

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»

**ОСНОВИ ПОБУДОВИ СУЧАСНИХ МОБІЛЬНИХ СИСТЕМ
ТЕХНІЧНОГО ЗОРУ
Частина 2
ОБРОБЛЕННЯ ЗОБРАЖЕНЬ І ВІДЕОДАНИХ**

Навчальний посібник

Харків «ХАІ» 2018

УДК 004.942(075.8)

K78

Колектив авторів:

Л. О. Краснов, К. Ю. Дергачов, С. В. Багінський, Є. В. Пявка

Рецензенти: канд. техн. наук В. О. Кочура,
канд. техн. наук С. М. Флерко

Краснов, Л. О.

K78 Основи побудови сучасних мобільних систем технічного зору [Текст] : навч. посіб. Ч. 2. Оброблення зображень і відеоданих / Л. О. Краснов, К. Ю. Дергачов, С. В. Багінський, Є. В. Пявка. – Харків : Нац. аерокосм. ун-т ім. М. Є. Жуковського «Харків. авіац. ін-т», 2018. – 92 с.

ISBN 978-966-662-581-9

Подано матеріали з теоретичного й практичного вивчення методів побудови сучасних систем технічного зору. Розглянуто питання застосування мови програмування Python спільно з бібліотекою для оброблення зображень OpenCV на базі мікрокомп'ютерів Raspberry Pi з метою вирішення завдань оброблення даних і керування процесом відеоспостереження. Наведено приклади практичної роботи з реальними зображеннями й даними відеокамер.

Для студентів, що навчаються за напрямками підготовки «Авіоніка», «Аеронавігація» і «Системна інженерія» спеціальностей «Системи управління літальними апаратами і комплексами», «Комп'ютеризовані системи управління і автоматика», «Аеронавігаційні системи і системи аеронавігаційного обслуговування».

Іл. 51. Табл. 11. Бібліогр.: 19 назв

УДК 004.942(075.8)

© Колектив авторів, 2018

© Національний аерокосмічний
університет ім. М. Є. Жуковського

«Харківський авіаційний інститут», 2018

ISBN 978-966-662-581-9

ВСТУП

Створення сучасних мобільних систем технічного зору потребує від розробників уже на етапі ескізного проектування вирішення цілого комплексу завдань з обґрунтування й вибору апаратної платформи, комп'ютера й засобів програмування.

Ці завдання було досить детально розглянуто в першій частині пропонуваної роботи. На сучасному етапі оптимальним з огляду на ефективність роботи, мобільність і вартість системи є використання одноплатової комп'ютерної платформи Raspberry Pi з базовою операційною системою Raspbian і мови програмування Python.

У другій частині навчального посібника детально досліджуються питання використання основних ресурсів оброблення зображень і відео в системах технічного зору. Розглядається програмування на мові Python основних алгоритмів оброблення зображень з використанням бібліотеки Pillow (Python Imaging Library), яка забезпечує необхідний набір засобів для різного роду перетворень графічної інформації.

Велика увага приділяється підключенню й роботі в програмі Python бібліотеки оброблення зображень OpenCV (Open Source Computer Vision Library). Використання в'язки Python і бібліотеки OpenCV дає змогу проводити оброблення відеоінформації в реальному масштабі часу із застосуванням найсучасніших алгоритмів оброблення відеоданих.

Подаються детальні рекомендації щодо оброблення зображень і відеоінформації як при використанні операційної системи Windows при роботі на звичайному комп'ютері, так і операційної системи Raspbian при роботі на мікрокомп'ютері Raspberry Pi.

Інтерес до цієї проблеми постійно підвищується, а актуальність проведення нових наукових досліджень і створення нових інженерних розробок в області технічного зору неухильно зростає. Тому всі ці питання й низка інших питань, викладених у цій роботі, поза сумнівом, будуть корисними для читача при самостійній роботі.

1. ОСНОВНІ РЕСУРСИ ОБРОБЛЕННЯ ЗОБРАЖЕНЬ І ВІДЕО В СИСТЕМАХ ТЕХНІЧНОГО ЗОРУ

Сучасні системи технічного зору орієнтовані на використання мови програмування **Python** на базі операційної системи **Windows** при їх стаціонарному розміщенні. У мобільних системах технічного зору, що базуються на рухомих носіях, зазвичай застосовують одноплатову платформу **Raspberry Pi** з рекомендованою виробником операційною системою **Raspbian**. При цьому базовою мовою програмування також є **Python**.

Основні завдання технічного зору з оброблення зображень і відеоданих при програмуванні на мові **Python** прийнято вирішувати з використанням бібліотеки **Pillow (Python Imaging Library)**, що забезпечує досить повний набір функцій і методів оброблення зображень і відео. Проте найбільш ефективним є спільне використання ресурсів бібліотек **Pillow** і **OpenCV (Open Source Computer Vision Library)**.

Експлуатація **OpenCV** у в'язці з **Python** є зручною для створення простих і зрозумілих додатків, проведення експериментів і синтезу різних прототипів. На мові **Python** практично вся функціональність бібліотеки **OpenCV** є доступною. При використанні векторно-матричної ідеології, аналогічної прийнятому підходу в **Matlab**, можна отримувати досить швидкі коди, що забезпечують вирішення завдань у реальному масштабі часу. Для цього слід уникати оброблення зображень попіксельно та інших масивів поелементно в циклах. Замість цього доцільно використовувати векторні функції, які працюють відразу зі всім зображенням або масивом.

Далі в цьому розділі розглянемо основні властивості бібліотеки **OpenCV** і методику її підключення до **Python**.

1.1. Загальна інформація про бібліотеку OpenCV

OpenCV – це комп'ютерний зір з відкритим вихідним кодом і програмна бібліотека машинного навчання. **OpenCV** створено для забезпечення загальної інфраструктури додатків комп'ютерного зору й прискорення використання машинного сприйняття в комерційних продуктах. Будучи ліцензованим продуктом, **OpenCV** спрощує використання і модифікацію коду.

Основні відомості про характеристики бібліотеки розміщено на сайті <http://opencv.org>., де можна завантажити бібліотеку для різних платформ, знайти повну документацію стосовно бібліотеки, короткі інструкції щодо встановлення (**Quick Start**), уроки з її використання (**Tutorials**), довідкову карту (**Cheat Sheet**), а також посилання на корисні сайти й т. ін.

На сайті <http://docs.opencv.org> зібрано всю документацію, зокрема, детальні описи всіх функцій з повнотекстовим пошуком. На сайті <http://answers.opencv.org> можна ставити запитання, що стосуються бібліотеки, та отримувати на них відповіді від розробників або інших користувачів.

У бібліотеці зібрано більше 2500 оптимізованих алгоритмів, що являють собою повний набір класичних і сучасних алгоритмів комп'ютерного зору й машинного навчання. Ці алгоритми можна використовувати для виявлення й розпізнавання осіб, ідентифікації об'єктів, класифікації дій людини у відео, відслідковування руху камери й рухомих об'єктів, створення 3D-моделей сцени при роботі зі стереокамерами, зшивання зображень разом для одержання високої роздільності.

Бібліотека **OpenCV** має інтерфейси **C ++**, **C**, **Python**, **Java** і **Matlab**, підтримує **OC Windows**, **Linux**, **Android** і **Mac OS** та орієнтується в основному на додатки в режимі реального часу. Сьогодні активно розвиваються повнофункціональні інтерфейси **CUDA** і **OpenCL**. Існує більше 500 алгоритмів і 10-кратна кількість функцій, які складають або підтримують ці алгоритми. Бібліотека **OpenCV** написана на мові **C ++**, має шаблонний інтерфейс.

На першому етапі бібліотека розроблялася як набір великих універсальних модулів (**cxcore**, **Cvaux**, **Highgui**, **Cvaux**).

Проте у версії 2.2 структуру бібліотеки було реорганізовано – тепер великі модулі бібліотеки **OpenCV** замінені на менші за функціональним принципом:

- `opencv_core` – ядро: базові структури, обчислення (математичні функції, генерація псевдовипадкових чисел, DFT, DCT, введення/виведення в XML і т. ін.);
- `opencv_imgproc` – оброблення зображень (фільтри, перетворення й т. ін.);
- `opencv_highgui` – простий UI (завантаження/зберігання зображень і відео);
- `opencv_ml` – методи й моделі машинного навчання (SVM, для прийняття рішень і т. д.);
- `opencv_features2d` – різні дескриптори (SURF);
- `opencv_video` – аналіз руху й відслідковування об'єктів (оптичний потік, шаблони руху, видалення фону);
- `opencv_objdetect` – детектування об'єктів на зображенні (вейвлети Хаару, HOG і т. д.);
- `opencv_calib3d` – калібрування камери, пошук стереовідповідності й елементів оброблення тривимірних даних;
- `opencv_flann` – бібліотека швидкого пошуку найближчих сусідів (FLANN);
- `opencv_contrib` – супутній код, ще не готовий для застосування;
- `opencv_legacy` – попередній код, збережений заради зворотної сумісності;
- `opencv_gpu` – прискорення деяких функцій **OpenCV** з допомогою CUDA (NVidia).

Для початку роботи з бібліотекою **OpenCV** зазвичай рекомендують використовувати її в'язку з мовою **Python** (<http://python.org>). Це створює багато зручностей при реалізації проектів. Програму на цій мові можна писати в будь-якому текстовому редакторі. Далі без попередньої компіляції вона виконується інтерпретатором **Python**. Мова **Python** з бібліотекою **Numpy** перетворюється на безкоштовний аналог системи **Matlab**, при цьому сама мова є розвиненішою, ніж **Matlab**. Ще одна перевага мови **Python** полягає в її хорошій документованості. В інтерактивній сесії **Python** (у командному вікні) можна швидко отримати список доступних функцій і методів конкретного класу, довідку про кожну функцію **OpenCV**, наприклад:

```
>>> import cv2 as cv
>>> dir(cv) # Список функцій і класів у cv
>>> cv.resize.__doc__ # Опис функції resize
>>> cap = cv.VideoCapture(0)
>>> dir(cap) # Список методів VideoCapture
>>> cap.read.__doc__ # Опис методу read
```

Перед початком роботи рекомендується ознайомитися з набором прикладів на мові **Python** з теки <Opencv>/samples/python3. Це усунення розмиття внаслідок руху камери, розпізнавання рукописних цифр, відеоефекти й т. д.

1.2. Підключення бібліотеки OpenCV до Python для ОС Windows

Питання підключення модулів і бібліотек до **Python** для **Windows** досить детально описано в першій частині цієї роботи «Комп'ютери і засоби програмування». Нагадаємо лише, що для установлення модулів і бібліотек при використанні операційної системи **Windows** насамперед необхідно перевірити розрядність системи (– **x64** чи – **x86**), а потім скачати вибрану бібліотеку, урахувавши розрядність операційної системи й раніше встановлену версію **Python**. Там же як приклад дуже детально описано процедуру підключення бібліотеки **Pillow**, що є основним ресурсом **Python** для оброблення зображень і відеоінформації.

Для встановлення бібліотеки **OpenCV** далі необхідно зайти в системну теку «**Python**», куди раніше встановлювалася програма, наприклад, указавши такий шлях: «C:\users\user\appdata\local\Programs\python\python36-32». У командному рядку ввести команду «cmd» і натиснути кнопку **Enter** (рис. 1.1). Після цього відкриється вікно адміністратора (рис. 1.2, а).

У вікні адміністратора за допомогою вбудованого пакета **PIP** необхідно додати раніше скачану бібліотеку **OpenCV** з допомогою команди «install» і підтвердити це натисненням кнопки **Enter**. Тоді в цьому вікні з'явиться повідомлення про успішну інсталяцію потрібної бібліотеки

(рис. 1.3, б). Технологію роботи з алгоритмами бібліотеки **OpenCV** для оброблення зображень у **Python** буде описано далі.

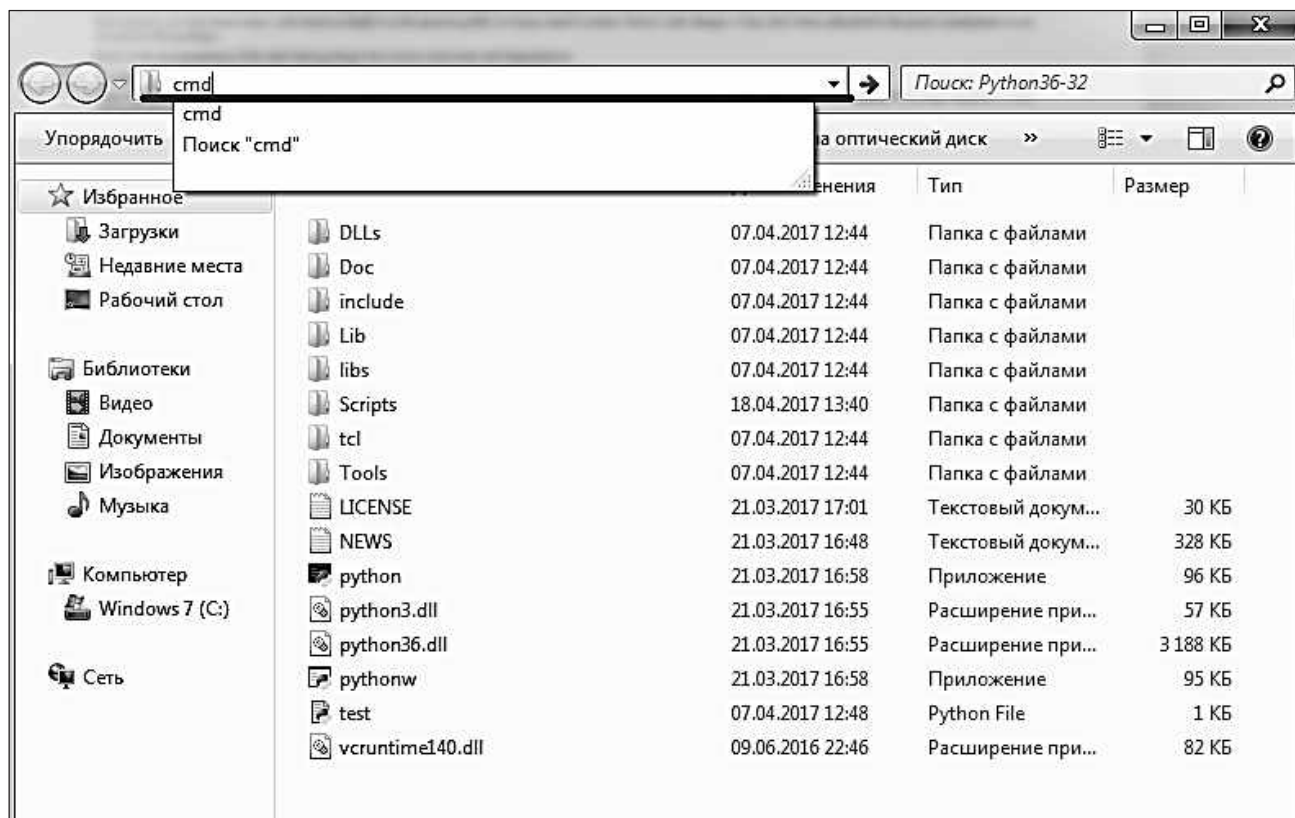
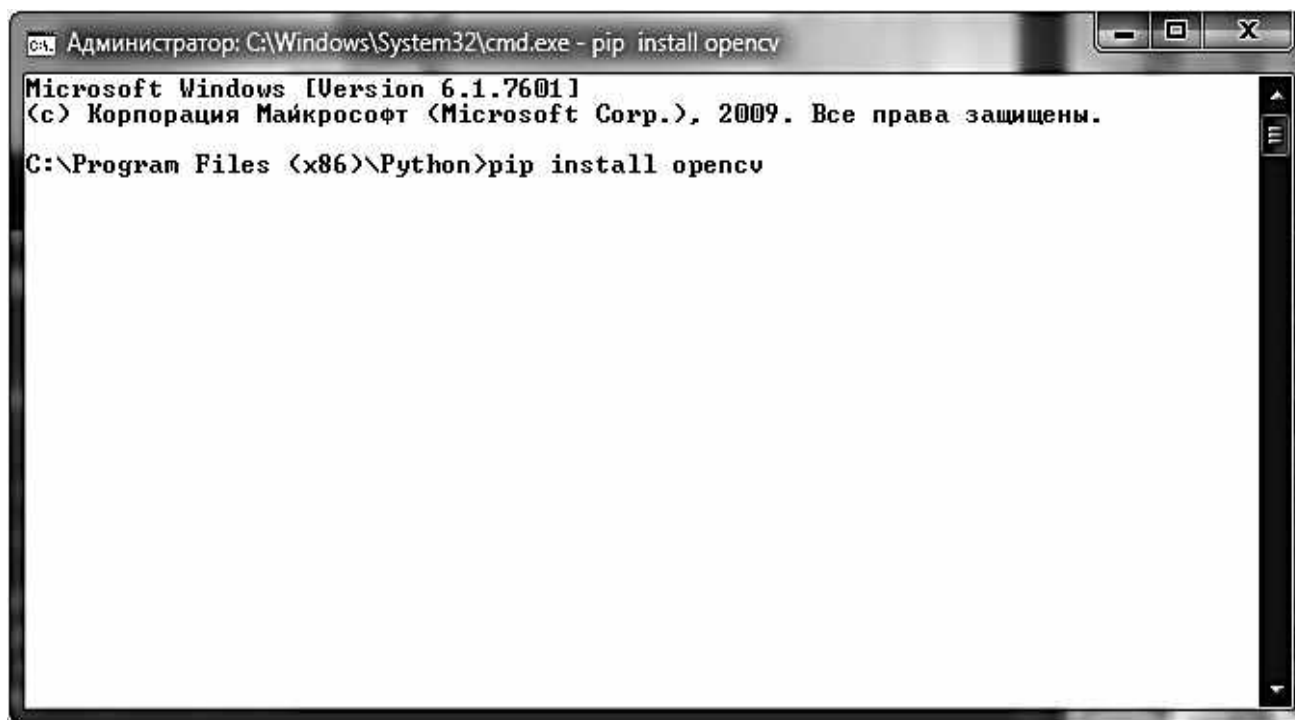
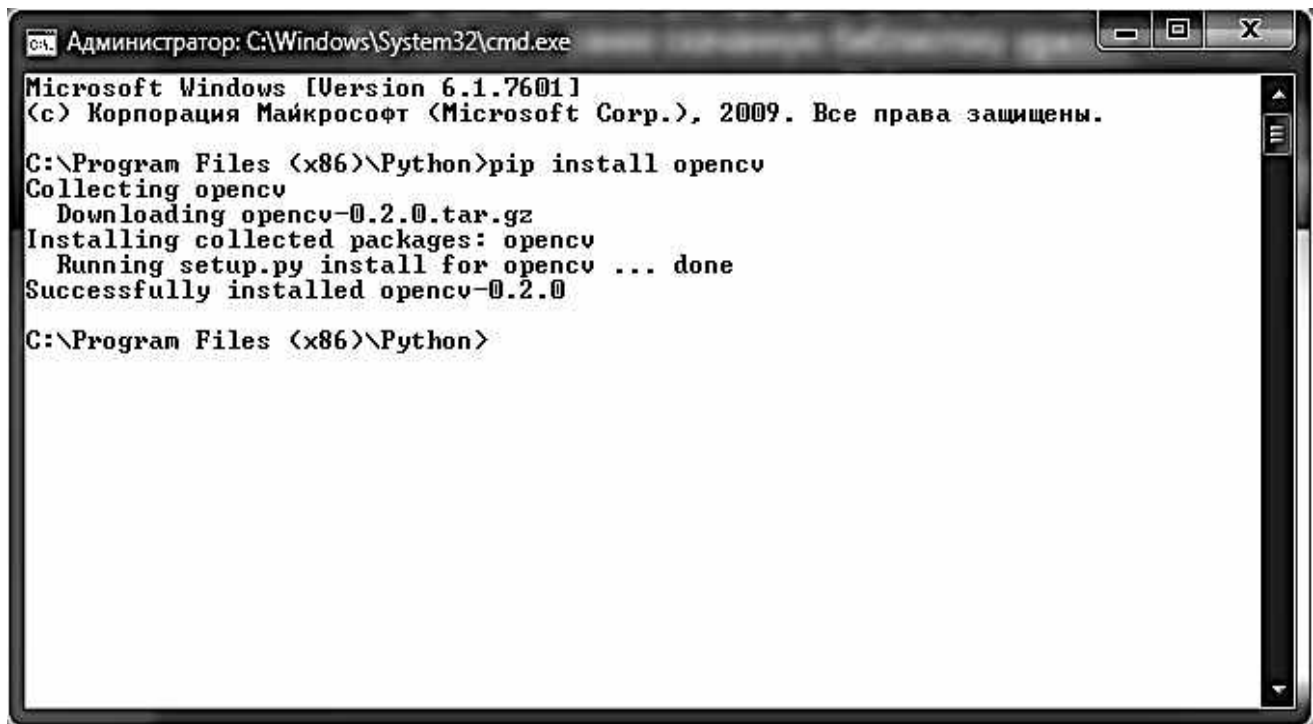


Рис. 1.1. Вибір у системній теці «**Python**»



а

Рис. 1.2. Вікно адміністратора Windows



```
Администратор: C:\Windows\System32\cmd.exe
Microsoft Windows [Version 6.1.7601]
(c) Корпорация Майкрософт (Microsoft Corp.), 2009. Все права защищены.

C:\Program Files (x86)\Python>pip install opencv
Collecting opencv
  Downloading opencv-0.2.0.tar.gz
Installing collected packages: opencv
  Running setup.py install for opencv ... done
Successfully installed opencv-0.2.0

C:\Program Files (x86)\Python>
```

б

Рис. 1.2. Закінчення

1.3. Підключення OpenCV до одноплатового комп'ютера Raspberry Pi

Пройшовши всі етапи встановлення операційної системи й першого запуску програм операційної системи **Raspbian** і **Python 3**, слід установити необхідні бібліотеки для роботи зі всіма модулями, що цікавлять нас, і бібліотеками для вирішення конкретних завдань. Детальну методику встановлення операційної системи й мови програмування на комп'ютер **Raspberry Pi 2** також детально описано в першій частині цієї роботи «Комп'ютери і засоби програмування».

