єктів на зображенні до одного пікселя в ширину. Вона застосовується шляхом видалення пікселів об’єкта доти, доки об’єкт залишатиметься зв’язним;

– скелетизація – це процес, обернений тонкій обробці, який використовується для виділення «скелета» об’єктів на зображенні, що представляє собою мінімальне зв’язне представлення об’єкта.

Рис. 7.1. Форма (синім кольором) і його морфологічне розширення (зеленим кольором) і ерозія (жовтим кольором) ромбоподібним структурним елементом

Основна ідея бінарної морфології полягає в тому, щоб досліджувати зображення за допомогою простої, попередньо визначеної форми, роблячи висновки про те, як ця форма підходить або пропускає форми на зображенні. Цей простий «зонд» називається структурним елементом (рис. 7.2) і сам по собі є бінарним зображенням (тобто підмножиною простору або сітки).

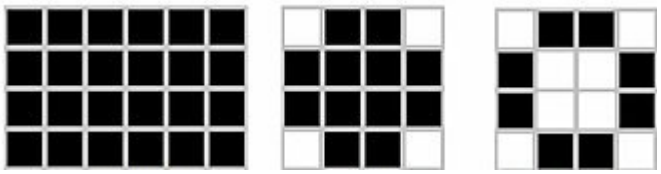


Рис. 7.2. Найбільш поширені структурні елементи: BOX[H,W] – прямокутник заданого розміру; DISK[R] – диск заданого розміру; RING[R] – кільце заданого розміру

Ерозія – це одна з основних процедур в математичній морфології, яка використовується для зменшення контуру об’єктів на зображенні шляхом видалення пікселів на їх межах. Цей процес допомагає видаляти маленькі деталі, шум та розмитість, змінюючи форму об’єктів та уточнюючи їх границі.

Алгоритм процесу ерозії можна описати наступним чином:

- Визначте структурний елемент, який буде використовуватися для сканування зображення.
- Почніть сканування зображення з використанням цього структурного елемента.
- Для кожної позиції структурного елемента перевірте, чи всі пікселі відповідають пікселям об’єкта на зображенні.
- Якщо всі пікселі відповідають, центральний піксель залишається, в протилежному випадку він стає чорним.
- Повторюйте цей процес для кожної позиції на зображенні.

Для ерозії обробки зображення можна скористатися бібліотекою OpenCV в Python. Приклад коду для застосування ерозії до зображення:

```
import cv2
import numpy as np
def custom_erosion(image, kernel):
    rows, cols = image.shape[:2]
    k_rows, k_cols = kernel.shape[:2]
    result = np.zeros_like(image)

    for i in range(k_rows//2, rows-k_rows//2):
        for j in range(k_cols//2, cols-k_cols//2):
            min_value = 255
            for m in range(k_rows):
                for n in range(k_cols):
                    pixel_value = image[i+m-k_rows//2][j+n-
k_cols//2]
                    if kernel[m][n] == 1:
                        if pixel_value < min_value:
                            min_value = pixel_value
            result[i][j] = min_value
    return result

image = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)
kernel = np.ones((5,5), np.uint8)
erosion_image = custom_erosion(image, kernel)
cv2.imshow('Original Image', image)
cv2.imshow('Erosion Image', erosion_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Також можна застосувати вбудовану функцію для ерозії в бібліотеку openCV:

```
import cv2
import numpy as np
image = cv2.imread('image.jpg')

kernel = np.ones((5,5), np.uint8)
erosion_image = cv2.erode(image, kernel, iterations=1)
cv2.imshow('Original Image', image)
cv2.imshow('Erosion Image', erosion_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Алгоритм для цього коду виглядає наступним чином:

- зчитати зображення за допомогою функції **cv2.imread()**;
- задати структурний елемент для ерозії. У представленому прикладі використовується квадратний структурний елемент, розміром 5x5;
- застосувати ерозію до зображення за допомогою функції **cv2.erode()**;
- відобразити початкове та оброблене зображення за допомогою функції **cv2.imshow()**;

Ось пояснення алгоритму функції **custom_erosion**:

- визначаємо розміри вхідного зображення **image** та структурного елемента (ядра) **kernel**. Для зручності також отримуємо розміри ядра **k_rows** і **k_cols**;
- створюємо пусте зображення **result**, яке матиме той же розмір, що й вхідне зображення, і заповнимо його нулями;
- в циклі перебираємо всі пікселі зображення, залишаючи краї незмінними. Для цього використовуємо діапазони **range(k_rows//2, rows - k_rows//2)** і **range(k_cols//2, cols - k_cols//2)**, щоб виключити краї з обробки;
- для кожного пікселя в ядрі перевіряємо, чи дорівнює він 1. Це означає, що він є частиною структурного елемента. Якщо так, ми обчислюємо мінімальне значення пікселя серед сусідніх пікселів, використовуючи координати зображення та відповідні координати ядра;

- записуємо знайдене мінімальне значення у відповідний піксель нашого результату **result**;
- після завершення обробки всіх пікселів повертаємо зображення **result**, яке містить результат ерозії.