Модулі, які необхідно підключити під час встановлення бібліотек:

- **NumPy** – фундаментальний пакет для наукових обчислень з **Python**;
- **Pillow (Python Imaging Library)** – основна бібліотека **Python** для роботи із зображеннями;
- **OpenCV** – алгоритми комп'ютерного зору з відкритим вихідним кодом і програмна бібліотека машинного навчання.

Звертаємо Вашу увагу на важливість процедур встановлення перелічених бібліотек і пакетів. У зв'язку з цим рекомендуємо покроково виконати інструкції щодо їх інсталяції.

Крок 1. Встановлення й оновлення операційної системи виконується з допомогою двох команд, прописаних у командному рядку Terminal:

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

Крок 2. Далі встановлюється пакет **cMake** (крос-платформена автоматизована система складання проектів):

```
$ sudo apt-get install build-essential cmake pkg-config
```

Окрім цього також необхідно встановити пакети з відомими кодеками для читання й відображення графічних і відеофайлів з дозволом (таких, як jpeg, tiff, jasper, png, vidcore, x264):

```
$ sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev
```

```
$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
```

```
$ sudo apt-get install libxvidcore-dev libx264-dev
```

Для створення простих екранних форм необхідно прописати:

```
$ sudo apt-get install libgtk2.0-dev
```

Для прискорення роботи з матрицями й матричними обчисленнями необхідно встановити спеціальний пакет:

```
$ sudo apt-get install libatlas-base-dev gfortran
```

Крок 3. Дуже важливим є скачування **OpenCV** з репозитарію. Для цього необхідно:

- перейти в теку Downloads:

```
$ cd ~/Downloads
```

- завантажити архів за допомогою команди `wget` і розархівувати його за допомогою команди `unzip`:

```
$ wget -O opencv.zip  
(https://github.com/opencv/opencv/archive/master.zip)  
$ unzip opencv.zip
```

- скачати другий архів з доповненнями й додатками до **OpenCV**:

```
$ wget -O opencv_contrib.zip  
https://github.com/opencv/opencv_contrib/archive/master.zip  
$ unzip opencv_contrib.zip
```

- встановити пакет `numpy`:

```
$ pip install numpy
```

Крок 4. Необхідно зайти в теку з вихідними кодами **OpenCV** і виконати такі команди:

```
$ cd ~/Downloads/opencv-master  
$ mkdir build  
$ cd build  
$ cmake -D CMAKE_BUILD_TYPE=RELEASE\  
-D CMAKE_INSTALL_PREFIX=/usr/local\  
-D INSTALL_PYTHON_EXAMPLES=ON\  
$ make  
$ sudo make install
```

```
-D OPENCV_EXTRA_MODULES_PATH=~Downloads/opencv_contrib-master/modules
-D BUILD_EXAMPLES=ON ..
```

Далі, не виходячи з поточної теки, необхідно запустити команду `make`:

```
$ make -j4
```

Використання розширення `j4` означає, що складається пакет із використанням усіх чотирьох ядер **Raspberry Pi**. Це сильно прискорює процедуру, але навіть за таких умов складання триває майже півтори години.

Крок 5. Далі необхідно встановити складений бінарний пакет у теку з **Python**:

```
$ sudo make install
```

```
$ sudo ldconfig
```

Крок 6. Після встановлення пакета слід переконатися, що необхідні модулі бібліотеки **OpenCV** уже встановлені й готові до роботи з **Python**.

Перевірку можна провести таким чином. Для оцінювання правильності встановлення необхідно запустити інтерпретатор мови **Python** і написати команду імпорту модуля `cv2`, програма поверне дані про версію бібліотеки **OpenCV**:

```
$ workon cv
$ python
>>> import cv2
>>> cv2.__version__
'3.1.0'
>>>
```

Слід зазначити, що успішне встановлення програмного забезпечення **Python** і підключення бібліотеки **OpenCV** на платформу **Raspberry Pi** гарантується лише при використанні SD-карти для завантаження й підтримки роботи операційної системи й програмного забезпечення місткістю не менше 16 Гб. А при роботі з алгоритмами бібліотеки **OpenCV** слід використовувати SD-карту місткістю 32 Гб, що забезпечить необхідну швидкодію оброблення відеоданих.

Наприкінці цього розділу зазначимо деякі особливості використання бібліотеки **OpenCV** у взаємодії з мовою програмування **Python** на базі одноплатової платформи **Raspberry Pi** з операційною системою **Raspbian**, що рекомендується виробником. Сьогодні така композиція апаратно-програмних засобів дає змогу оптимально вирішувати безліч завдань технічного зору. В'язка **OpenCV + Python** найбільш продуктивною є при вирішенні завдань оброблення відеоданих у реальному часі на рухомих носіях (роботах і безпілотних ЛА).

2. ВИКОРИСТАННЯ БІБЛІОТЕКИ PILLOW ПРИ ОБРОБЛЕННІ ЗОБРАЖЕНЬ

Як зазначалося раніше, одним із основних завдань побудови систем технічного зору є оброблення зображень і відеоінформації. На мові програмування **Python** це здійснюється за допомогою бібліотеки **Pillow**, що підключається. Для ефективного використання наявних ресурсів розглянемо структуру цієї бібліотеки, властивості пропонованих користувачеві функцій і методів. Офіційну документацію щодо бібліотеки **Pillow** розміщено за адресою <http://pillow/readthedocs.org/>.

2.1. Основні модулі пакета Pillow

Розглянемо основні модулі пакета **Pillow** для роботи із зображеннями на мові **Python**. Переклад бібліотеки **Python Imaging Library (PIL)** зроблено за офіційною документацією. Бібліотека містить кілька найнеобхідніших модулів, наведених у табл. 2.1.

Таблиця 2.1

Найменування модуля	Уміст модуля
The Image Module	Функції, методи та їх властивості для відкриття, зберігання й маніпулювання зображеннями
The ImageChops Module	Багато арифметичних операцій над зображеннями
The ImageColor Module	Функції для перетворення рядка визначення кольору на кортеж формату RGB
The ImageDraw Module	Модуль для простої 2D-графіки, створення нових зображень і тексту, ретушування зображень
The ImageGrab Module	Функції, які допомагають зробити знімок екрана
The ImageFont Module	Функціонал для роботи зі шрифтами TrueType і OpenType

Модуль ***The Image Module*** містить функції, методи та їх властивості для відкриття, зберігання й аналізу зображень. Основні функції цього модуля наведено в табл. 2.2. Далі ці функції й методи будуть описані детальніше й проілюстровані прикладами коду в ***Python***.

Таблиця 2.2

Ім'я і синтаксис функції	Застосування функції
new – <code>Image.new(mode, size, color)</code> ⇒ зображення	Створення нового зображення з заданими режимом і розміром
Open – <code>Image.open(infile, mode)</code> ⇒ зображення	Відкриття й ідентифікація файла зображення
blend – <code>Image.blend(image1, image2, alpha)</code> ⇒ зображення	Створення нового зображення шляхом інтерполяції між заданими зображеннями за допомогою постійної «альфа»
composite – <code>Image.composite(image1, image2, mask)</code> ⇒ зображення	Створення нового зображення шляхом інтерполяції між заданими зображеннями з використанням такої маски, як «альфа»
eval – <code>Image.eval(image, function)</code> ⇒ зображення	Застосування функції для кожного пікселя в заданому зображенні
frombuffer – <code>Image.frombuffer(mode, size, data)</code> ⇒ зображення	Створення образу пам'яті з піксельних даних у рядку або буфері з допомогою стандартного «сирого» декодера
fromstring – <code>Image.fromstring(mode, size, data)</code> ⇒ зображення	Створення образу пам'яті з піксельних даних у рядку або буфері з допомогою стандартного «сирого» декодера
merge – <code>Image.merge(mode, bands)</code> ⇒ зображення	Створення нового зображення з декількох окремих зображень

Основні методи модуля *The Image Module* наведено в табл. 2.3.

Таблиця 2.3

Ім'я та синтаксис методу	Застосування методу
convert – <code>im.convert(mode, matrix)</code> ⇒ зображення	Повертання перетвореної копії зображення. Перетворення зображення RGB на L або RGB з допомогою матриці перетворення
copy – <code>im.copy()</code> ⇒ зображення	Копіювання зображення
crop – <code>im.crop(box)</code> ⇒ зображення	Повертання прямокутної області з поточного зображення
draft – <code>im.draft(mode, size)</code>	Налаштування завантажувача файлів зображень для того, щоб він повертав версію зображення, яка точніше відповідає заданим режиму й розміру
filter – <code>im.filter(filter)</code> ⇒ зображення	Повертання копії зображення з допомогою заданого фільтра
fromstring – <code>im.fromstring(data, decoder, parameters)</code>	Завантаження даних у поточне зображення
getbands – <code>im.getbands()</code> ⇒ кортеж рядків	Повертання кортежу, що містить імена всіх груп
getbbox – <code>im.getbbox()</code> ⇒ 4-елементний кортеж или None	Обчислення прямокутника, який обмежує ненульові області в зображенні
getcolors – <code>im.getcolors()</code> ⇒ список кортежів (кількість, колір) або None	Повертання несортованого списку кортежів (кількість, колір), де підраховується, скільки разів відповідний колір трапляється в зображенні
getdata – <code>im.getdata()</code> ⇒ послідовність	Повертання зображення у вигляді послідовності, що містить значення пікселів
getextrema – <code>im.getextrema()</code> ⇒ 2-елементний кортеж	Повертання 2-елементного кортежу, що містить мінімальне й максимальне значення зображення
getpixel – <code>im.getpixel(xy)</code> ⇒ значення або кортеж	Повертає значення пікселя в певній позиції

Ім'я та синтаксис методу	Застосування методу
histogram – <code>im.histogram()</code> ⇒ список <code>im.histogram(mask)</code> ⇒ список	Повертання гістограми зображення. Гістограма повертається як список пікселів. Повертання гістограми для тих частин зображення, де зображення маски не дорівнює нулю
load – <code>im.load()</code>	Завантаження пікселів з файла зображення
paste – <code>im.paste(image, box)</code>	Вставлення одного зображення в інше. <code>Box</code> – це 2-елементний кортеж, що визначає верхній лівий кут, або 4-елементний кортеж, що визначає лівий верхній і правий нижній пікселі координат або <code>None</code> (те саме, що <code>(0, 0)</code>)
point – <code>im.point(table)</code> ⇒ зображення	Повертання копії зображення, де кожен піксель був зіставлений за заданою таблицею
putpalette – <code>im.putpalette(sequence)</code>	Палітра для P- або L-зображень
putpixel – <code>im.putpixel(xy, colour)</code>	Змінення кольору пікселя в певний позиції
resize – <code>im.resize(size, filter)</code> ⇒ зображення	Повертання копії зображення із зміненими розмірами
rotate – <code>im.rotate(angle, filter=NEAREST, expand=0)</code> ⇒ зображення	Створення копії зображення, поверненої на вказану кількість градусів проти годинникової стрілки навколо його центра
save – <code>im.save(outfile, format, options...)</code>	Зберігання зображення у файл
seek – <code>im.seek(frame)</code>	–
show – <code>im.show()</code>	Відображення зображення
split – <code>im.split()</code> ⇒ послідовність	Повертання кортежу з окремих смуг зображення
tell – <code>im.tell()</code> ⇒ число	Повертання номера поточного кадра
thumbnail – <code>im.thumbnail(size, filter).</code>	Змінення розміру самого зображення

Закінчення табл. 2.3

Ім'я та синтаксис методу	Застосування методу
tobitmap – <code>im.tobitmap()</code> ⇒ рядок	Повертання зображення, перетвореного в X11-растрове зображення
tostring – <code>im.tostring(encoder, parameters)</code> ⇒ рядок	Повертання рядка, що містить дані пікселів, шляхом кодування даних
transform – <code>im.transform(size, method, data, filter)</code> ⇒ зображення	Створення нового зображення із заданим розміром і таким самим режимом, що й оригінал
transpose – <code>im.transpose method)</code> ⇒ зображення	Повертання поверненої копії зображення. Метод може мати такий вигляд: <code>FLIP_LEFT_RIGHT</code> , <code>FLIP_TOP_BOTTOM</code> , <code>ROTATE_90</code> , <code>ROTATE_180</code> , <code>ROTATE_270</code> або число від 1 до 4 залежно від версій модуля
verify – <code>im.verify()</code>	Спроба визначити, чи пошкоджений файл, без фактичного декодування даних зображення

Основні властивості модуля **The Image Module** наведено в табл. 2.4.

Таблиця 2.4

Ім'я властивості	Характеристика властивості
Format – <code>im.format</code> ⇒ рядок або None	Формат вихідного файлу
Mode – <code>im.mode</code> ⇒ рядок	Режим зображення. Типові значення 1, L, RGB або CMYK
Size – <code>im.size</code> ⇒ (width, height)	Розмір зображення в пікселях. Розмір, указаний у вигляді 2-елементного кортежу (ширина, висота)
Palette – <code>im.palette</code> ⇒ палітра або None	Колірна палітра таблиці, якщо вона є
Info – <code>im.info</code> ⇒ словник	Словник даних, пов'язаних із зображенням

ImageChops. Модуль містить безліч арифметичних операцій над зображеннями. Вони можуть бути використані для різних цілей, у тому числі для спеціальних ефектів, зображення композиції, алгоритмічного живопису. Операції реалізовані лише для 8-бітових зображень (наприклад, L і RGB).

Функції модуля *Image Chops Module*. Основні арифметичні операції над зображеннями модуля наведено в табл. 2.5.

Таблиця 2.5

Найменування операції	Застосування операції
Constant – <code>ImageChops.constant (image, value) ⇒ зображення</code>	Повернення нового шару з таким розміром, як розмір указанного зображення, заповненого кольором, який задано значенням <code>value</code> (0 – 255)
Duplicate – <code>mageChops.duplicate (image) ⇒ зображення</code>	Копіювання зображення
invert – <code>ImageChops.invert (image) ⇒ зображення</code>	Інверсія зображення
Lighter – <code>ImageChops.lighter (image1, image2) ⇒ зображення</code>	Порівняння двох зображень, піксель за пікселем, і повертання нового зображення, що містить «легкі» кольори
Darker – <code>ImageChops.darker (image1, image2) ⇒ зображення</code>	Порівняння двох зображень, піксель за пікселем, і повертання нового зображення, що містить «темні» кольори
Difference – <code>ImageChops.difference (image1, image2) ⇒ зображення</code>	Повертання абсолютного значення різниці між двома зображеннями
Multiply – <code>ImageChops.multiply (image1, image2) ⇒ зображення</code>	Накладання двох зображень одне на одне
Screen – <code>ImageChops.screen (image1, image2) ⇒ зображення</code>	Накладання двох інвертованих зображень одне на одне
Add – <code>ImageChops.add (image1, image2, scale, offset) ⇒ зображення</code>	Додавання двох зображень, поділивши результат на масштаб і додавши зсув. Масштаб за умовчанням 1.0 і зсув 0.0
Subtract – <code>ImageChops.subtract (image1, image2, scale, offset) ⇒ зображення</code>	Віднімання двох зображень, поділивши результат на масштаб і додавши зсув. Масштаб за умовчанням 1.0 і зсув 0.0

Найменування операції	Застосування операції
Blend – <code>ImageChops.blend</code> (<code>image1, image2, alpha</code>) $\Rightarrow$ зображення	Те саме, що й функція <code>blend</code> у модулі <code>Image</code>
Composite – <code>ImageChops.composite</code> (<code>image1, image2, mask</code>) $\Rightarrow$ зображення	Те саме, що й функція <code>composite</code> в модулі <code>Image</code>

Модуль ***Imagecolor*** містить функції для перетворення рядка, що вказує колір, у RGB-кортеж. Цей модуль використовується в модулях ***Image.new*** і ***Image.draw***.

Модуль ***Imagecolor*** підтримує такі формати рядків:

- шістнадцяткові специфікатори кольору вигляду "#rgb" або "#rrggbb", наприклад: "#ff0000";
- RGB-функції вигляду "rgb (червоний, зелений, синій)", наприклад: "rgb(255, 0, 0)" і "rgb(100%, 0%, 0%);
- HSL-функції (тон – насиченість – яскравість), задані як "HSL (тон, насиченість %, яскравість %)", де тон кольору подається як кут від 0° до 360°, насичення – як значення від 0 % до 100 %, яскравість – як значення від 0 % до 100 %, наприклад: "HSL(0, 100%, 50%)";
- HTML-назви кольорів (модуль ***Imagecolor*** підтримує близько 140 стандартних назв кольорів), наприклад: "red" або "white".

Функції модуля *The Imagecolor Module*. Основні арифметичні операції над зображеннями цього модуля наведено в табл. 2.6.

Таблиця 2.6

Ім'я та синтаксис функції	Застосування функції
getrgb – <code>getrgb(color)</code> $\Rightarrow$ (червоний, зелений, синій)	Перетворення рядка, що позначає колір, на RGB-кортеж
getcolor – <code>getcolor(color, mode)</code> $\Rightarrow$ (червоний, зелений, синій) або ціле число	Те саме, що й функція <code>getrgb</code> , але можна задати режим "RGB", "RGBA" або "L"

Модуль ***The Imagedraw Module*** містить інструменти для простої 2D-графіки. Формат кольору такий же, як у модулі ***Imagecolor***.

Шрифти – PIL. Можна використовувати растрові шрифти або шрифти `Opentype/truetype`. Для завантаження шрифтів `Opentype/truetype` слід використовувати функції ***Truetype*** з модуля ***Imagefont***.

Функція ***Draw(image)*** $\Rightarrow$ об'єкт `Draw` створює об'єкт, який можна використовувати для рисування в заданому зображенні.

Основні методи модуля ***The Imagedraw Module*** наведено в табл. 2.7.

Таблиця 2.7

Ім'я та назва методу	Застосування методу
arc – draw.arc(xy, start, end, fill=None)	Рисування дуги між початковими й кінцевими кутами всередині заданої обмежувальної рамки
bitmap – draw.bitmap(xy, bitmap, fill=None)	Рисування зображення (маски) на заданій позиції з використанням поточного кольору заливки
chord – draw.chord(xy, start, end, fill=None, outline=None)	Те саме, що й дуга, але з'єднує кінці прямою лінією
ellipse – draw.ellipse(xy, fill=None, outline=None)	Рисування еліпса
line – draw.line(xy, fill=None, width=0)	Рисування лінії. Аргумент може бути списком, кортежем або списком кортежів
pieslice – draw.pieslice(xy, start, end, fill=None, outline=None)	Те саме, що й дуга, але рисує прямі лінії між кінцевими точками і в центрі обмежувальної рамки
point – draw.point(xy, fill=None)	Рисування точки (окремих пікселів) на заданій координаті
polygon – draw.polygon(xy, fill=None, outline=None)	Рисування багатокутника
rectangle – draw.rectangle(box, fill=None, outline=None)	Рисування прямокутника
text – draw.text(position, string, fill=None, font=None)	Рисування рядка в заданій позиції
textsize – draw.textsize(string, font=None) ⇒ (width, height)	Повертання розміру рядка в пікселях

Модуль The Imagegrab Module може бути використаний для копіювання екрана (скріншот). Поточна версія працює лише в ***Windows***. Функції цього модуля наведено в табл. 2.8.

Таблиця 2.8

Ім'я та синтаксис функції	Застосування функції
grab – <code>ImageGrab.grab(BBOX)</code> ⇒ зображення	Знімання екрана й повертання RGB-зображення. При використанні з аргументом можна копіювати лише частину екрана. BBOX – це кортеж, що визначає координати лівого верхнього пікселя, ширину й висоту
grabclipboard – <code>ImageGrab.grabclipboard()</code> ⇒ зображення або список рядків або None	Знімання вмісту буфера обміну й повертання зображення або списку імен файлів. Якщо буфер обміну не містить даних, то функція повертає None

Модуль *The Imagefont Module* використовується для роботи зі шрифтами *Truetype* і *Opentype*. Цей модуль можна використовувати разом з модулем *Imagedraw*:

```
import ImageFont, ImageDraw
draw = ImageDraw.Draw(image)
# use a bitmap font
font = ImageFont.load("arial.pil")
draw.text((10, 10), "hello", font=font)
# use a truetype font
font = ImageFont.truetype("arial.ttf", 15)
draw.text((10, 25), "world", font=font)
```

Функції модуля ***The Imagefont Module*** наведено в табл. 2.9.

Таблиця 2.9

Ім'я та синтаксис функції	Застосування функції
load – <code>ImageFont.load(file)</code> ⇒ об'єкт шрифту	Завантаження шрифту з указанного файла й повертання відповідного об'єкта шрифту
load_path – <code>ImageFont.load_path(file)</code> ⇒ об'єкт шрифту	Те саме, що й функція <code>load</code> , але шукається файл разом з <code>sys.path</code> , якщо його не знайдено в поточному каталозі
truetype – <code>ImageFont.truetype(file, size, encoding=value)</code> ⇒ об'єкт шрифту	Завантаження файла шрифту <i>Truetype</i> або <i>Opentype</i> і створення об'єкта зі шрифтом заданого розміру, задання кодування: "UNIC" (Unicode), "Symb" (Microsoft Symbol), "ADOB" (Adobe Standard), "ADBE" (Adobe Expert), і "armn" (Apple Roman)

Методи модуля **The Imagefont Module** наведено в табл. 2.10

Таблиця 2.10

Ім'я та синтаксис методу	Застосування методу
getsize – <code>font.getsize(text) ⇒ (width, height)</code>	Повертання кортежу з розміром шрифту
getmask – <code>font.getmask(text, mode=" ") ⇒ зображення</code>	Повертання зображення для тексту

2.2. Особливості використання функцій і методів модулів пакета Pillow

Розглянемо детальніше властивості і синтаксис основних функцій і методів пакета модулів бібліотеки **Pillow**, а також наведемо конкретні приклади кодів, що дають змогу практично реалізувати програмні можливості цієї бібліотеки.

2.2.1. Функції для відкриття, копіювання і зберігання файлів зображень

Функція `open()` відкриває файл з готовим зображенням і повертає об'єкт, за допомогою якого ведеться подальша робота із зображенням. Якщо відкрити файл із зображенням не вдалося, то у вікні інтерпретатора з'являється виключення `Ioerror`. Синтаксис функції `open()` є таким:

```
open(<Шлях або файловий об'єкт>[,mode='r'])
```

У першому параметрі вказується абсолютний або відносний шлях до зображення. Необов'язковий другий параметр задає режим доступу до файла. Якщо його не вказано, то файл буде доступним лише для читання. Відкриємо файл `image_1.jpg`, розташований у поточному робочому каталозі (рис. 2.1):

```
>>> img=Image.open("C:/Users/Леонід/Desktop/image_1.jpg")
```

Замість цього можна передати файловий об'єкт, відкритий у бінарному режимі, наприклад:

```
# Відкриваємо файл у бінарному режимі
>>> f = open("C:/Users/Леонід/Desktop/12.jpg", "rb")
>>> img = Image.open(f) # Передаємо об'єкт файла
>>> img.size             # Отримуємо розмір зображення
(304, 190)
>>> img.format           # Виводимо формат зображення
'JPG'
>>> f.close()            # Закриваємо файл
```

З прикладу видно, що формат зображення визначається автоматично. Після відкриття файлу за допомогою функції `open()` саме зображення не завантажується відразу в пам'ять, завантаження відбувається при першій операції із зображенням.

У цьому прикладі для перегляду зображення використовується метод `show()`. Їм користуватимемося й надалі.

Метод `show()` створює тимчасовий файл у форматі BMP і запускає програму для перегляду зображень, яка використовується в операційній системі за умовчанням (наприклад, це може бути засіб перегляду фотографій у Windows 7).



Рис. 2.1. Приклад відкриття файлу з готовим зображенням

Метод `load` дає змогу завантажити зображення явним чином, повертає об'єкт, за допомогою якого можна отримати доступ до окремих пікселів зображення. Указавши всередині квадратних дужок два значення – горизонтальну й вертикальну координати пікселя, можна отримати або задати його колір, наприклад:

```
>>> img=Image.open("foto.jpg")
>>> obj = img.load()
# Отримуємо колір пікселя
>>> obj = [25,45]
(122, 86, 62)
)# Задаємо колір пікселя (червоний)
>>> obj = [25,45] = (255, 0, 0)
```

Для доступу до окремого пікселя замість методу `load()` можна використовувати методи `getpixel` і `putpixel()`. Метод `getpixel(<Координати>)` дає змогу отримати колір указанного пікселя, а метод `putpixel(<Координати>, <Колір>)` змінює колір пікселя. Координати пікселя вказуються у вигляді кортежу з двох елементів. Зазначимо, що ці методи працюють повільніше, ніж метод `load()`. Наведемо приклад використання методів `getpixel` і `putpixel()`:

```
>>> img=Image.open("foto.jpg")
# Отримуємо колір пікселя
>>> img.getpixel((25,45))
(122, 86, 62)
# Змінюємо колір пікселя
>>> img.putpixel((25,45), (255, 0, 0))
# Отримуємо колір пікселя
(255, 0, 0)
# Переглядаємо зображення
>>> img.show()
```

Метод `copy()` створює копію зображення (рис. 2.2):

```
>>> from PIL import Image
# Відкриваємо файл
>>> img=Image.open("C:/Users/Леонід/Desktop/image_2.jpg")
>>> img.show() # Переглядаємо оригінал
>>> img1 = img.copy() # Створюємо копію
>>> img1.show() # Переглядаємо копію
```



а



б

Рис. 2.2. Створення копії зображення: а – оригінал; б – копія

Метод `thumbnail(<Розмір>[, <Фільтр>])` створює зменшену версію зображення вказаного розміру – мініатюру. Розмір задається у вигляді кортежу з двох елементів: (`<Ширина>`, `<Висота>`). Розмір змінюється пропорційно, тобто за основу береться мінімальне значення, а друге значення обчислюється пропорційно першому. У параметрі `<Фільтр>` може

бути вказаний фільтр NEAREST, BILINEAR, BICUBIC або LANCZOS. Якщо параметр не вказаний, то використовується значення BICUBIC. Метод змінює зображення і нічого не повертає. Наведемо приклад (рис. 2.3):

```
>>> from PIL import Image
>>> img=Image.open("C:/Users/Леонід/Desktop/image_3.jpg")
>>> img.size # Вихідні розміри зображення
(297, 170)
>>> img.show()
>>>img1 = img.thumbnail((148,85),Image.LANCZOS)
>>> img.size # Змінюється саме зображення
(148, 85)
>>> img.show()
```



а



б

Рис. 2.3. Створення зменшеної мініатюри зображення: а – оригінал; б – мініатюра

Метод **save()** призначений для зберігання зображення у файл. Формат і синтаксис цього методу є такими:

```
save (<Шлях або файловий об'єкт> [,<формат> [,<Опції>] ])
```

У першому параметрі вказується абсолютний або відносний шлях. Замість шляху можна передати файловий об'єкт, відкритий у бінарному режимі. Для прикладу збережемо зображення у форматах JPEG і BMP різними способами:

```
>>> from PIL import Image
>>> img = Image.open("C:/Users/Леонід/Desktop/12.jpg")
>>> img.save("tmp.jpg") # Зберігання у форматі JPEG
>>> img.save("tmp.bmp","BMP") # Зберігання у форматі BMP
>>> f = open("tmp1.bmp","wb")
>>> img.save(f,"BMP") # Передаємо файловий об'єкт
>>> f.close()
```

Зазначимо, що файл може бути відкритий в одному форматі, а збережений в іншому. Якщо параметр `<формат>` не задано, то формат зображення визначається за розширенням файла, проте, якщо методу `save()` як перший параметр передається файловий об'єкт, то формат має бути вказаний обов'язково.

У параметрі `<Опції>` можна передавати додаткові опції. Підтримувані опції залежать від формату зображення. Наприклад, за умовчанням зображення JPEG зберігаються з якістю 75. За допомогою опції `quality` можна вказати інше значення в діапазоні від 1 до 100:

```
# Указання якості
>>> img.save("tmp2.jpg", "JPEG", quality=100)
```

2.2.2. Створення нового зображення

Бібліотека ***Pillow*** дає змогу працювати не лише з готовими зображеннями, а й створювати нові. Розглянемо таку можливість детальніше.

Функція `new()` для створення нового зображення має такий формат:

```
new(<Режим>, <Розмір> [, (<Колір фону>)])
```

У параметрі `<Режим>` указується один із таких режимів:

- 1 – 1 біт, чорно-біле;
- L – 8 бітів, чорно-біле;
- P – 8 бітів, кольорове (256 кольорів);
- RGB – 24 біти, кольорове;
- RGBA – 32 біти, кольорове з альфа-каналом;
- CMYK – 32 біти, кольорове;
- YCbCr – 24 біти, кольорове, відеоформат;
- Lab – 24 біти, кольорове, використовується колірний простір Lab;
- HSV – 24 біти, кольорове, використовується колірний простір HSV;
- I – 32 біти, кольорове, кольори кодуються цілими числами;
- F – 32 біти, кольорове, кольори кодуються дійсними числами.

У другому параметрі необхідно передати розмір створюваного зображення (полотна) у вигляді кортежу з двох елементів `<Ширина>`, `<Висота>`. У необов'язковому параметрі `<Колір фону>` задається колір фону. Якщо параметр не вказано, то фон буде чорного кольору. Для режиму RGB колір указується у вигляді кортежу з трьох цифр від 0 до 255 (`<Частка червоного>`, `<Частка зеленого>`, `<Частка синього>`). Крім того, можна вказати назву кольору англійською мовою і у вигляді рядка у форматах `"#rgb"` і `"#rrggbb"`. Різні способи вказування кольору наведено далі:

```
>>> img = Image.new("RGB", (100,100))
>>> img.show() # Чорний квадрат
>>> img = Image.new("RGB", (100,100), (255,0,0))
>>> img.show() # Червоний квадрат
>>> img = Image.new("RGB", (100,100), ("green"))
>>> img.show() # Зелений квадрат
>>> img = Image.new("RGB", (100,100), ("#f00"))
>>> img.show() # Червоний квадрат
>>> img = Image.new("RGB", (100,100), ("#ff0000"))
>>> img.show() # Червоний квадрат
```

2.2.3. Отримання інформації про зображення

Отримати інформацію про зображення дають змогу такі атрибути об'єкта зображення:

- `size` – розмір зображення у вигляді кортежу з двох елементів: <Ширина>, <Висота>;
- `format` – формат зображення у вигляді рядка (наприклад, 'gif', 'jpeg' і т. д.);
- `mode` – режим у вигляді рядка (наприклад, 'p ', 'rgb' і т. д.);
- `info` – додаткова інформація про зображення у вигляді словника.

Наведемо приклад виведення інформації про зображення у форматах JPEG, GIF, BMP, TIFF і PNG:

```
>>> img=Image.open("foto.jpg")
>>> img.size, img.format, img.mode
((800, 600), 'JPEG', 'RGB')
>>> img.info
{'jfif':258, 'jfif_unit':0, 'adobe':100, 'progression':1,
'jfif_version':(1,2), 'adobe_transform':100,
'jfif_density'(100, 100)}
>>> img=Image.open("foto.gif")
>>> img.size, img.format, img.mode
((800, 600), 'GIF', 'P')
>>> img.info
{'version': 'GIF89a, 'background':254}
```

2.2.4. Перетворення зображень

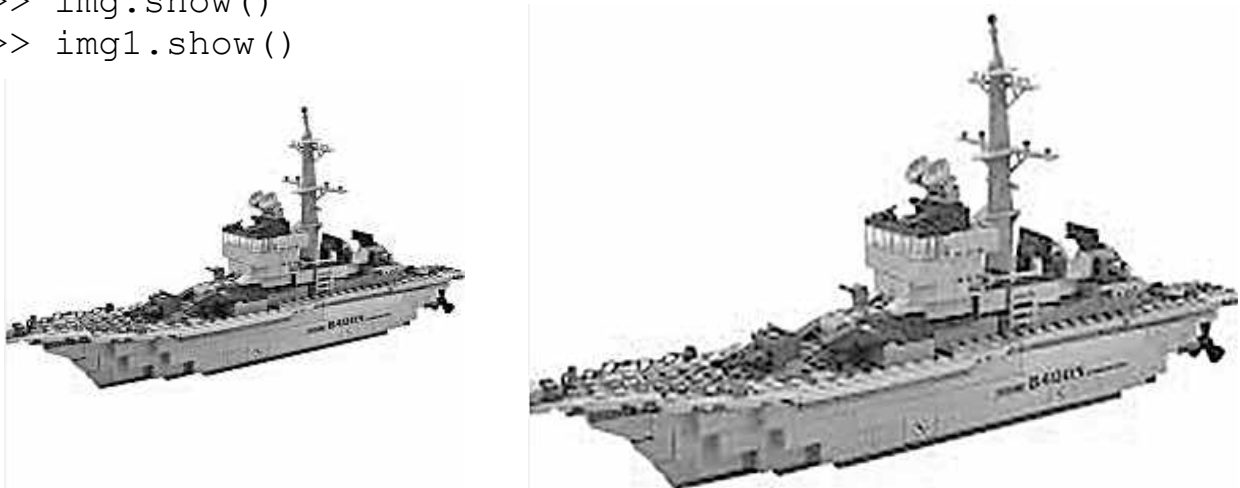
Наведемо приклади використання основних методів перетворення зображень.

Метод `resize(<Розмір>[, <Фільтр>])` змінює розмір зображення. На відміну від методу `Thumbnail()` повертає нове зображення, а не змінює початкового. Розмір змінюється непропорційно, і якщо пропорції не дотримуються, то зображення спотворюється. У параметрі <Фільтр> мо-

жуть бути вказані фільтри NEAREST, BILINEAR, BICUBIC, LANCZOS. Якщо параметр не вказано, то використовується значення NEAREST.

Наведемо приклад (рис. 2.4):

```
>>> from PIL import Image
>>> img=Image.open("C:/Users/Леонід/Desktop/image_5.jpg")
>>> img.size          # Вихідні розміри зображення
(225, 225)
>>> img1=img.resize((450,450),Image.LANCZOS)
>>> img1.size         # Пропорційне збільшення
(450, 450)
>>> img.show()
>>> img1.show()
```



а

б

Рис. 2.4. Змінення розміру зображення: а – оригінал; б – копія

Метод `rotate(<Кут>[,<Фільтр>][,<expand=0>])` повертає зображення на вказаний кут, що вимірюється в градусах проти годинникової стрілки. У параметрі `[<Фільтр>]` можуть бути вказані фільтри NEAREST, BILINEAR, BICUBIC. Якщо параметр не вказано, то використовується значення NEAREST. Якщо параметр `expand` має значення True, то розмір зображення буде збільшений так, щоб воно повністю помістилося: за умовчанням розмір зображення зберігається, а якщо зображення не поміщається, то його буде обрізано.

Наведемо приклад (рис. 2.5):

```
>>> from PIL import Image
>>> img=Image.open("C:/Users/Леонід/Desktop/image_6.jpg")
>>> img.size
(275, 183)
>>> img.show()
>>> img1=img.rotate(45,Image.NEAREST,expand=True)
>>> img1.size
(325, 325)
>>> img1.show()
```



а



б

Рис. 2.5. Повертання зображення: а – оригінал; б – копія

Метод **Transpose** (<Перетворення >) повертає дзеркальний образ або зображення. Як параметр вказують значення `Flip_left_right`, `Flip_top_bottom`, `Rotate_90`, `Rotate_180`, `Rotate_270`, `TRANSPOSE`. Метод повертає нове зображення, наприклад (рис. 2.6):

```
>>> from PIL import Image
>>> img=Image.open("C:/Users/Леонід/Desktop/image_2.jpg")
>>> img.show()      # Початкове зображення
>>> img1 = img.transpose(Image.FLIP_LEFT_RIGHT)
>>> img1.show()      # Горизонтальний дзеркальний образ
>>> img2 = img.transpose(Image.FLIP_TOP_BOTTOM)
>>> img2.show()      # Вертикальний дзеркальний образ
>>> img3 = img.transpose(Image.ROTATE_90)
>>> img3.show()      # Поворот на 90 градусів проти годин. стр.
>>> img4 = img.transpose(Image.ROTATE_180)
>>> img4.show()      # Поворот на 180 градусів
```



а



б



в



г

Рис. 2.6. Дзеркальні відображення (а – в) і повертання зображень (д)



д

Рис. 2.6. Закінчення

Метод `crop(<x1>,<y1>,<x2>,<y2>)` вирізає прямокутний фрагмент із початкового зображення. Як параметр вказується кортеж із чотирьох елементів: перші два елементи задають координату лівого верхнього кута вирізуваного фрагмента, а другі два елементи – координати його нижнього правого кута. Передбачається, що початок координат розташовується у верхньому лівому куті зображення. Додатна вісь x напрямлена вправо, а додатна вісь y – униз. Як значення метод повертає нове зображення. Прочитування області з початкового зображення виконується лише при першій операції над новим зображенням. Якщо після виконання методу `crop()` над початковим зображенням були проведені операції, то вони відображатимуться й на новому зображенні. Щоб явним чином здійснити прочитування області, необхідно відразу після методу `crop()` викликати метод `load()`.

Наведемо приклад (рис. 2.7):

```
>>> from PIL import Image
>>> img=Image.open("C:/Users/Леонід/Desktop/image_4.jpg")
# Виділяємо фрагмент
>>> img1 = img.crop( [0, 0, 100, 100])
# Прочитуємо фрагмент, створюючи нове зображення
>>> img1.load()
# Визначаємо розміри нового зображення
>>> img1.size
(100,100)
>>> img1.show()
```



а



б

Рис. 2.7. Виділення прямокутного фрагмента зображення: а – початкове зображення; б – частина зображення

Метод ***paste(<Колір>, <Область>[, <Маска>])*** зафарбовує прямокутну область певним кольором. Координати області вказуються у вигляді кортежу з чотирьох елементів: перші два елементи задають координату лівого верхнього кута зафарбованої області, а другі два елементи – координати її правого нижнього кута. Зафарбуємо цю область червоним кольором (рис. 2.8):

```
>>> from PIL import Image
>>> img=Image.open("C:/Users/Леонід/Desktop/image_4.jpg")
>>> img.paste((255, 0, 0), (0, 0, 100, 100))
>>> img.show()
```



Рис. 2.8. Зафарбовування прямокутної області певним кольором

Тепер зафарбуємо все зображення зеленим кольором (рис. 2.9):

```
>>> from PIL import Image
>>> img=Image.open("C:/Users/Леонід/Desktop/image_4.jpg")
>>> img.paste((0, 128, 0), img.getbbox())
>>> img.getbbox()
(0, 0, 275, 183)
>>> img.show()
```



Рис. 2.9. Зафарбовування всього зображення певним кольором

У цьому прикладі використано метод `getbbox()`, який повертає координати прямокутної області, у яку вписується все зображення.