ПОСЛІДОВНІСТЬ ВИКОНАННЯ РОБОТИ

1. Ознайомитися з алгоритмом обходу в ширину.
2. Ознайомитися з алгоритмами математичної морфології та дослідити код для ерозії на представлених прикладах на мові Python.
3. Реалізуйте програму, яка на вхід першим параметром приймає ім’я файлу, наприклад: ‘test.jpg’ (рис. 7.3) та видає кількість силуетів на картинці. Для «обходу» силуету застосуйте власний алгоритм BFS.

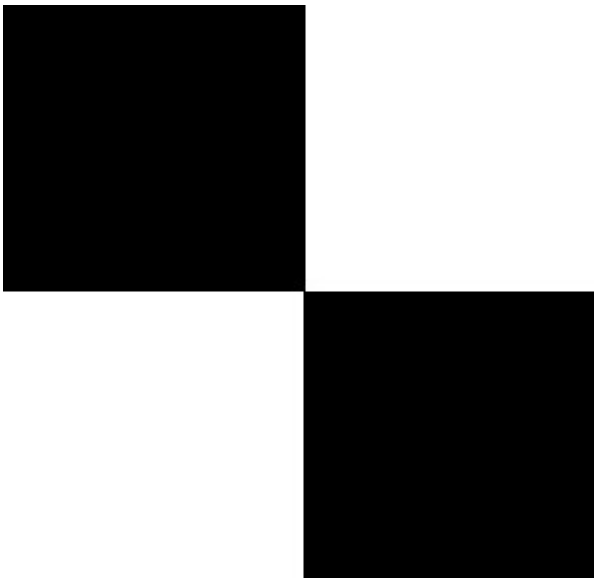


Рис. 7.3. Зображення для обробки

4. Зробіть висновки по роботі із зазначенням, де може бути корисним алгоритм BFS.

ЗАВДАННЯ № 8. Вивчення комп’ютерного зору на прикладі відеопотоків

Мета роботи: навчитися використовувати бібліотеки обробки відеопотоків для розпізнавання жестів у реальному часі з веб-камери або відеофайлу.

Теоретичні відомості

OpenCV (Open Source Computer Vision Library) – це бібліотека з відкритим кодом, що надає широкий спектр інструментів для комп'ютерного зору та обробки зображень.

OpenCV може читати відео з файлів або веб-камер, а також записувати оброблене відео. Ви можете отримувати доступ до окремих кадрів відео, а також обробляти їх за допомогою різних алгоритмів, таких як:

- зміна розміру;
- поворот;
- фільтрація;
- перетворення кольорів;
- розпізнавання обличчя;
- відстеження об'єктів.

OpenCV може використовуватися для вилучення характеристик з відео, таких як оптичний потік, траєкторії об'єктів, текстури та форми. Ці характеристики можна використовувати для машинного навчання та розробки складних систем комп'ютерного зору.

Основні методи OpenCV для роботи з відео:

- VideoCapture – клас для читання відео з файлів або веб-камер;
- VideoWriter – клас для запису обробленого відео;
- Imread – функція для читання зображення з файлу;
- Imshow – функція для візуалізації зображення;
- CvtColor – функція для перетворення кольорів зображення;
- Canny – функція для виявлення країв зображення;
- FindContours – функція для знаходження контурів об'єктів на зображенні;
- DrawContours – функція для візуалізації контурів об'єктів.
- CascadeClassifier – клас для каскадного класифікатора, який використовується для розпізнавання обличчя, об'єктів та інших патернів.