Метод `paste(<Зображення>, <Область>[, <Маска>])` вставляє вказане зображення в прямокутну область. Координати області вказуються у вигляді кортежу з двох або чотирьох елементів. Якщо вказаний кортеж складається з двох елементів, то він задає початкову точку цієї області. Для прикладу завантажимо зображення, створимо його зменшену копію, а потім вставимо її в початкове зображення. Навколо вставленого зображення відображатимемо рамку червоного кольору (рис. 2.10):

```
>>> from PIL import Image
>>> img=Image.open("C:/Users/Леонід/Desktop/image_4.jpg")
>>> img.size
>>> img1=img.resize((100,75))          # Створюємо мініатюру
>>> img1.size
(200, 150)
>>> img.show()
>>> img.paste((255, 0, 0), (9, 9, 111, 86))      # Рамка
>>> img.paste(img1, (10,10))          # Вставляємо мініатюру
>>> img.show()
```



а



б

Рис. 2.10. Вставляння вказаного зображення в прямокутну область:
а – початкове зображення; б – вставлена мініатюра

Необов'язковий параметр <Маска> дає змогу задати міру прозорості зображення, що вставляється, або колір. Для прикладу виведемо напівпрозору білу горизонтальну смугу заввишки 100 пікселів (рис. 2.11):

```
>>> from PIL import Image
>>> img=Image.open("C:/Users/Леонід/Desktop/image_6.jpg")
>>> img.show()
>>> white=Image.new("RGB", (img.size[0],100), (255,255,255))
>>> mask=Image.new("L", (img.size[0],100), 64) # Маска
>>> img.paste(white, (0,0), mask)
>>> img.show()
```



а



б

Рис. 2.11. Накладання напівпрозорої білої горизонтальної смуги:
а – початкове зображення; б – на зображенні біла горизонтальна смуга

Метод *split()* повертає канали зображення у вигляді кортежу. Наприклад, для зображення в режимі RGB повертається кортеж із трьох елементів: (R,G,B). Виконати зворотну операцію (відновити зображення з каналів) дає змогу метод *merge* (<Режим>, <Канали>). Наведемо приклад перетворення режиму RGB на режим RGBA (рис. 2.12):

```
>>> from PIL import Image
>>> img=Image.open("C:/Users/Леонід/Desktop/image_8.jpg")
>>> img.mode
'RGB'
>>> R,G,B = img.split()
>>> mask = Image.new("L",img.size, 128)
>>> img1 = Image.merge("RGBA", (R, G, B, mask))
>>> img1.mode
'RGBA'
>>> img1.show()
```



Рис. 2.12. Перетворення режиму RGB на режим RGBA

Метод **Convert** – `convert (<Новий режим> [, <Матриця> [, <Режим змішування кольорів> [, <Палітра> [, <Кількість кольорів>]]]]`. Цей метод перетворює зображення у вказаний режим і повертає нове зображення. Третій параметр указує спосіб отримання складних кольорів з простіших шляхом змішування і має сенс при перетворенні зображень у форматі RGB або L на зображення у формат P або 1 – доступні значення None (змішування не виконується) і `Image.FLOYDSTEINBERG` (значення за умовчанням). Четвертий параметр задає тип палітри при перетворенні RGB на P: `Image.WEB` (Web-сумісна палітра – значення за умовчанням) або `Image.ADAPTIVE` (адаптивна палітра). П'ятий параметр задає кількість кольорів у палітрі, за умовчанням – 256. Перетворимо зображення в режимі RGB на зображення в режимі RGBA (рис. 2.13):

```
>>> from PIL import Image
>>> img=Image.open("C:/Users/Леонід/Desktop/image_12.jpg")
>>> img.mode
'RGB'
>>> img1 = img.convert("RGBA")
>>> img1.mode
'RGBA'
>>> img1.show()
```



Рис. 2.13. Перетворення зображення

Перетворимо зображення у форматі RGB на зображення у форматі P, указавши змішування кольорів і адаптивну палітру з 128 кольорів:

```
>>> from PIL import Image
>>> img=Image.open("C:/Users/Леонід/Desktop/image_12.jpg")
>>> img.mode
'RGB'
>>> img1=img.convert("P", None, Image.FLOYDSTEINBERG,
Image.ADAPTIVE, 128)
>>> img1.mode
'P'
>>> img1.show()
```

Метод ***Filter(<Фільтр>)*** застосовує до зображення вказаний фільтр і повертає нове зображення. Як приклад можна навести фільтри **BLUR, CONTOUR, DETAIL, Edge_enhance, Edge_enhance_more, emboss, Find_edges, SHARPEN, SMOOTH**, і **Smooth_more** з модуля **Imagefilter**. Пропонований набір фільтрів виконує різного роду операції (рис. 2.14):

```
>>> from PIL import Image
>>> from PIL import ImageFilter
>>> img=Image.open("C:/Users/Леонід/Desktop/image_2.jpg")
>>> img.show() # Початкове зображення
>>> img1=img.filter(ImageFilter.BLUR)
>>> img1.show() # Відфільтроване зображення
>>> img2=img.filter(ImageFilter.CONTOUR)
>>> img2.show()
>>> img3=img.filter(ImageFilter.DETAIL)
>>> img3.show()
>>> img4=img.filter(ImageFilter.EDGE_ENHANCE)
>>> img4.show()
>>> img5=img.filter(ImageFilter.EDGE_ENHANCE_MORE)
>>> img5.show()
>>> img6=img.filter(ImageFilter.EMBOSS)
>>> img6.show()
>>> img7=img.filter(ImageFilter.FIND_EDGES)
>>> img7.show()
>>> img8=img.filter(ImageFilter.SHARPEN)
>>> img8.show()
>>> img9=img.filter(ImageFilter.SMOOTH)
>>> img9.show()
>>> img10=img.filter(ImageFilter.SMOOTH_MORE)
>>> img10.show()
```

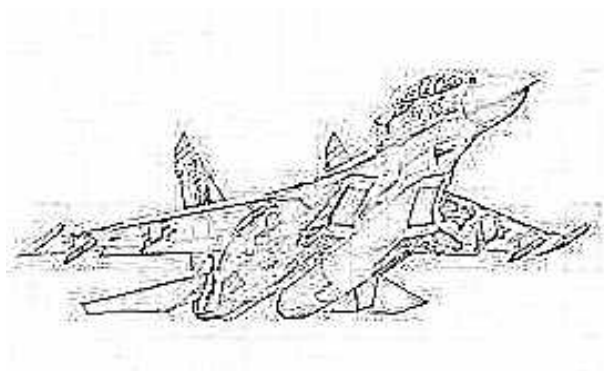


а



б

Рис. 2.14. Варіанти фільтрації зображень:
а – початкове зображення; **б** – фільтр **BLUR**; **в** – фільтр **CONTOUR**; **г** – фільтр **DETAIL**;
д – фільтр **EDGE_ENHANCE**; **е** – фільтр **EDGE_ENHANCE_MORE**; **ж** – фільтр **EMBOSS**;
з – фільтр **FIND_EDGES**; **и** – фільтр **SHARPEN**; **л** – фільтр **SMOOTH**;
м – фільтр **SMOOTH_MORE**



в



г



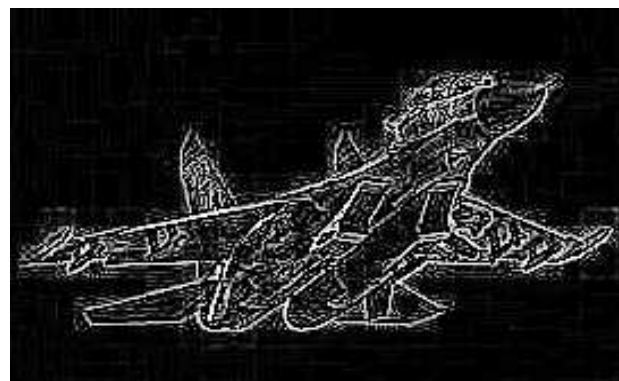
д



е



ж



и



к



л

Рис. 2.14. Продовження



М

Рис. 2.14. Закінчення

Наведені приклади не охоплюють усі види перетворень зображень, які можна здійснити за допомогою бібліотеки **Pillow**, проте формують загальні уявлення про методи й засоби оброблення графічної інформації на мові **Python**. Подальше використання цих методів можна виконати самостійно.

3. АЛГОРИТМИ OPENCV ДЛЯ ОБРОБЛЕННЯ ЗОБРАЖЕНЬ У ПРОГРАМІ PYTHON

Розглянемо далі основні ресурси, які надає бібліотека **OpenCV** для оброблення зображень при спільній роботі з програмою **Python**. Для цього наведемо кілька прикладів, що ілюструють можливості бібліотеки, а також особливості синтаксису її функцій при написанні програмних кодів для перетворення зображень.

3.1. Уведення й візуалізація зображень і відеоданих

Уведення й візуалізація зображень. Для початку роботи із зображенням його необхідно ввести в програму й відобразити для візуального перегляду. Для цього слід написати короткий код:

```
>>> import cv2 as cv
>>> img = cv.imread('1.jpg')
>>> cv.imshow("RGB",img)
```

Зверніть увагу, що в першому рядку коду здійснюється імпорт ресурсів бібліотеки **OpenCV**. Ця операція буде успішно виконана, якщо перед початком роботи на комп'ютері зроблено підключення бібліотеки, методику якого детально описано в розд. 1 цієї роботи.

Крім того, потрібне зображення необхідно помістити в системну теку **Python**. Це спрощує процедуру введення (не потрібно прописувати шлях до файла зображення) і запобігає виникненню синтаксичних помилок. Слід нагадати, що обов'язково має бути вказаний формат зображення, у цьому прикладі – jpg (рис. 3.1).



Рис. 3.1. Приклад уведення зображення та його вигляд

Підключення відеокамери. У практиці використання систем технічного зору застосовуються два основні режими роботи з відеоданими – уведення й оброблення відео в реальному масштабі часу безпосередньо від відеореєстратора й аналіз відеоданих з раніше записаного файла. Перший режим зазвичай використовують для вирішення завдання керування рухомими об'єктами на основі аналізу даних відеоспостереження сцени, а другий – для перегляду подій, зареєстрованих, наприклад, у системах охорони об'єктів, або для аналізу наукової відеоінформації.

Наведемо приклад програмного коду на **Python** з використанням функції **OpenCV** `cv.VideoCapture ( )` для підключення камери:

```
import cv2 as cv                # Підключення бібліотеки OpenCV
cap = cv.VideoCapture(0)        # Створення пристрою захоплення
while True:                    # Цикл для перебору кадрів
    ret, img = cap.read()       # Читання поточного кадру
    cv.imshow("frame",img)     # Відображення поточного кадру
    if cv.waitKey(1) & 0xFF == ord('q'): # Читаємо поточний кадр, якщо файл ураховано як ret = True, інакше False
        break
cap.release()                  # Закриття пристрою захоплення
cv.destroyAllWindows()        # Закриття всіх створених вікон
```

Зазначимо, що для використання однієї з декількох камер у функції введення необхідно вказати цілочисловий аргумент. У випадку, коли є лише одна камера, аргумент слід вибирати таким, що дорівнює нулю, – `cv.VideoCapture (0)`. На рис. 3.2 показано приклад підключення web-камери при використанні Notebook.



Рис. 3.2. Підключення web-камери

Читання відеоданих з файла. Опишемо процедуру введення відеоданих з готового файлу. Як і у випадках введення зображень, потрібний файл відео необхідно помістити в системну теку **Python**. Це спрощує процедуру введення (не потрібно прописувати шлях до файлу) і запобігає виникненню синтаксичних помилок. Нагадаємо, що обов'язково має бути вказаний формат відео, у цьому прикладі – `avi`.

Нижче наведено код уведення відеоданих з файлу за допомогою функції `cap = cv.VideoCapture ('filename')`, а на рис. 3.3 зображено один із фреймів уведеної відеопослідовності. Зазначимо, що в програмі введення відеоданих передбачено процедуру переривання внаслідок натиснення клавіші 'q'.

```

# Підключення бібліотеки OpenCV
import cv2 as cv
# Створення пристрою захоплення
cap = cv.VideoCapture('V_109.avi')
# Цикл для перебирання кадрів
while(cap.isOpened()):
# Читання поточного кадру, якщо файл ураховано як ret =
    True, інакше False
    ret, frame = cap.read()
    if(ret==False):      # Якщо ret = False - перервати цикл
        break
    cv.imshow('frame',frame) #Відображення поточного кадру
    if cv.waitKey(1)& 0xFF == ord('q'): #Оброблення натис-
        нення клавіші 'q', при натисненні - перервати цикл і
        завершити програму
        break
cap.release()           #Закриття пристрою захоплення
cv.destroyAllWindows()  #Закриття всіх створених вікон

```



Рис. 3.3. Читання відео з файла

3.2. Перетворення зображень за допомогою функцій OpenCV

Перетворення RGB-зображення на напівтонове. У бібліотеці **OpenCV** міститься безліч функцій перетворення типів зображень. Розглянемо найбільш поширені перетворення – за допомогою функції `cv.COLOR_RGB2GRAY` початкове кольорове зображення перетворюється на зображення у відтінках сірого (рис. 3.4):

```
import cv2 as cv
img = cv.imread('16.jpg')
cv.imshow("RGB",img)
gray = cv.cvtColor(img, cv.COLOR_RGB2GRAY)
cv.imshow("Gray",gray)
```

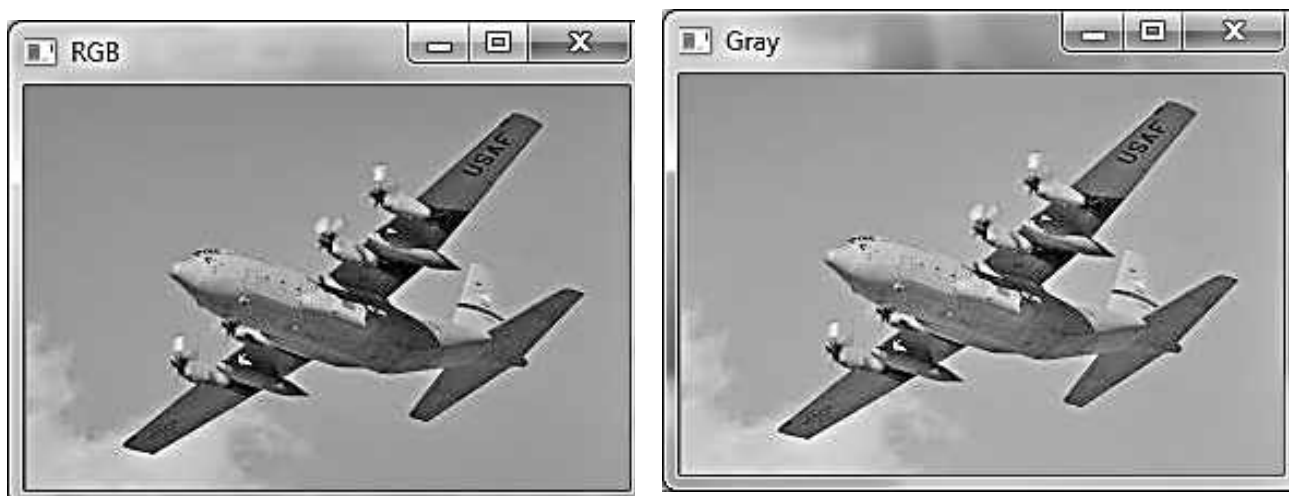


Рис. 3.4. Перетворення RGB-зображення на напівтонове

Перетворення колірних просторів (Converting between color spaces). У системах технічного зору при використанні повнокольорових зображень зазвичай орієнтуються на використання колірного простору RGB. Проте, окрім RGB-моделі подання зображень існує ряд моделей, які більше відповідають технічним цілям і є краще пристосованими до людського сприйняття.

Розглянемо основні особливості колірних просторів і наведемо приклади перетворень з одного простору на інший, використовуючи відповідні функції перетворення бібліотеки **OpenCV**.

Повнокольорове **RGB**-зображення (Red, Green, Blue) – це масив $M \times N \times 3$, що складається з трьох матриць розміром $M \times N$. Матриці відповідають трьом колірним компонентам: червоний, зелений і синій. Три монохромні зображення, що формують єдине RGB-зображення, називають червоною, зеленою й синьою компонентами зображення. Їх подають у форматі `uint8`, `uint16`, `single` або `double`. Використовуються такі числові дані для подання колірних компонент:

- `[0, 1]` – для форматів `double` і `single`;
- `[0, 255]` – для формату `uint8`;
- `[0, 65535]` – для формату `uint16`.

Кількість бітів, що використовуються для подання величини кольорового пікселя за всіма складовими `RGB`, називають глибиною кольору зображення. Наприклад, якщо кожна компонента є 8-бітовим зображенням, то відповідне `RGB`-зображення має глибину 24 біти. У моделі `RGB` не розділені яскравісна й відтінкова компоненти кольору, тут легко вказати яскравість одного з основних кольорів, але важко вказати відтінок з необхідним колірним тоном (наприклад, тілесним) і насиченістю.

Колірна модель **HSV** (*Hue, Saturation, Value* – колірний тон, насиченість, міра яскравості) – модель, орієнтована на людину і можливість явного задання необхідного відтінку кольору. Серед інших моделей, що використовуються, ця модель відображає фізичні властивості кольору і найбільш точно відповідає способу сприйняття кольору людським оком.

У колірній моделі **YUV** колір подається як три компоненти: одна – яскравість (**Y**); дві інші – кольорорізницеви (**U** і **V**). Модель широко застосовується в телебаченні і для зберігання/оброблення відеоданих. Яскравісна компонента містить чорно-біле (у відтінках сірого) зображення, а інші дві компоненти містять інформацію для відновлення необхідного кольору. Це було зручно в момент появи кольорового ТБ для сумісності з чорно-білими телевізорами.

У колірному просторі **YUV** є одна компонента, яка являє собою яскравість (сигнал яскравості), а дві інші – колір (сигнал колірності). У той час як яскравість передається з усіма деталями, у компонентах кольорорізницевого сигналу, що не містить інформації про яскравість, деякі деталі можуть бути видалені шляхом пониження розділення відліків (фільтрації або усереднення).

Далі наведемо приклад перетворення зображення в форматі **RGB** на зображення у форматі **YUV** (рис. 3.5):

```
import cv2

img = cv2.imread('./images/input.jpg')

yuv_img = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)

cv2.imshow('Input', img)
cv2.imshow('YUV image', yuv_img)
cv2.imshow('Y channel', yuv_img[:, :, 0])
cv2.imshow('U channel', yuv_img[:, :, 1])
cv2.imshow('V channel', yuv_img[:, :, 2])

cv2.waitKey()
```

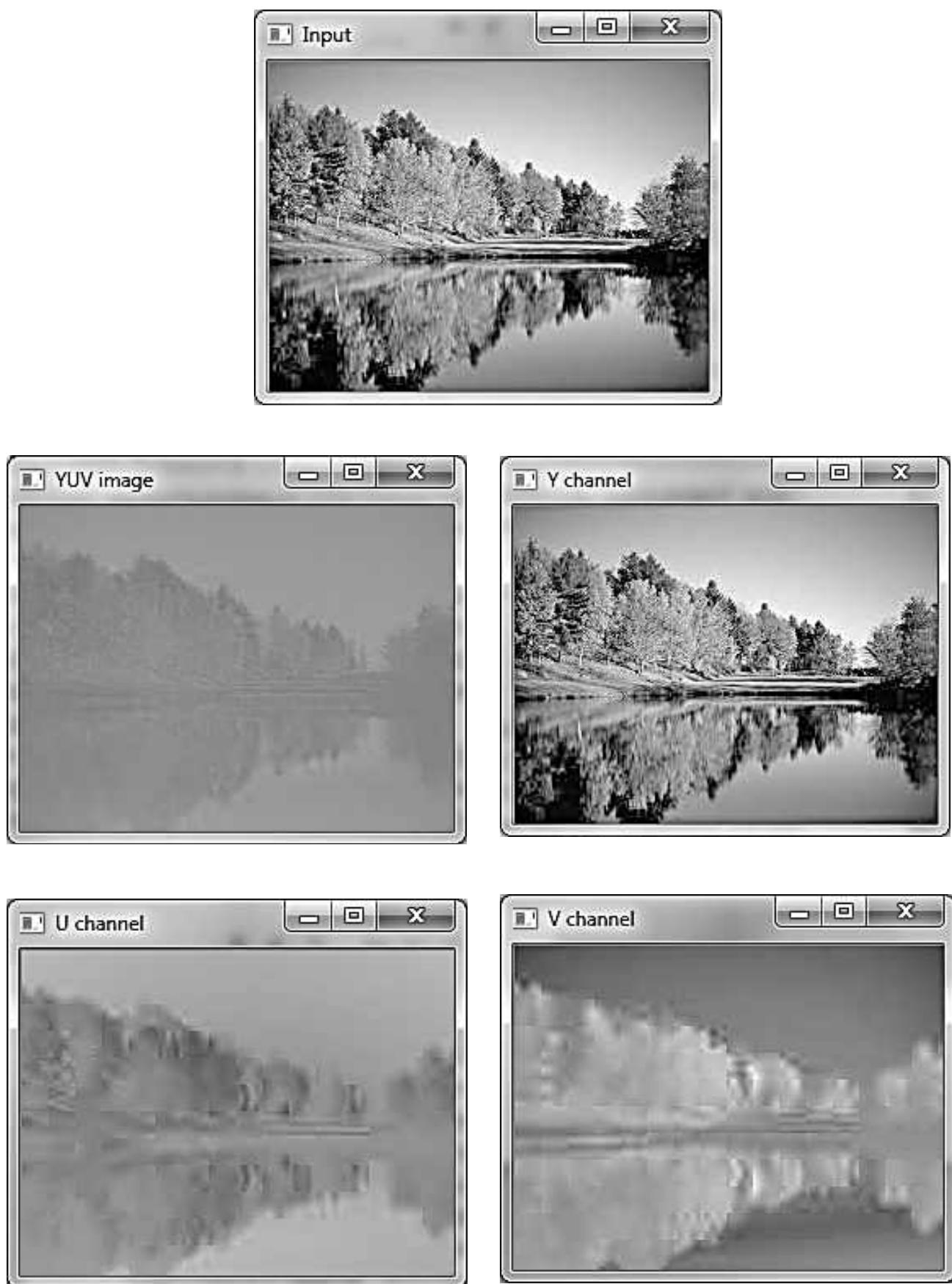


Рис. 3.5. Перетворення RGB-зображення на YUV-зображення

Наведемо ще один приклад перетворення зображення в форматі RGB на зображення в форматі HSV з використанням функції `cv2.cvtColor` (`img, cv2.COLOR _ BGR2HSV`) (рис. 3.6):

```
import cv2

img = cv2.imread('./images/input.jpg')
hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

cv2.imshow('Input', img)
cv2.imshow('HSV image', hsv_img)
cv2.imshow('H channel', hsv_img[:, :, 0])
cv2.imshow('S channel', hsv_img[:, :, 1])
cv2.imshow('V channel', hsv_img[:, :, 2])

cv2.waitKey()
```



Рис. 3.6. Перетворення RGB-зображення на HSV-зображення



Рис. 3.6. Закінчення

Виділення меж на зображенні. Функція `BW=edge (I, method)` призначена для виділення меж на початковому напівтоновому зображенні `I`. Ця функція повертає бінарне зображення `BW` такого ж розміру, як початкове `I`. Піксель `BW(r,c)` дорівнює 1, якщо піксель `I(r,c)` належить межі. Для виявлення меж може використовуватися декілька методів. Метод задається в параметрі `method` у вигляді одного з таких рядків: 'sobel', 'prewitt', 'roberts', 'log', 'zerocross', 'canny'. Ці назви відповідають іменам авторів різних алгоритмів. Найбільш точним та ефективним при визначенні меж є метод Кенні.

Для кожного з методів визначення меж можна задати додаткові параметри. Для цього використовується одна з функцій `BW=edge (I, method, thresh)`, де параметр `thresh` задає поріг для визначення того, чи належить піксель межі. Наведемо приклад програмного коду визначення меж за методом Кенні із застосуванням функції `edges = cv.Canny (img, thresh)`:

```
import cv2 as cv
# Читання RGB-зображення
img = cv.imread('17.jpg')
# Показ RGB-зображення
cv.imshow("RGB",img)
# Перетворення RGB-зображення у відтінки сірого
gray = cv.cvtColor(img, cv.COLOR_RGB2GRAY)
cv.imshow("Gray",gray)
# Виділення меж зображення (Алгоритм Кенні)
edges = cv.Canny(img,0,95)
cv.imshow("Edges",edges)
cv.waitKey()
```

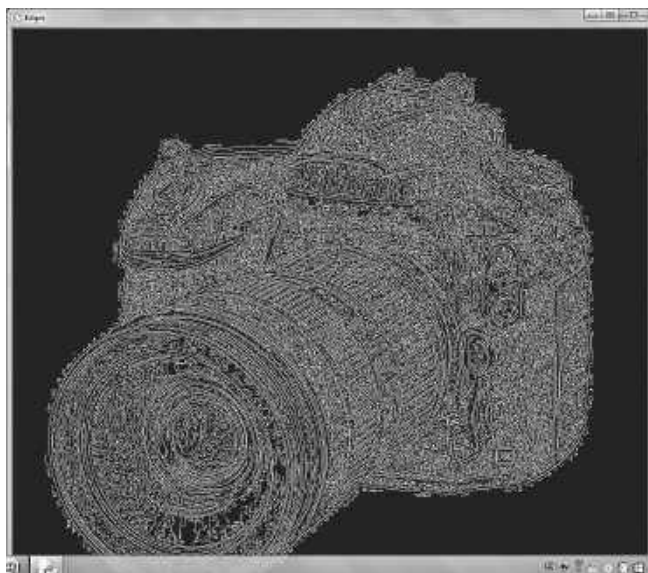
Результати оброблення показано на рис. 3.7.



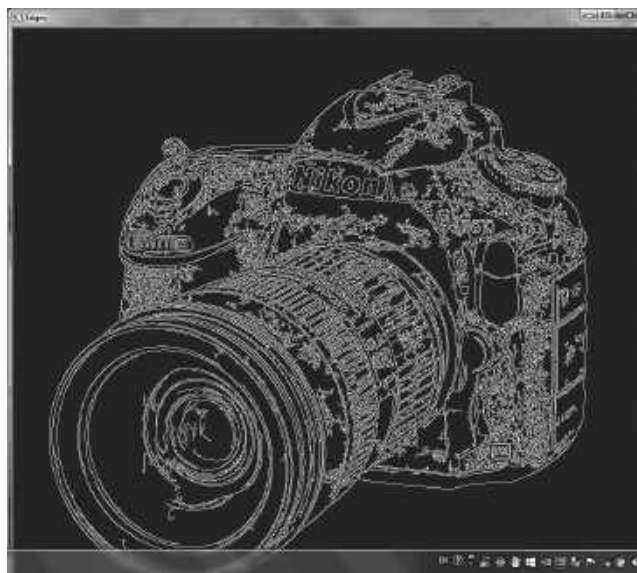
а



б



в



г

Рис. 3.7. Виділення меж

На рис. 3.7, в і г видно, що результати оброблення можуть бути різними – при малих значеннях параметра `thresh` (0.5 ... 0.7) виділяються межі всіх дрібних фрагментів зображення, а при великих значеннях цього параметра (0.9 ... 0.95) виділяються лише глобальні контури. Але ефективність виділення меж значною мірою залежить і від самого зображення – рисунки з крупною текстурою обробляються краще, що добре видно на рис. 3.8.

У цій роботі для зменшення обсягу матеріалу, що викладається, не наведено даних про виділення меж методом Собеля. Читач може розглянути ці алгоритми самостійно.

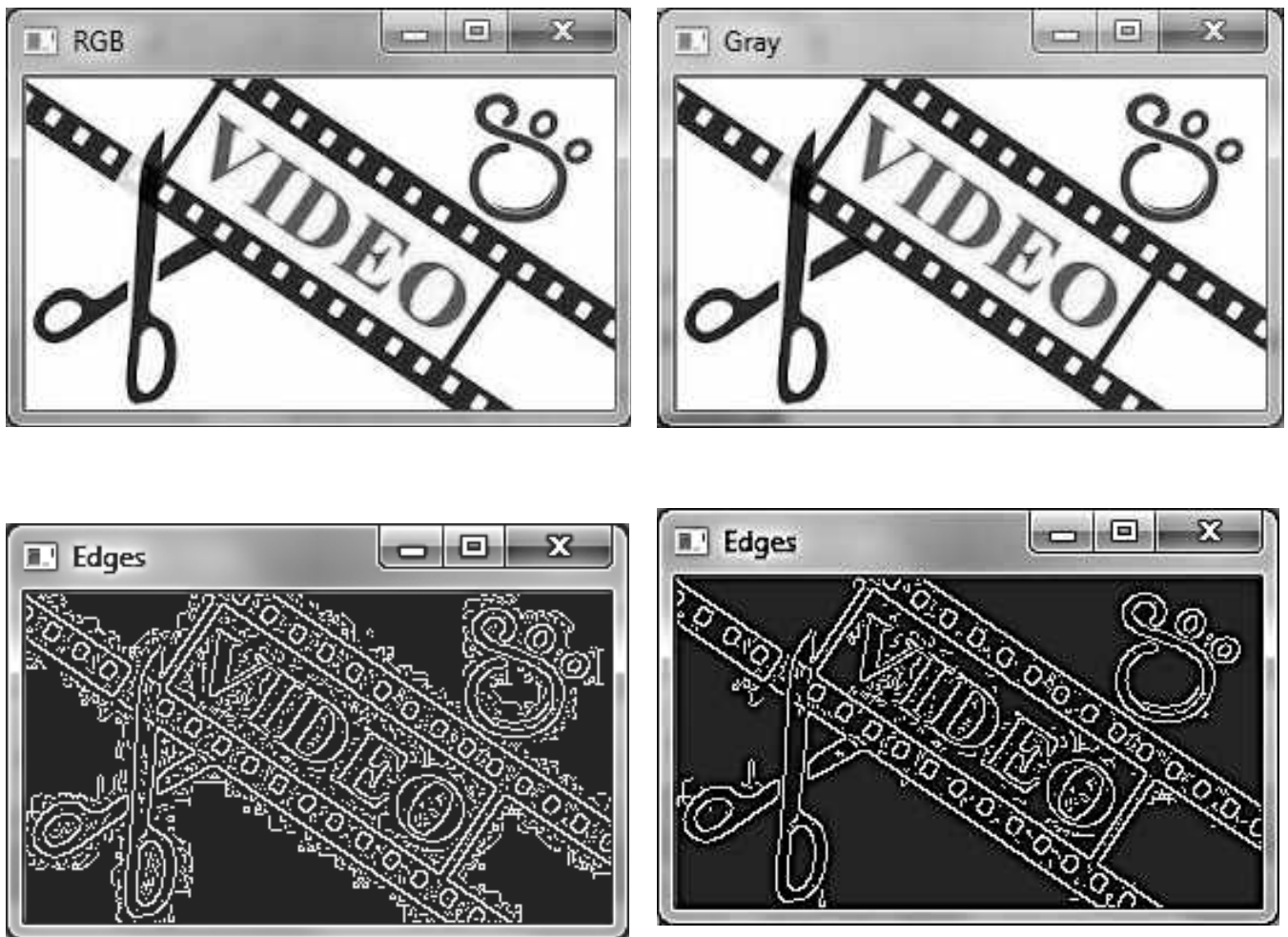


Рис. 3.8. Ефективність виділення меж

Морфологічні перетворення зображень (Erosion and dilation).

Розглянемо основні морфологічні перетворення:

- **Erode** – розмивання (операція звуження);
- **Dilate** – дилатація (операція розширення).

Ерозія (розмивання, звуження) зображення зазвичай використовується для видалення випадкових крапелок на зображенні. Ідея полягає в тому, що крапелки при розмиванні усуваються, тоді як великі й відповідно візуально більш значущі області залишаються звуженими (операція стягнення).

Дилатація (розширення) ж, імовірно, так само має усувати шум і сприяти об'єднанню областей зображення, розділених шумом, тінями та ін. Застосування ж невеликого розширення має об'єднати ці області в одну.

Морфологічні операції найчастіше застосовуються для дій над двійковими зображеннями, які виходять після порогового перетворення (thresholding). Тому як початкове зображення у цьому прикладі використовуємо файл формату **png*. Для реалізації програмного коду викорис-

ТОВУЄМО функції `cv2.erode(img, kernel, iterations=1)` і `cv2.dilate(img, kernel, iterations=1)` (рис. 3.9):

```
import cv2
import numpy as np

img = cv2.imread('30.png', 0)

kernel = np.ones((5,5), np.uint8)

img_erosion = cv2.erode(img, kernel, iterations=1)
img_dilation = cv2.dilate(img, kernel, iterations=1)

cv2.imshow('Input', img)
cv2.imshow('Erosion', img_erosion)
cv2.imshow('Dilation', img_dilation)

cv2.waitKey(0)
```



а



б



в

Рис. 3.9. Приклад ерозії (б) і дилатації (в) зображення (а)

3.3. Алгоритми фільтрації зображень у бібліотеці OpenCV

У практиці цифрового оброблення зображень широко використовується маскова фільтрація. Зазвичай як маска застосовується множина вагових коефіцієнтів, заданих в усіх точках околу, які симетрично оточують поточну точку кадру.

Поширеним виглядом околу, що часто використовується на практиці, є квадрат 3×3 з поточним елементом у центрі. Застосовують різні маски. Одним з евристичних варіантів є рівномірна маска, усі дев'ять вагових коефіцієнтів якої дорівнюють $1/9$. Такий вибір коефіцієнтів відповідає умові збереження середньої яскравості, унаслідок чого вихідний сигнал знаходиться в діапазоні яскравості вхідного сигналу.

Просторова фільтрація виконується як операція двовимірної згортки імпульсної характеристики фільтра h із зображенням $f(x, y)$. Вхідний сигнал фільтра, вихідний сигнал фільтра й імпульсна перехідна характеристика (ІПХ) є двовимірними функціями x_{ij} , y_{ij} і h_{kl} відповідно і дискретна згортка також має двовимірний характер:

$$y_{ij} = \sum_k \sum_l x_{i-k, j-l} h_{kl}.$$

Точка зображення на виході двовимірного лінійного фільтра є сумою (середнім значенням) усіх сусідніх точок фільтрованого зображення, узятих з вагами, що визначаються виглядом ІПХ фільтра.

Таку процедуру ще називають віконною фільтрацією, де вікно фільтра – це його ІПХ. Усі елементи фільтрованого зображення, що попали у вікно, усереднюються, унаслідок чого формується вихідний сигнал фільтра.

Процедура фільтрації для всього зображення – це послідовне виконання фільтрації в кожній точці. Вікно фільтра переміщається по всіх елементах фільтрованого зображення і в кожному своєму положенні формує усереднене значення яскравості за всіма точками, що попали у вікно (рис. 3.10).

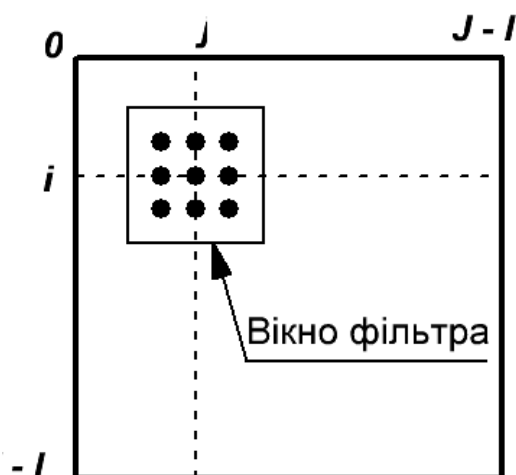


Рис. 3.10. Схема віконної (маскової) фільтрації

Наведемо приклади використання масок для фільтрації зображень різними способами.

Фільтри НЧ. Для зменшення шумів широко застосовуються НЧ-фільтри, оскільки шум є ВЧ-сигналом. Зокрема, для НЧ-фільтрації застосовується усереднювання сигналу в масці, наприклад, при $n=m=3$:

$$H = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad H = \frac{1}{10} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad H = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}.$$

Фільтри ВЧ. Якщо зображення має нечіткі межі (низьку чіткість), то його можна профільтрувати ВЧ-фільтром, який підкреслить перепади яскравості на зображенні й зробить його чіткішим. Для ВЧ-фільтрації зазвичай використовують такі типові маски:

$$H = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}, \quad H = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}, \quad H = \begin{bmatrix} 1 & -2 & 1 \\ -2 & 5 & -2 \\ 1 & -2 & 1 \end{bmatrix}.$$

Нормування є необхідним для того, щоб звести значення відгуку фільтра до діапазону вхідних даних. Нормувальний коефіцієнт визначаємо з умови рівності одиниці суми всіх коефіцієнтів ІПХ фільтра.

Наведемо приклади різних фільтрів для оброблення зображень з використанням в програмних кодах **Python** алгоритмів бібліотеки **OpenCV**.

Фільтрація НЧ (low pass filter). Наведемо програмний код низькочастотного фільтра з масками розміром 3x3 і 5x5 пікселів:

```
import cv2
import numpy as np

img = cv2.imread('5.jpg')

kernel_identity = np.array([[0,0,0], [0,1,0], [0,0,0]])
rows, cols = img.shape[:2]

kernel_identity = np.array([[0,0,0], [0,1,0], [0,0,0]])
kernel_3x3 = np.ones((3,3), np.float32) / 9.0
kernel_5x5 = np.ones((5,5), np.float32) / 25.0

cv2.imshow('Original', img)

output = cv2.filter2D(img, -1, kernel_identity)
cv2.imshow('Identity filter', output)

output = cv2.filter2D(img, -1, kernel_3x3)
cv2.imshow('3x3 filter', output)

output = cv2.filter2D(img, -1, kernel_5x5)
cv2.imshow('5x5 filter', output)
```

```
cv2.waitKey(0)
```

Результати фільтрації показано на рис. 3.11. Добре видно, що при збільшенні розміру маски (рис. 3.11 в, г) збільшується міра розмиття меж зображення.

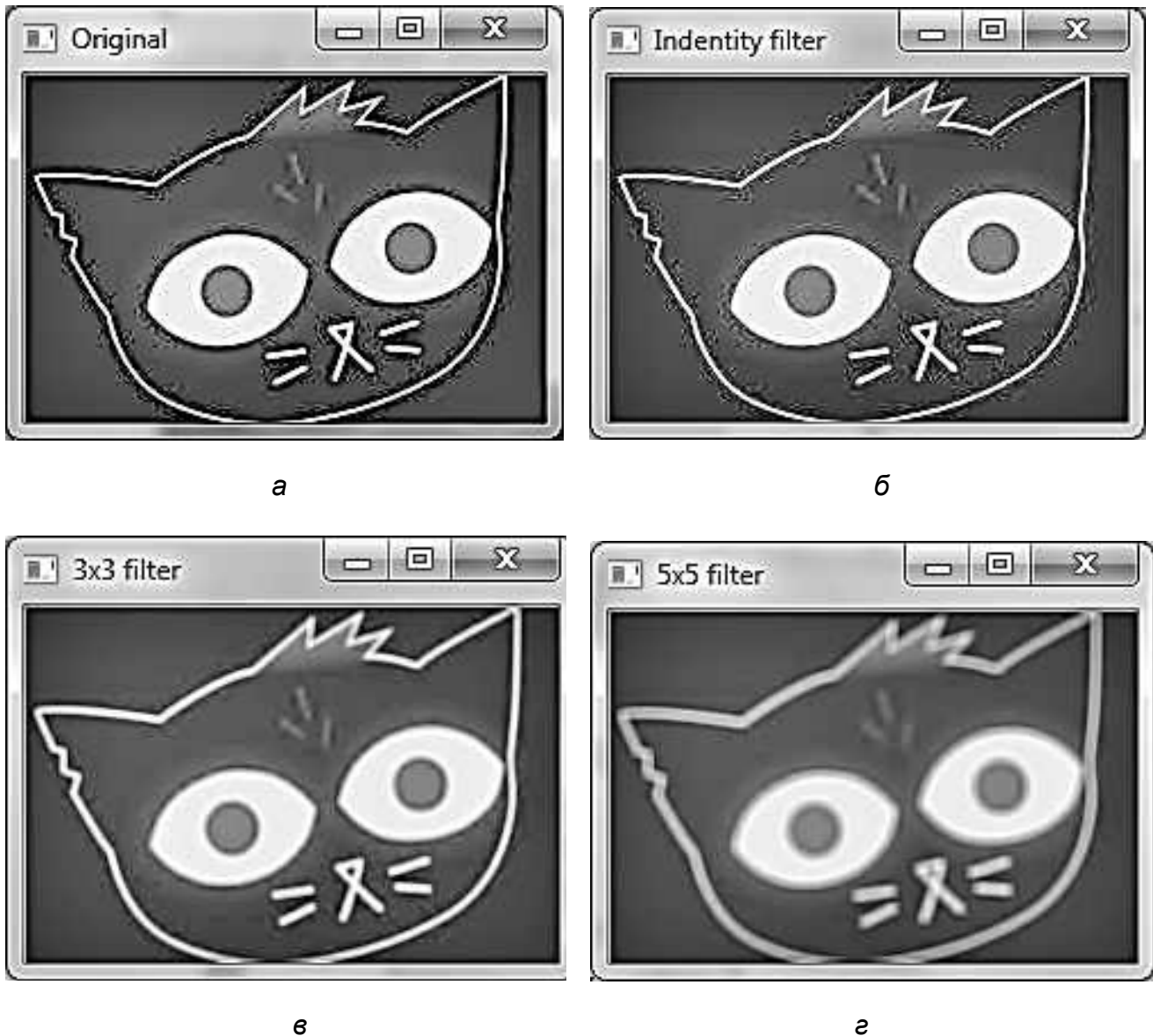


Рис. 3.11. Оброблення зображення фільтрами НЧ з масками різного розміру

Фільтрація ВЧ (high pass filter) широко використовується, якщо необхідно підкреслити розмиті межі об'єктів на зображенні. Це добре видно на рис. 3.12, б. Зверніть увагу, що при написанні програмних кодів фільтрації у всіх прикладах, окрім підключення бібліотеки **OpenCV** (`import cv2`), у програму необхідно імпортувати модуль `numpy` для формування маски фільтра (`kernel`):

```
import cv2
import numpy as np
```

```
img = cv2.imread('6.jpg')
cv2.imshow('Original', img)

kernel_identity = np.array([[ -1, -1, -1], [ -1, 9, -1], [ -1, -1, -1]])
kernel_3x3 = np.ones((3,3), np.float32)

output = cv2.filter2D(img, -1, kernel_identity)
cv2.imshow('3x3 filter', output)

cv2.waitKey(0)
```