Для обробки статичних зображень openCV має інструменти для роботи з відеопотоками. Наступний лістинг коду показує, як програмно можна обробляти відео:

```
#посилання на тестове відео http://surl.li/qnsih

import cv2
```

```
cap = cv2.VideoCapture("video_test.mp4")
object_detection = cv2.createBackgroundSubtractorMOG2(
history=100,varThreshold=50)

while True:
    ret, frame = cap.read()
    frame = cv2.resize(frame, (800, 600))
    # h, w, _ = frame.shape
    # print(h, w)
    roi = frame[150: 300, 100: 400]
    mask = object_detection.apply(roi)
           _,   mask   =   cv2.threshold(mask,   250,   255,
cv2.THRESH_BINARY)
      contours, _ = cv2.findContours(mask, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
    for cnt in contours:
        area = cv2.contourArea(cnt)
        if area > 700:
           x, y, w, h = cv2.boundingRect(cnt)
            cv2.rectangle(roi, (x, y), (x+w, y+h), (0, 255,
0), 3)
    cv2.imshow("Roi", roi)
    cv2.imshow("Frame", frame)
    # cv2.imshow("Mask", mask)
    if cv2.waitKey(3) & 0xFF == ord('q'):
       break
cap.release()
cv2.destroyAllWindows()
```

Алгоритм роботи наступний:

1. Спочатку потрібно завантажити тестове відео (рис. 8.1) за вказаним адресом або використати власне. Для простоти роботи рекомендується завантажене відео розміщувати поряд з файлом основної програми.

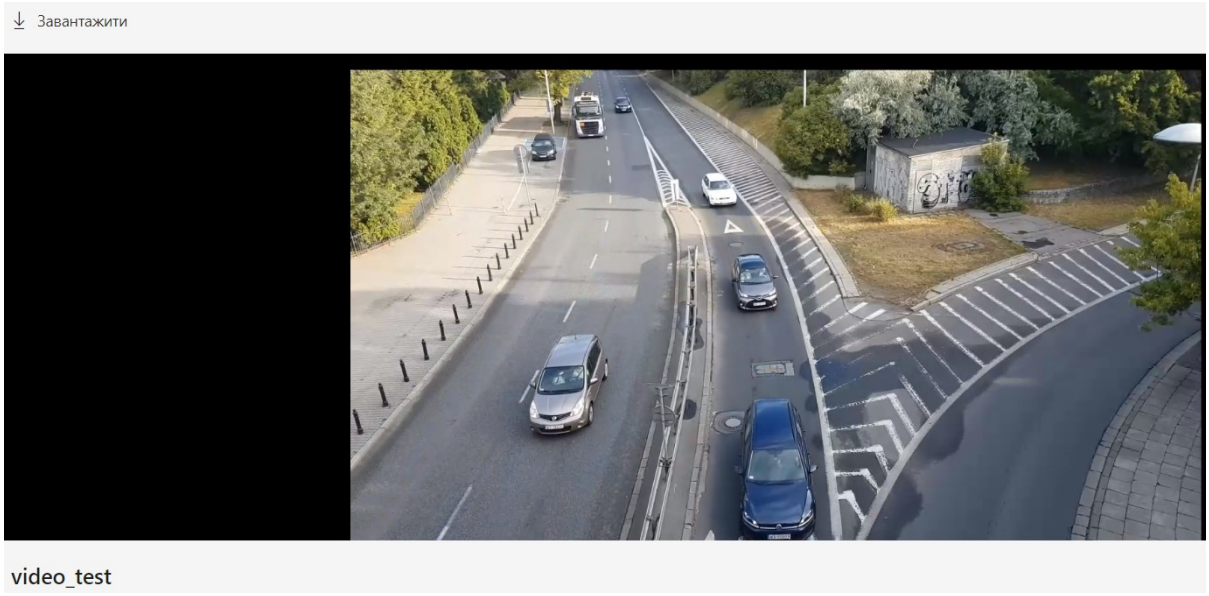


Рис. 8.1. Приклад завантаженого відео

2. Підключити бібліотеку openCV до проєкту.

3. Програмно за допомогою метода VideoCapture відкрити відео та передати його потік в об’єкт для подальшої обробки.

4. Створюємо об’єкт для виділення рухомих об'єктів на основі алгоритму MOG2 (з історією 100 кадрів і порогом 50). **MOG2** (Mixture of Gaussian 2) – це алгоритм розділення фонів, який використовується для виділення рухомих об’єктів (транспортних засобів, живих істот) у відеопотоці. Алгоритм MOG2 моделює кожен піксель кадру як суміш двох гаусових розподілів: **Фоновий розподіл:** Моделює пікселі, які належать фону, та **Розподіл переднього плану:** Моделює пікселі, які належать рухомим об’єктам. Алгоритм постійно оновлює ці два розподіли, щоб адаптуватися до змін у фоні та освітленні. Пікселі, які не відповідають жодному з розподілів, вважаються пікселями переднього плану (рухомими об’єктами). MOG2 може бути чутливим до шуму в кадрі і може неточно виділяти рухомі об’єкти, особливо якщо вони мають подібний колір до фону.