а

б

Рис. 3.12. Оброблення зображення ВЧ-фільтром

Фільтрація з ефектом руху (Motion Blur) зазвичай використовується для створення ефекту руху, що істотно розмиває зображення. Приклад програмного коду такої фільтрації наведено нижче, а результат оброблення показано на рис. 3.13, б:

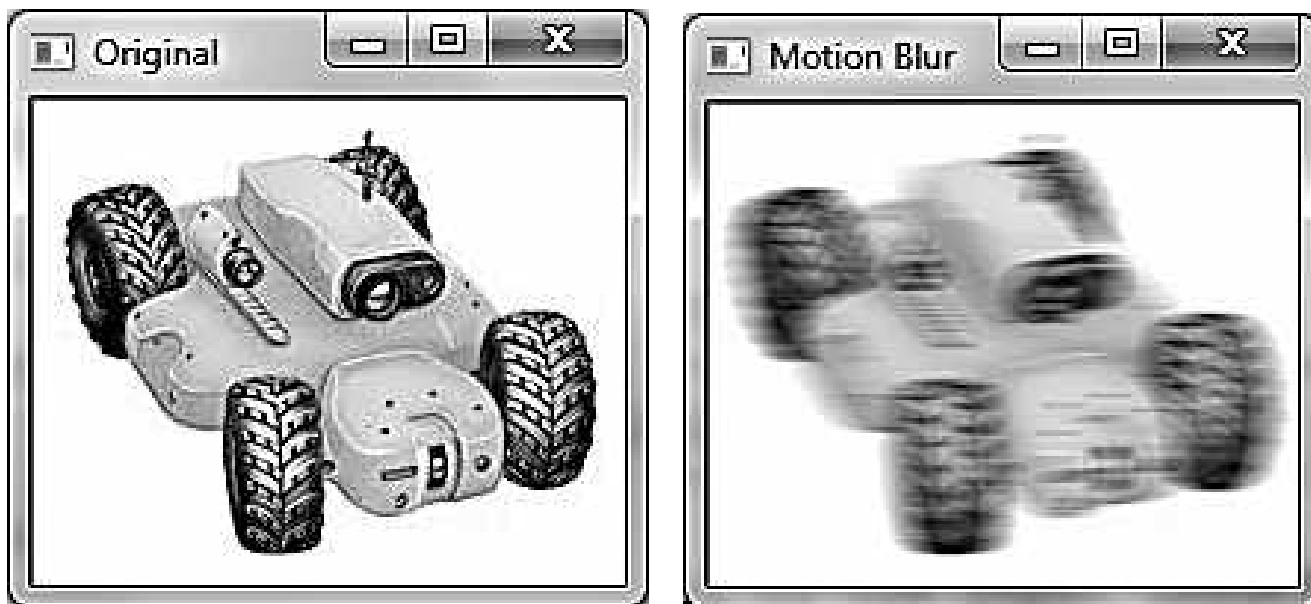
```
import cv2
import numpy as np

img = cv2.imread('2.jpg')
cv2.imshow('Original', img)

size = 15

# generating the kernel
kernel_motion_blur = np.zeros((size, size))
kernel_motion_blur[int((size-1)/2), :] = np.ones(size)
kernel_motion_blur = kernel_motion_blur/size
```

```
# applying the kernel to the input image
output = cv2.filter2D(img, -1, kernel_motion_blur)
cv2.imshow('Motion Blur', output)
cv2.waitKey(0)
```



а

б

Рис. 3.13. Варіанти фільтрації з ефектом руху

Фільтрація з підкресленням меж (sharpening filter), що є одним із різновидів ВЧ-фільтрації, створює різні візуальні ефекти (рис. 3.14) унаслідок підкреслення меж зображення:

```
import cv2
import numpy as np

img = cv2.imread('3.jpg')
cv2.imshow('Original', img)

# generating the kernels
kernel_sharpen_1 = np.array([[ -1, -1, -1], [ -1, 9, -1], [ -1, -1, -1]])
kernel_sharpen_2 = np.array([[ 1, 1, 1], [ 1, -7, 1], [ 1, 1, 1]])
kernel_sharpen_3 = np.array([[ -1, -1, -1, -1, -1],
                               [ -1, 2, 2, 2, -1],
                               [ -1, 2, 8, 2, -1],
                               [ -1, 2, 2, 2, -1],
                               [ -1, -1, -1, -1, -1]]) / 8.0

# applying different kernels to the input image
output_1 = cv2.filter2D(img, -1, kernel_sharpen_1)
output_2 = cv2.filter2D(img, -1, kernel_sharpen_2)
```

```

output_3 = cv2.filter2D(img, -1, kernel_sharpen_3)

cv2.imshow('Sharpening', output_1)
cv2.imshow('Excessive Sharpening', output_2)
cv2.imshow('Edge Enhancement', output_3)

```

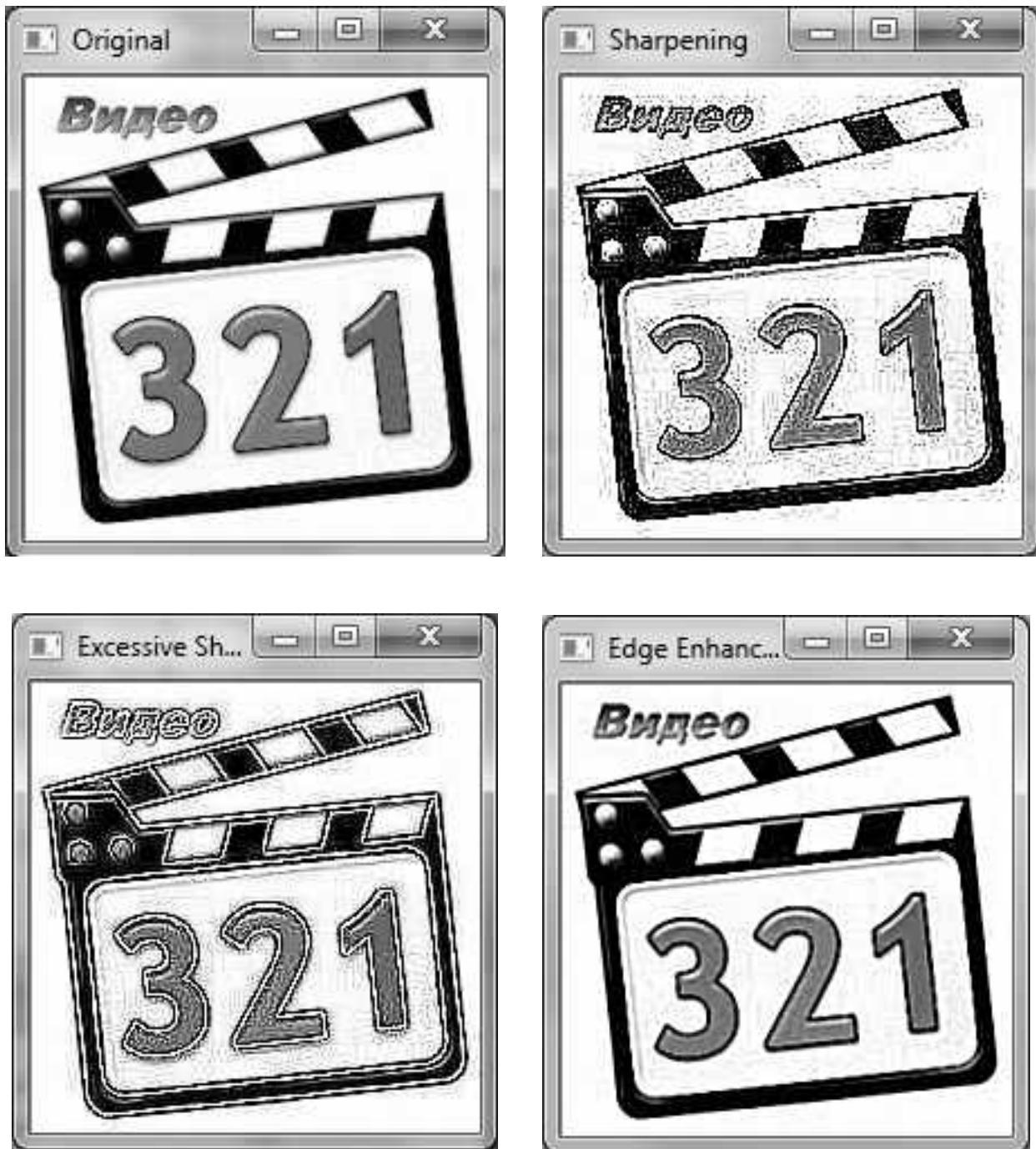


Рис. 3.14. Варіанти фільтрації з підкресленням меж

Фільтрація з перетворенням зображення на рельєфне (embossing filter) базується на поданні фрагментів зображення з рівномірною яскравістю пікселями сірого кольору, а областей зі значними пере-

падами яскравості – пікселями білого кольору. Це створює ефект тиснення й об'ємності зображення. Змінити міру тиснення можна за допомогою різного вигляду фільтрувальних масок (`kernel_emboss`). Зазначимо також, що перед процедурою фільтрації необхідно перетворити кольорове зображення на напівтонове за допомогою функції `cv2.COLOR_BGR2GRAY`:

```
import cv2
import numpy as np

img_emboss_input = cv2.imread('7.jpg')

# generating the kernels
kernel_emboss_1 = np.array([[0,-1,-1],
                             [1,0,-1],
                             [1,1,0]])
kernel_emboss_2 = np.array([[-1,-1,0],
                             [-1,0,1],
                             [0,1,1]])
kernel_emboss_3 = np.array([[1,0,0],
                             [0,0,0],
                             [0,0,-1]])

# converting the image to grayscale
gray_img =
cv2.cvtColor(img_emboss_input,cv2.COLOR_BGR2GRAY)

# applying the kernels to the grayscale image and adding
the offset
output_1 = cv2.filter2D(gray_img, -1, kernel_emboss_1) +
128
output_2 = cv2.filter2D(gray_img, -1, kernel_emboss_2) +
128
output_3 = cv2.filter2D(gray_img, -1, kernel_emboss_3) +
128

cv2.imshow('Input', img_emboss_input)
cv2.imshow('Embossing - South West', output_1)
cv2.imshow('Embossing - South East', output_2)
cv2.imshow('Embossing - North West', output_3)

cv2.waitKey(0)
```

Результати фільтрації показано на рис. 3.15.

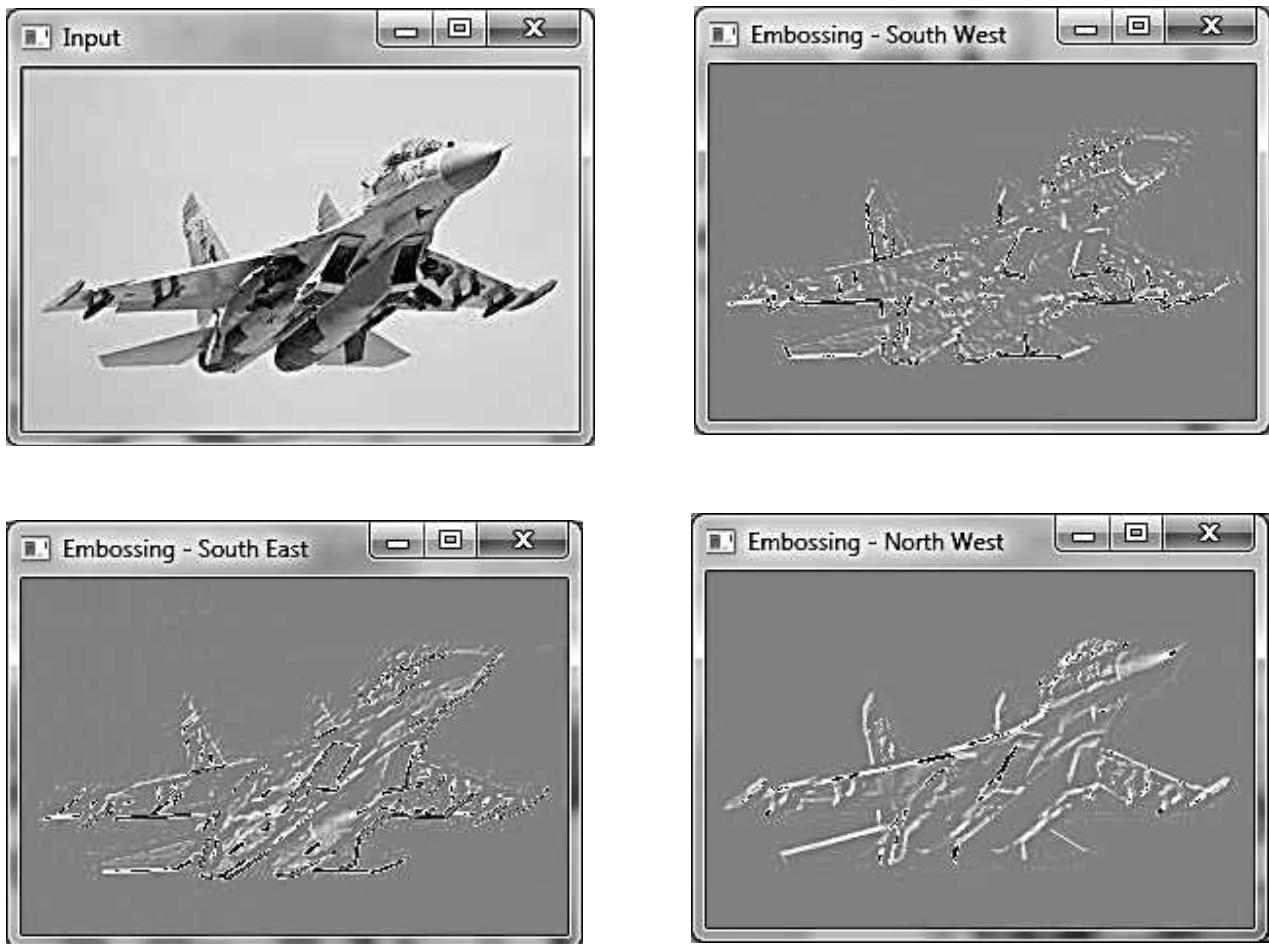


Рис. 3.15. Фільтрація з перетворенням зображення на рельєфне

Фільтрація гаусівськими й білатеральними фільтрами (Gaussian and Bilateral filters). Нагадаємо, що фільтр Гаусса має імпульсну перехідну характеристику вигляду

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}},$$

де x – розміри фільтрувальної маски по горизонтальній осі; y – розміри маски по вертикальній осі; σ – стандартне відхилення розподілу Гаусса, або ж міра розмиття. Фільтрація Гаусса є фільтрацією згладжувального типу, але, на відміну від прямокутної фільтрації, розмиття буде більш рівномірно розподілятися від центральної точки до країв. Фільтр Гаусса є високочастотним фільтром, а отже, його вигідно застосовувати для підвищення чіткості зображення.

Білатеральна фільтрація, що розширює поняття «згладжування Гаусса», підвищує якість фільтра. Пікселі, які сильно відрізняються за інтенсивністю від центрального пікселя, збільшуються меншою мірою, навіть не зважаючи на те, що вони можуть знаходитися в безпосередній близькості до центрального пікселя, що фактично є викривленням нелінійного фільтра Гаусса.

Наведемо приклад гаусівської і білатеральної фільтрації, використовуючи функції `cv2.GaussianBlur` і `cv2.bilateralFilter`. При цьому застосуємо два варіанти фільтрувальної маски для фільтра Гаусса – розмірами 7×7 і 15×15 . Код цієї програми наведено нижче:

```
import cv2
import numpy as np

img = cv2.imread('21.jpg')
img_gaussian = cv2.GaussianBlur(img, (7,7), 0)
img_gaussian1 = cv2.GaussianBlur(img, (15,15), 0)
img_bilateral = cv2.bilateralFilter(img, 13, 70, 50)

cv2.imshow('Input', img)
cv2.imshow('Gaussian filter', img_gaussian)
cv2.imshow('Gaussian filter 15x15', img_gaussian1)
cv2.imshow('Bilateral filter', img_bilateral)

cv2.waitKey()
```

Результати оброблення зображено на рис. 3.16.



Рис. 3.16. Використання гаусівських і білатеральних фільтрів

3.4. Гістограми розподілу яскравості зображень

Однією з найбільш істотних характеристик зображення є характер розподілу яскравості пікселів. Цей розподіл прийнято кількісно характеризувати гістограмою (рис. 3.17). На графіку вісь абсцис є шкалою яскравостей – у цьому випадку від 0 (чорні пікселі) до максимуму 255 (білі пікселі). Це відповідає поданню напівтонових зображень у форматі `uint8` з діапазоном значень яскравості $[0, 255]$. По осі ординат відкладають кількість пікселів, що відповідають кожному значенню яскравості. Це допомагає візуально оцінити домінуючий діапазон яскравостей на реальному зображенні. Так, на рис. 3.17 добре видно, що на зображенні переважають темні (погано освітлені) фрагменти.

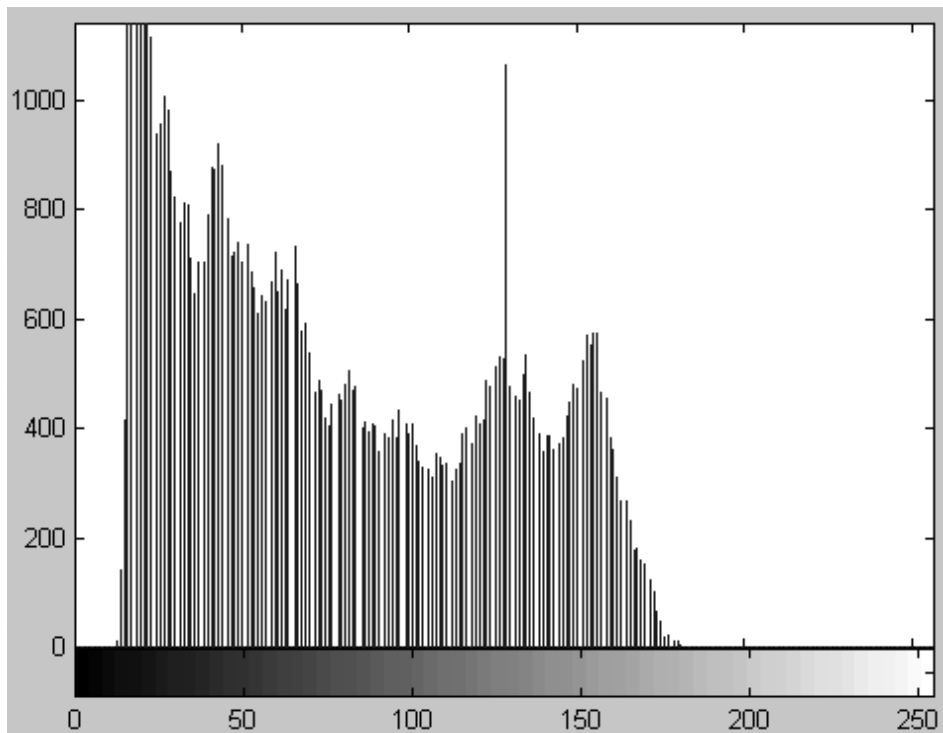


Рис. 3.17. Гістограма розподілу яскравостей

Урахування характеру розподілу яскравостей дає можливість провести перетворення, яке допомагає поліпшити контрастність зображення. Ця процедура сприяє вирівнюванню гістограми у всьому діапазоні яскравостей і має назву «еквалізація». Окрім того, можна здійснити бінаризацію зображення з відсіканням за заданим порогом яскравості, дає змогу просторово локалізувати в площині зображення фрагменти з великим або малим рівнем освітленості. У системах технічного зору бінаризація з високим порогом відсікання широко використовується для оцінювання положення спостережуваного об'єкта в кадрі.

Наведемо приклад програмного коду побудови гістограм зображення, включеного в інсталяцію бібліотеки **OpenCV** для спільної роботи з програмою **Python** як демонстраційний приклад:

```
#!/usr/bin/env python

''' This is a sample for histogram plotting for RGB images
and grayscale images for better understanding of colour
distribution

Benefit : Learn how to draw histogram of images
          Get familiar with cv2.calcHist,
cv2.equalizeHist,cv2.normalize and some drawing functions

Level : Beginner or Intermediate

Functions : 1) hist_curve : returns histogram of an image
drawn as curves
            2) hist_lines : return histogram of an image
drawn as bins ( only for grayscale images )

Usage : python hist.py <image_file>

Abid Rahman 3/14/12 debug Gary Bradski
'''

# Python 2/3 compatibility
from __future__ import print_function

import cv2
import numpy as np

bins = np.arange(256).reshape(256,1)

def hist_curve(im):
    h = np.zeros((300,256,3))
    if len(im.shape) == 2:
        color = [(255,255,255)]
    elif im.shape[2] == 3:
        color = [ (255,0,0), (0,255,0), (0,0,255) ]
    for ch, col in enumerate(color):
        hist_item =
cv2.calcHist([im],[ch],None,[256],[0,256])

cv2.normalize(hist_item,hist_item,0,255,cv2.NORM_MINMAX)
        hist=np.int32(np.around(hist_item))
        pts = np.int32(np.column_stack((bins,hist)))
        cv2.polylines(h,[pts],False,col)
    y=np.flipud(h)
    return y

def hist_lines(im):
    h = np.zeros((300,256,3))
```

```

    if len(im.shape)!=2:
        print("hist_lines applicable only for grayscale
images")
        #print("so converting image to grayscale for rep-
resentation")
        im = cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
        hist_item = cv2.calcHist([im],[0],None,[256],[0,256])

cv2.normalize(hist_item,hist_item,0,255,cv2.NORM_MINMAX)
hist=np.int32(np.around(hist_item))
for x,y in enumerate(hist):
    cv2.line(h,(x,0),(x,y),(255,255,255))
y = np.flipud(h)
return y

if __name__ == '__main__':
    import sys

    if len(sys.argv)>1:
        fname = sys.argv[1]
    else :
        fname = '../data/36.jpg'
        print("usage : python hist.py <image_file>")

    im = cv2.imread(fname)

    if im is None:
        print('Failed to load image file:', fname)
        sys.exit(1)

    gray = cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
    print('''' Histogram plotting \n
Keymap :\n
a - show histogram for color image in curve mode \n
b - show histogram in bin mode \n
c - show equalized histogram (always in bin mode) \n
d - show histogram for color image in curve mode \n
e - show histogram for a normalized image in curve
mode \n
Esc - exit \n
''')

    cv2.imshow('image',im)
    while True:
        k = cv2.waitKey(0)
        if k == ord('a'):
            curve = hist_curve(im)

```

```

        cv2.imshow('histogram',curve)
        cv2.imshow('image',im)
        print('a')
    elif k == ord('b'):
        print('b')
        lines = hist_lines(im)
        cv2.imshow('histogram',lines)
        cv2.imshow('image',gray)
    elif k == ord('c'):
        print('c')
        equ = cv2.equalizeHist(gray)
        lines = hist_lines(equ)
        cv2.imshow('histogram',lines)
        cv2.imshow('image',equ)
    elif k == ord('d'):
        print('d')
        curve = hist_curve(gray)
        cv2.imshow('histogram',curve)
        cv2.imshow('image',gray)
    elif k == ord('e'):
        print('e')
        norm = cv2.normalize(gray, gray, alpha =
0,beta = 255,norm_type = cv2.NORM_MINMAX)
        lines = hist_lines(norm)
        cv2.imshow('histogram',lines)
        cv2.imshow('image',norm)
    elif k == 27:
        print('ESC')
        cv2.destroyAllWindows()
        break
cv2.destroyAllWindows()

```

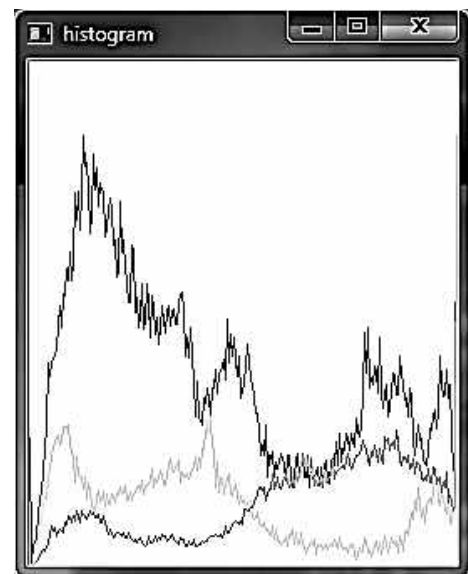
Результати роботи програми показано на рис. 3.18. У програмі закладено можливість перетворення початкового зображення на напівтонове зображення та його еквалізація. Спочатку відбувається візуалізація початкового зображення (рис. 3.18, а), у вікні Python 3.6.1 Shell пропонується інтерактивне меню:

- a - show histogram for color image in curve mode
- b - show histogram in bin mode
- c - show equalized histogram (always in bin mode)
- d - show histogram for color image in curve mode
- e - show histogram normalized image in curve mode.

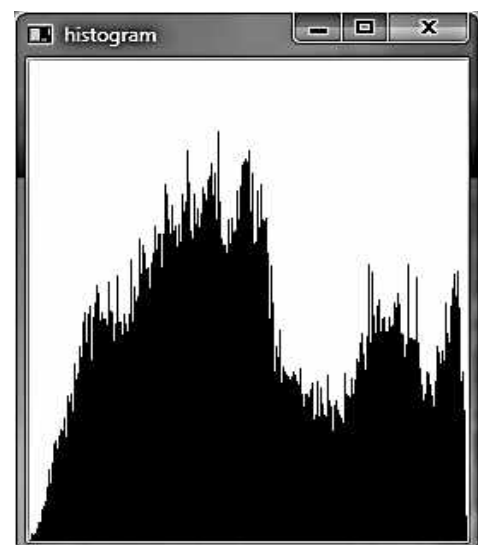
По черзі натискаючи клавіші а – е (рис. 3.18, б – е), можна вибрати потрібне перетворення початкового зображення та його гістограму.



a



б

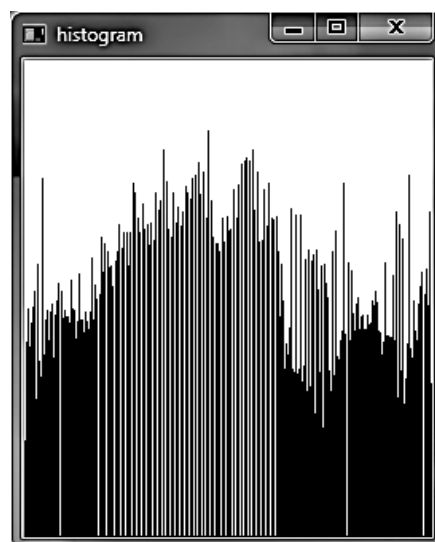


в

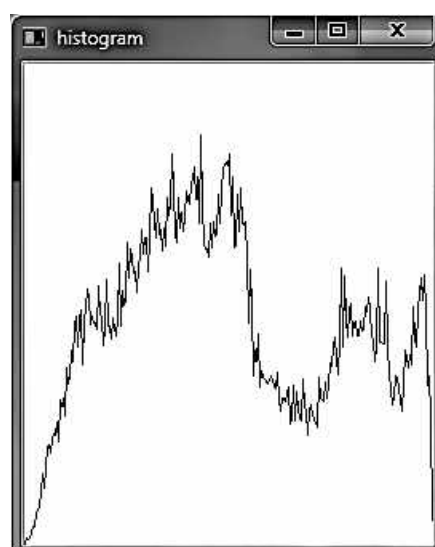
Рис. 3.18. Різні гістограми зображення



а



б



в

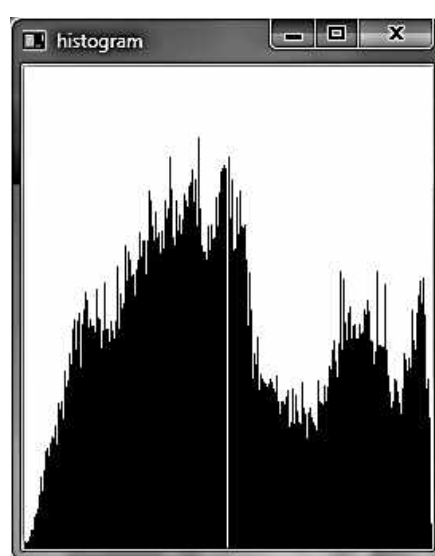


Рис. 3.18. Закінчення

Поліпшення контрасту напівтонових зображень (Enhancing the contrast in an image). Наведемо найпростіший приклад підвищення контрастності зображення. Зазвичай еквалізація виконується із застосуванням функції `histeq = cv2.equalizeHist(img)` на RGB-зображенні, перетвореному на напівтонове. Програмний код такого оброблення:

```
import cv2
import numpy as np
img = cv2.imread('1.jpg', 0)
# equalize the histogram of the input image
histeq = cv2.equalizeHist(img)
cv2.imshow('Input', img)
cv2.imshow('Histogram equalized', histeq)
cv2.waitKey(0)
```

Результат оброблення зображення показано на рис. 3.19, б.



а

б

Рис. 3.19. Результат контрастування зображення

Підвищення контрасту повнокольорових зображень. Набагато більший практичний інтерес становить процедура підвищення контрасту кольорових зображень. Це можна зробити шляхом перетворення RGB-зображення на зображення у форматі YUV. Нагадаємо, що це колірна модель, у якій колір подається як три компоненти: одна – яскравість (Y); дві інші – кольорорізницеві (U і V). На зображенні у форматі YUV еквалізації піддається лише компонента яскравості (Y) за допомогою функції `img_yuv[:, :, 0] = cv2.equalizeHist(img_yuv[:, :, 0])`, а потім здійснюється зворотне перетворення з формату YUV на формат RGB. При цьому баланс кольору зберігається без змін, оскільки кольорорізницеві компоненти U і V перетворенням не піддавалися.

Програмний код оброблення має такий вигляд:

```

import cv2
import numpy as np

img = cv2.imread('input.jpg')
img_yuv = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)
# equalize the histogram of the Y channel
img_yuv[:, :, 0] = cv2.equalizeHist(img_yuv[:, :, 0])
# convert the YUV image back to RGB format
img_output = cv2.cvtColor(img_yuv, cv2.COLOR_YUV2BGR)

cv2.imshow('Color input image', img)
cv2.imshow('Histogram equalized', img_output)

cv2.waitKey(0)

```

Результати контрастування повнокольорового зображення за цією методикою показано на рис. 3.20, б.



Рис. 3.20. Підвищення контрасту повнокольорових зображень

3.5. Бінаризація зображень з відсіканням за порогом яскравості

Одним із важливих методів оброблення зображень у системах технічного зору є процедура їх бінаризації з відсіканням за порогом яскравості. Це дає можливість подати параметри зображення в бінарному вигляді й проводити з ними математичні обчислення. В алгоритмі бінаризації використовується функція відсікання бібліотеки **OpenCV** (`thresh, im_bw`) = `cv2.threshold(im_gray, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)` з регульованим порогом яскравості `thresh`. При бінаризації напівтонового зображення з діапазоном значень пікселів $0 \dots 255$ параметр `thresh` може змінюватися в межах від 10 до 254. Зазначимо, що при відсіканні за високим порогом яскравості на бінарному зображенні залишаться лише пікселі з максимальною яскравістю (білого кольору). Решта піксе-

лів матимуть нульові значення (чорний колір). Наведемо код цього перетворення (рис. 3.21):

```
import cv2

img = cv2.imread('34.jpg')
cv2.imshow('Input', img)

im_gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
cv2.imshow("Gray", im_gray)

(thresh, im_bw) = cv2.threshold(im_gray, 0, 255,
cv2.THRESH_BINARY | cv2.THRESH_OTSU)
thresh = 45
im_bw = cv2.threshold(im_gray, thresh, 255,
cv2.THRESH_BINARY)[1]
cv2.imshow('Binary', im_bw)
```

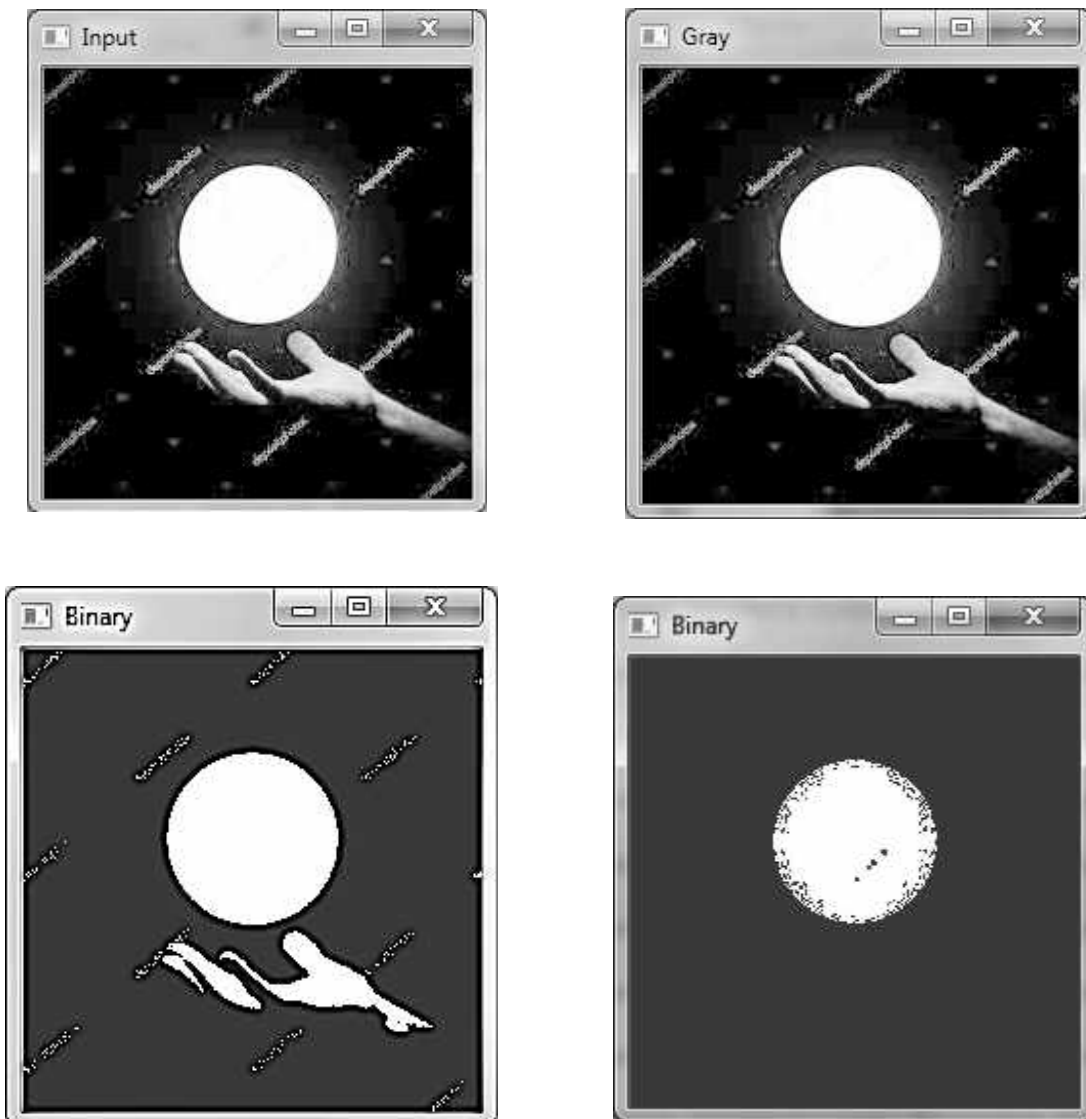


Рис. 3.21. Приклад бінаризації з різними порогоми яскравості

3.6. Афінні й проєктивні перетворення зображень

Нагадаємо основні теоретичні відомості про суть афінних перетворень. Поворот, змінення розмірів зображення, зміщення та інші перетворення належать до геометричних перетворень, які називають афінними. Афінне перетворення можна записати в матричній формі:

$$[x \ y \ 1] = [w \ z \ 1]T \begin{vmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t_{32} & 0 \end{vmatrix}.$$

Такою формулою можна задати стискання, поворот, перенесення або зміщення, відповідним чином визначаючи елементи матриці T . У табл. 3.1 наведено величини для здійснення різних перетворень.

Таблиця 3.1

Типи афінних перетворень		
Тип	Афінна матриця T	Координатні рівняння
Розтягування	$\begin{vmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{vmatrix}$	$X = S_x W,$ $Y = S_y Z$
Поворот	$\begin{vmatrix} \cos(a) & \sin(a) & 0 \\ -\sin(a) & \cos(a) & 0 \\ 0 & 0 & 1 \end{vmatrix}$	$x = w \cos(a) - z \sin(a),$ $y = w \sin(a) + z \cos(a)$
Зміщення (горизонтальне)	$\begin{vmatrix} 1 & 0 & 0 \\ a & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$	$x = W + aZ,$ $y = Z$
Зміщення (вертикальне)	$\begin{vmatrix} 1 & b & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$	$x = W,$ $y = bW + Z$
Перенесення	$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ S_x & S_y & 1 \end{vmatrix}$	$X = W + S_x,$ $Y = Z + S_y$

Розглянемо властивості функцій **OpenCV**, що забезпечують геометричні перетворення для змінення просторового положення елементів у зображенні.

Найширше в комп'ютерному зорі використовуються функції змінення розмірів зображень (`Image resize`) і повороту зображень (`Image rotation`). Слід мати на увазі, що просторові перетворення призводять до погіршення якості зображення.

Особливо це помітно на межах зображення. Тому процедура геометричних перетворень зазвичай доповнюється одним із видів інтерполяції. За допомогою функцій інтерполяції бібліотеки **OpenCV** `cv2.INTER_LINEAR`, `cv2.INTER_CUBIC`, `cv2.INTER_AREA` інтерполяція здійснюється або по білінійній поверхні, або по бікубичній.

Змінення розмірів зображення (Image scaling). Розглянемо програму для змінення розмірів зображення. При використанні функції `cv2.resize` необхідно вказати масштаб зміни розмірів зображення (fx і fy), який може бути більше або менше 1, а також вибрати потрібний тип інтерполяції даних:

```
import cv2
import numpy as np

img = cv2.imread('6.jpg')
cv2.imshow('Original', img)

img_scaled = cv2.resize(img, None, fx=1.5, fy=1.5, interpolation = cv2.INTER_LINEAR)
cv2.imshow('Scaling - Linear Interpolation', img_scaled)
img_scaled = cv2.resize(img, None, fx=1.5, fy=1.5, interpolation = cv2.INTER_CUBIC)
cv2.imshow('Scaling - Cubic Interpolation', img_scaled)
img_scaled = cv2.resize(img, None, fx=1.5, fy=1.5, interpolation = cv2.INTER_AREA)
cv2.imshow('Scaling - Skewed Size', img_scaled)
cv2.waitKey()
```

На рис. 3.22, б-а показано результати збільшення розмірів зображення в 1,5 раза з різними видами інтерполяції.

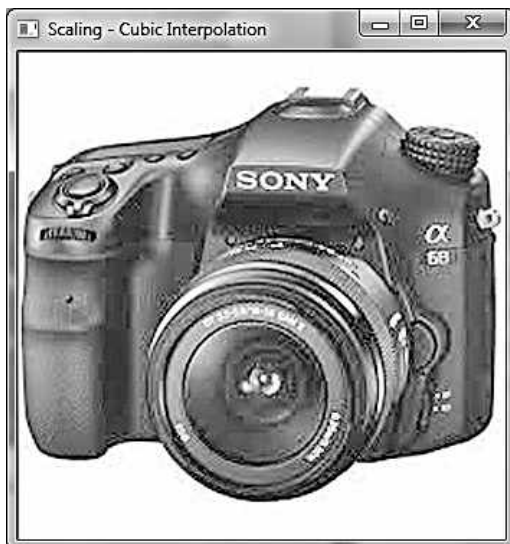


а



б

Рис. 3.22. Варіанти змінення розмірів зображення



б



г

Рис. 3.22. Закінчення

Поворот зображення (Image rotation). Для синтезу процедури повороту зображення в бібліотеці **OpenCV** формуються матриця повороту `cv2.getRotationMatrix2D` і функція афінних перетворень `cv2.warpAffine(img, rotation_matrix(num_cols, num_rows))`. Створюється зображення D , що відповідає поверненому початковому зображенню S , з використанням одного із зазначених методів інтерполяції. Кут повороту задається в градусах. Додатні значення цього параметра відповідають повороту проти годинникової стрілки, а від'ємні – за годинниковою стрілкою.