5. В циклі прочитати кадр з відеопотоку через метод `read()` об’єкта `cap`. Розмір кадру можна конфігурувати за власним розміром, наприклад, методом `resize(frame, (800, 600))`.

6. Виділяємо область інтересу (невелику область, де буде працювати алгоритм спостереження) з кадру. Це робиться для зменшення навантаження на апаратне забезпечення комп’ютера.

7. Застосовуємо алгоритм виділення рухомих об'єктів та перетворюємо маску рухомих об'єктів на бінарне зображення (`cv2.threshold (mask, 250, 255, cv2.THRESH_BINARY)`).

8. Знаходимо контури об'єктів на бінарному зображенні методом `findContours`.

9. Всі знайдені контури відображаємо на екрані.

Для підключення до відеосистеми комп'ютера через бібліотеку openCV існує наступний простий код:

```
import numpy as np
import cv2
WIDTH = 800
HEIGHT = 600
cap = cv2.VideoCapture(1)
while True:
    ret, frame = cap.read()
    cv2.imshow("Frame", frame)
    if cv2.waitKey(1) == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

В даному коді застосовується метод VideoCapture з параметром «1». Цей парамет відповідає за послідовний номер пристрою, до якого буде здійснено піднлючення. Якщо пристрою не буде знайдно, відео не буде відображено. Якщо до компютера підключено відеокамеру, але відео не відображається, перевірте з'єднання або змініть номер пристрою, який може бути від «0» до будь-якого цілого.

Окрім openCV, існують й інші бібліотеки для обробки зображень на основі інтелектуальних методів. Mediapipe є ще однією з таких бібліотек (https://developers.google.com/mediapipe/solutions/vision/gesture_recognizer/python).

Встановіть бібліотеку Mediapipe через pip:

```
python -m pip install mediapipe
```

Наступний лістинг коду демонструє можливості обробки жестів.

Для роботи даної програми попередньо необхідно завантажити модель даних за посиланням (http://surl.li/qntfq) та допоміжний модуль helpers (http://surl.li/qntfh):

```
import numpy as np
```

```
import cv2
import mediapipe as mp
from helpers import draw_landmarks
from mediapipe.tasks import python
from mediapipe.tasks.python import vision
MODEL_PATH = 'week3\gesture_recognizer.task'
BaseOptions = mp.tasks.BaseOptions
GestureRecognizer = mp.tasks.vision.GestureRecognizer
GestureRecognizerOptions =
mp.tasks.vision.GestureRecognizerOptions
VisionRunningMode = mp.tasks.vision.RunningMode

def render_frame(result, output_image, timestamp_ms):
    frame = draw_landmarks(output_image.numpy_view(),
result)
    if result.gestures:
        for gesture in result.gestures:
            print(gesture[0].category_name)

options = GestureRecognizerOptions(
    base_options=BaseOptions(model_asset_path=MODEL_PATH),
    running_mode=VisionRunningMode.LIVE_STREAM,
    result_callback=render_frame
    )
timestamp = 0
WIDTH = 800
HEIGHT = 600

cap = cv2.VideoCapture(1)
with GestureRecognizer.create_from_options(options) as
recognizer:
    while True:
        ret, frame = cap.read()
        cv2.imshow("Frame", frame)
        mp_image =
mp.Image(image_format=mp.ImageFormat.SRGB, data=frame)
        recognizer.recognize_async(mp_image, timestamp)
        timestamp +=1
        if cv2.waitKey(1) == ord('q'):
            break
cap.release()
cv2.destroyAllWindows()
```

ПОСЛІДОВНІСТЬ ВИКОНАННЯ РОБОТИ

1. Ознайомитися з особливостями роботи бібліотеки OpenCV з відеопотоком.
2. Дослідити запропоновані в роботі лістинги з кодом та надати коментарі до коду.
3. Запропонувати власне рішення коду для роботи із жестами.
4. Наведіть приклади, де було б корисно застосовувати мову жестів у робототехніці. Як це можна імплементувати?
5. Дослідити документацію бібліотеки Mediapipe.

Теми індивідуального завдання

1. Застосування колісного робота для переміщення вантажів у межах населеного пункту.
2. Застосування колісного робота для переміщення вантажів у межах будівельного майданчика.
3. Застосування колісних роботів для роботи на відкритих майданчиках логістичних центрів.
4. Застосування колісних роботів усередині приміщень.
5. Застосування квадрокоптерів для переміщення вантажів між міськими когломераціями.
6. Застосування квадрокоптерів для зовнішнього сканування об’єктів.
7. Застосування квадрокоптерів для внутрішнього сканування об’єктів.
8. Наземні системи для сканування територій та алгоритми їхньої роботи.
9. Повітряні системи для сканування територій та алгоритми їхньої роботи.
10. Застосування доповненої реальності в будівництві. Відомі рішення і бібліотеки.
11. Концепції «розумних» будинків. Будова та програмна реалізація.
12. Застосування нейронних мереж для комп’ютерного зору. Відомі приклади застосування.

13. Інформаційні системи для картографування території.
14. Застосування CoppeliaSim в робототехніці.
15. ROS – псевдоопераційна система роботів.
16. Технології 3D-друку в будівництві. Засоби і алгоритми.
17. Технології smart-виробництва.
18. Алгоритм і застосування фільтра частинок.
19. Локалізація роботів за методом Монте-Карло.
20. Технології комунікації «розумних» речей (IoT).
21. Модульні апаратно-програмовані роботизовані платформи.
22. Задачі управління роботами-маніпуляторами.
23. Методи і алгоритми розпізнання зображень.
24. Штучний інтелект в робототехніці для роботи із відеопотоками.
25. Нейронні мережі на прикладі TensorFlow для розв’язку задач класифікації.
26. Нейронні мережі на прикладі TensorFlow для розв’язку задач регресії.
27. Застосування «клієнт-серверної архітектури» в системах управління роботів.
28. Застосування нечіткої логіки в системах управління роботів.
29. Алгоритми та реалізації задачі переміщення робота по лінії.
30. Способи реалізації управління на основі скінченних множин станів.

СПИСОК ЛІТЕРАТУРИ

1. *Артюх О.М.* Основи мехатроніки : навч. посіб. / О.М. Артюх, О.В. Дударенко, В. В. Кузьмін та ін. – Запоріжжя : НУ «Запорізька політехніка», 2021. – 372 с.
2. *Васильєв О.* Програмування мовою Python / О. Васильєв. – Львів : навчальна книга. – Богдан, 2019. – 504 с.
3. *Danny Staple*. Learn Robotics Programming Build and control AI-enabled autonomous robots using the Raspberry Pi and Python / *Staple Danny.* Packt Publishing, 2021. – 602 p.
4. *Клетте Р.* Комп'ютерний зір. Теорія і алгоритми / Р. Клетте. Пер. з англ. – АМК: Springer, 2019. – 506 с.
5. *Міщук Д.О.* Програмування робототехнічних інформаційних систем : конспект лекцій / Д.О. Міщук, В.П. Рашківський. – Київ : КНУБА, – 2024. – 216 с.
6. *Поліщук М.М.* Робототехнічні системи : проєктування і моделювання [Електронний ресурс]: навч. посіб. / М.М. Поліщук, М.М. Ткач. – Київ : КПІ ім. Ігоря Сікорського, 2021. – 112 с.
7. *Peter Corke*. Machine Vision Toolbox for MATLAB / *Corke Peter*, Au: 2015. – 242 p.
8. *Цвіркун Л.І.* Робототехніка та мехатроніка : навч. посіб. для студ. вищ. навч. закл. / Л.І. Цвіркун. – Дніпропетровськ : НГУ, 2010. – 289 с.

ДЛЯ НОТАТОК

Навчально-методичне видання

ПРОГРАМУВАННЯ РОБОТОТЕХНІЧНИХ ІНФОРМАЦІЙНИХ СИСТЕМ

Методичні вказівки
до виконання самостійної роботи та індивідуального завдання
для здобувачів першого (бакалаврського) рівня вищої освіти
спеціальності F6 «Інформаційні системи і технології»

Укладачі: МІЩУК Дмитро Олександрович,
РАШКІВСЬКИЙ Володимир Павлович

Випусковий редактор *Л. С. Тавлуй*
Комп'ютерне верстання *К. А. Мавроді*

Підписано до друку 12.12.2025. Формат 60 х $84_{1/16}$
Ум. друк. арк. 5,11. Обл.-вид. арк. 5,5.
Електронний документ. Вид. № 93/ІІІ-25

Видавець і виготовлювач:
Київський національний університет будівництва і архітектури

Проспект Повітряних Сил, 31, Київ, Україна, 03037

Свідоцтво про внесення до Державного реєстру суб'єктів
видавничої справи ДК № 808 від 13.02.2002