У загальному випадку кількість пікселів результуючого зображення D є більшою або дорівнює кількості пікселів початкового зображення S . Значення пікселів зображення D , для яких немає відповідних значень пікселів у зображенні S , встановлюються в 0, що для напівтонових зображень відповідає чорному кольору:

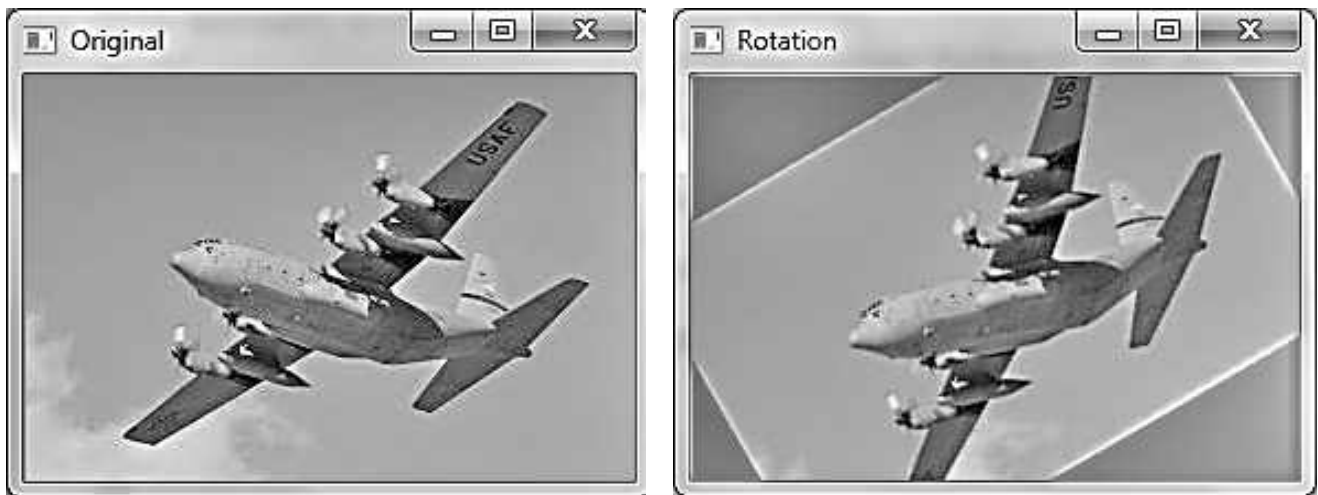
```
import cv2
import numpy as np

img = cv2.imread('16.jpg')
cv2.imshow('Original', img)
num_rows, num_cols = img.shape[:2]

rotation_matrix = cv2.getRotationMatrix2D((num_cols/2,
num_rows/2), 60, 1)
img_rotation = cv2.warpAffine(img, rotation_matrix,
(num_cols, num_rows))

cv2.imshow('Rotation', img_rotation)
cv2.waitKey(0)
```

Результат роботи коду з повороту зображення показано на рис. 3.23, б.



а

б

Рис. 3.23. Приклад повороту зображення

Паралельне перенесення зображення (*Image translation*). Здійснюється функцією афінного перетворення `cv2.warpAffine (img, translation_matrix (num_cols,num_rows))` з використанням матриці паралельного перенесення (`translation_matrix`), у якій у пікселях задається перенесення початкового зображення по рядках і стовпцях (рис. 3.24):

```
import cv2
import numpy as np
img = cv2.imread('1.jpg')
cv2.imshow(' Original ', img)
num_rows, num_cols = img.shape[:2]
translation_matrix = np.float32([ [1,0,70], [0,1,110] ])
img_translation = cv2.warpAffine(img, translation_matrix,
(num_cols, num_rows))
cv2.imshow('Translation', img_translation)
cv2.waitKey()
```

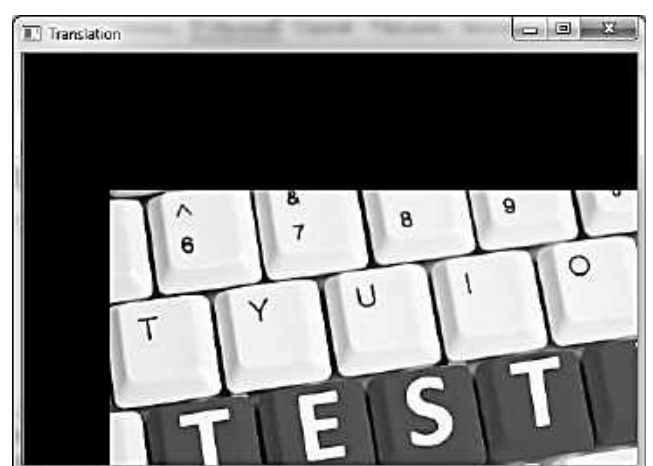


Рис. 3.24. Результати паралельного перенесення зображення

Скіс зображення (Image translation) і створення його дзеркальної копії. Ще один вид афінного перетворення – скіс прямокутного зображення. Наведемо приклад цього перетворення з використанням функції `cv2.getAffineTransform (src_points,dst_points)` (рис. 3.25, а, б):

```
import cv2
import numpy as np

img = cv2.imread('21.jpg')
rows, cols = img.shape[:2]

src_points = np.float32([[0,0], [cols-1,0], [0,rows-1]])
dst_points = np.float32([[0,0], [int(0.6*(cols-1)),0],
[int(0.4*(cols-1)),rows-1]])
affine_matrix = cv2.getAffineTransform(src_points,
dst_points)
img_output = cv2.warpAffine(img, affine_matrix,
(cols,rows))

cv2.imshow('Input', img)
cv2.imshow('Output', img_output)
cv2.waitKey()
```

Для створення дзеркального відображення в наведеному коді необхідно зробити таку заміну (рис. 3.25, в, г):

```
src_points = np.float32([[0,0], [cols-1,0], [0,rows-1]])
dst_points = np.float32([[cols-1,0], [0,0], [cols-1,
rows-1]])
```

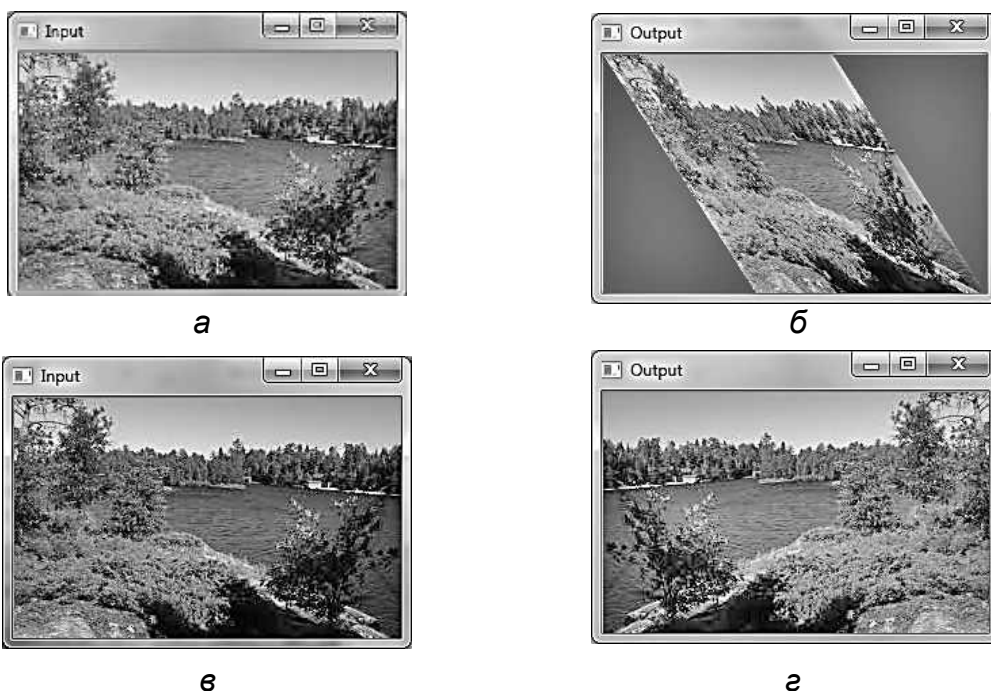


Рис. 3.25. Скіс зображення і створення його дзеркальної копії

Проективні перетворення зображення (Projective transformations). Як і при афінних перетвореннях, прямі лінії залишаються прямими, але паралельні лінії стають непаралельними з віддаленою точкою перетину.

Наведемо приклад програмного коду проективного перетворення зображення, результат якого показано на рис. 3.26, б:

```
import cv2
import numpy as np

img = cv2.imread('22.jpg')
rows, cols = img.shape[:2]

src_points = np.float32([[0,0], [cols-1,0], [0,rows-1],
[cols-1,rows-1]])
dst_points = np.float32([[0,0], [cols-1,0],
[int(0.33*cols),rows-1],
[int(0.66*cols),rows-1]])
projective_matrix =
cv2.getPerspectiveTransform(src_points, dst_points)
img_output = cv2.warpPerspective(img, projective_matrix,
(cols,rows))

cv2.imshow('Input', img)
cv2.imshow('Output', img_output)
cv2.waitKey()
```



Рис. 3.26. Приклад проективного перетворення зображення

Наведемо ще один приклад проективного перетворення, результати якого показано на рис. 3.27, б.

```
import cv2
import numpy as np

img = cv2.imread('21.jpg')
```

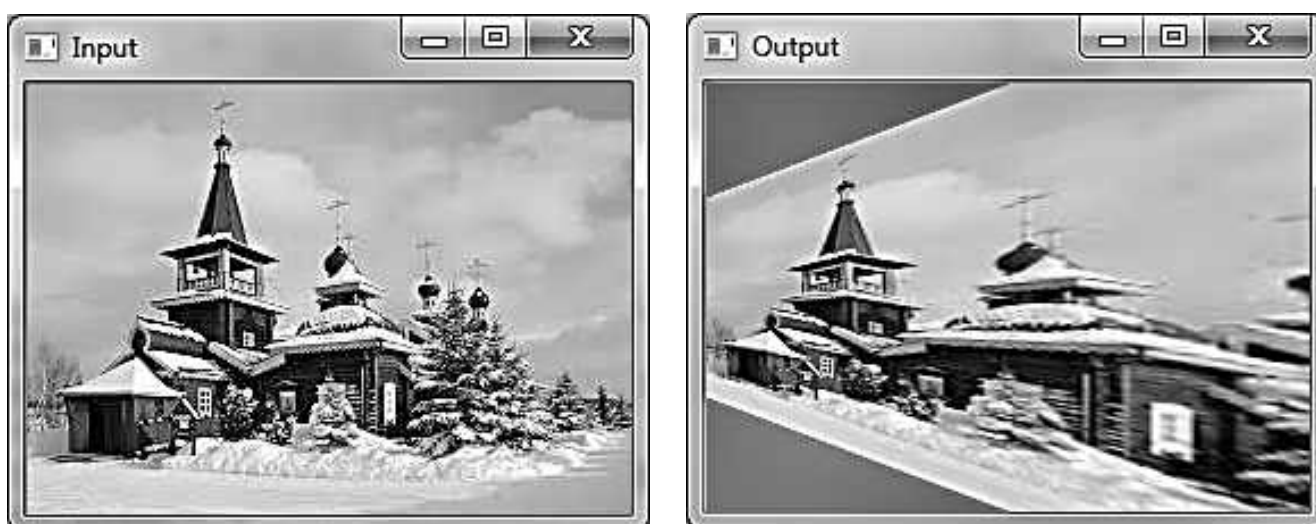
```

rows, cols = img.shape[:2]

src_points = np.float32([[0,0], [0,rows-1], [cols/2,0],
[cols/2,rows-1]])
dst_points = np.float32([[0,50], [0,rows-51], [cols/2,0],
[cols/2,rows-1]])
projective_matrix =
cv2.getPerspectiveTransform(src_points, dst_points)
img_output = cv2.warpPerspective(img, projective_matrix,
(cols,rows))

cv2.imshow('Input', img)
cv2.imshow('Output', img_output)
cv2.waitKey()

```



а

б

Рис. 3.27. Варіант проективного перетворення зображення

3.7. Детектування кутів на зображеннях у системах технічного зору

Детектування кутових точок є одним зі способів реалізації комп'ютерного зору. За допомогою цих алгоритмів у кадрі визначають характерні опорні точки для нерухомих об'єктів, які надалі можна використовувати під час розрахунку характеристик руху спостережуваних об'єктів. Існує декілька типів таких алгоритмів.

Для реалізації алгоритму пошуку кутів Харріса необхідно виміряти зміну інтенсивності пікселя (x, y) за допомогою зміщення невеликого квадратного вікна W розміром 3×3 із центром у (x, y) на один піксель у кожному з восьми можливих напрямів. Таку операцію слід виконати з кожним пікселем на зображенні. Для кожного пікселя зміна інтенсивності обчислюється за формулою

$$E(u, v) = \sum_{(x, y) \in W} w(x, y) [I(x + u, y + v) - I(x, y)]^2.$$

Функція $W(x, y)$ є функцією вікна W . Якщо $W(x, y) = 1$, то піксель з координатами (x, y) знаходиться в межах вікна, якщо ж $W(x, y) = 0$ – за його межами.

Запишемо формулу в матричному вигляді:

$$E(u, v) = [u, v] \begin{pmatrix} \sum I_x^2 & I_x I_y \\ I_x I_y & \sum I_y^2 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}.$$

Для невеликого значення $[u, v]$ отримуємо таке:

$$E(u, v) \cong [u, v] M \begin{bmatrix} u \\ v \end{bmatrix}, \quad M = \sum w(x, y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}.$$

Для кутових точок характерними є великі за модулем власні значення матриці M . Розрахунок власних значень цієї матриці є досить трудомістким завданням, тому Харріс додатково ввів міру відгуку

$$R = \det M - k(\text{trace} M)^2,$$

де k – величина, яка визначається емпірично, і її значення найчастіше лежить у діапазоні від 0.04 до 0.15. Таким чином, значення R є великим і додатним для кутових точок.

Після знаходження всіх кутових точок відбувається їх відсікання (відсікаються лише ті точки, значення R яких менше заданого порогу).

Детектор Харріса є інваріантним до поворотів і частково інваріантним до афінних перетворень. Його недоліком можна вважати залежність від масштабу.

Подібним чином працює й детектор кутів Ши-Томасі. Через обмеженість обсягу викладення принцип його роботи залишаємо на самотійний розгляд читача.

Наведемо приклади програмних кодів для детекторів кутів Харріса і Ши-Томасі. Результати роботи цих алгоритмів зображено на рис. 3.28, б і 3.29, б.

Детектор кутів Харріса (Harris Corner Detector). Приклад програмного коду, що реалізує алгоритм Харріса (рис. 3.28):

```
import cv2
import numpy as np

img = cv2.imread('25.jpg')
cv2.imshow('Original', img)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gray = np.float32(gray)
```

```

dst = cv2.cornerHarris(gray, 4,5, 0.04) # to detect only
sharp corners
#dst = cv2.cornerHarris(gray, 14, 5, 0.04) # to detect
soft corners

# Result is dilated for marking the corners
dst = cv2.dilate(dst,None)

# Threshold for an optimal value, it may vary depending on
the image.
img[dst > 0.01*dst.max()] = [0,0,0]

cv2.imshow('Harris Corners',img)
cv2.waitKey()

```

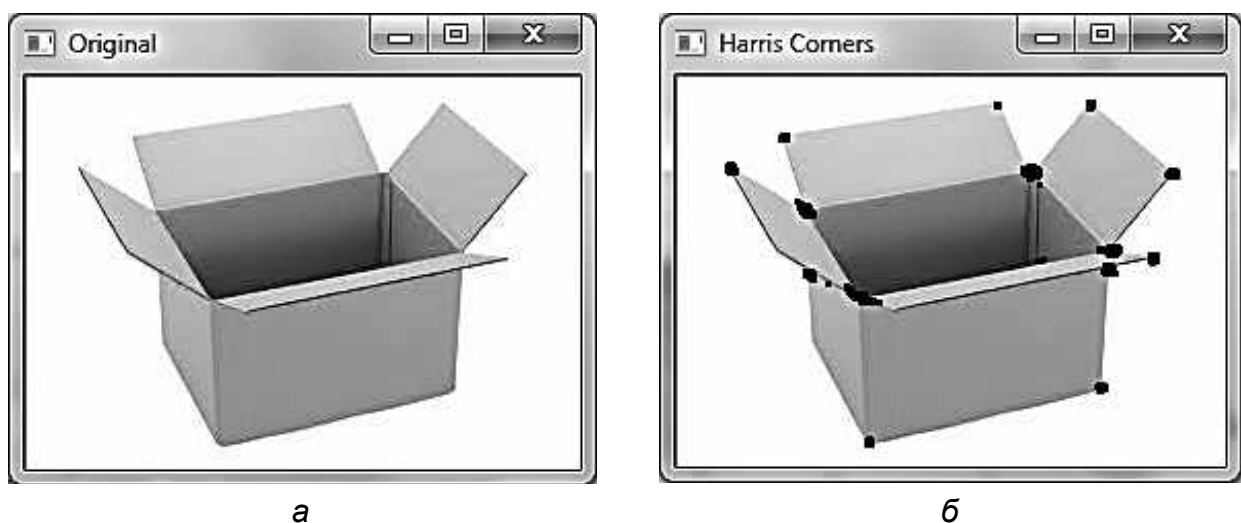


Рис. 3.28. Детектування кутів за алгоритмом Харріса

Детектор кутів Ши-Томасі (Shi-tomasi corner detector). Детектор Ши-Томасі дає аналогічні результати порівняно з алгоритмом Харріса (див. рис. 3.29).

```

import cv2
import numpy as np

img = cv2.imread('25.jpg')
cv2.imshow('Original', img)
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

corners = cv2.goodFeaturesToTrack(gray, 7, 0.05, 25)
corners = np.float32(corners)

for item in corners:
    x, y = item[0]
    cv2.circle(img, (x,y), 5, 255, -1)

cv2.imshow("Top 'k' features", img)
cv2.waitKey()

```

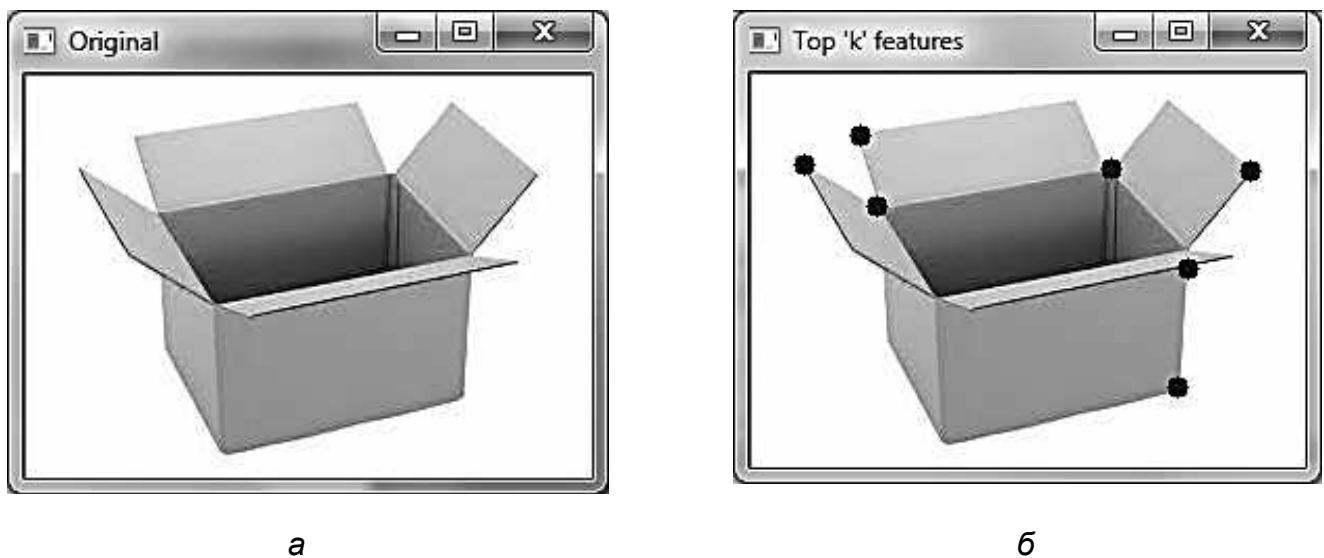


Рис. 3.29. Детектування кутів за алгоритмом Ши-Томасі

4. ВИКОРИСТАННЯ АЛГОРИТМІВ OPENCV ДЛЯ ОБРОБЛЕННЯ ВІДЕОДАНИХ

У розд. 3 описано кілька методів оброблення зображень на **Python** з використанням різних функцій бібліотеки **OpenCV**. Порівняльний аналіз ресурсів цієї бібліотеки з можливостями основних функцій і методів пакета модулів бібліотеки **Pillow** свідчить про те, що вони мають приблизно однакові функціональні можливості. Це стосується алгоритмів перетворення колірних просторів і фільтрації зображень, методів афінних і проєктивних перетворень та ін. Не менш ефективним для оброблення зображень є застосування додатка **Image Processing Toolbox** програми **Matlab**.

Проте це помилкове враження. Основною перевагою використання функцій бібліотеки **OpenCV** є істотно вища швидкодія. Це практично непомітно при обробленні окремих зображень, але при обробленні й аналізі відеопослідовностей цей чинник стає домінуючим. Більш того, у більшості випадків необхідно проводити оброблення в реальному масштабі часу, а це можливо лише з використанням алгоритмів **OpenCV**. Розглянемо ці переваги детальніше.

4.1. Оброблення відео з web-камери в реальному часі

Наведемо приклад реєстрації відео з web-камери з перетворенням повнокольорового зображення кадрів на напівтонове, а також виділення меж об'єктів у кадрі за алгоритмом Кенні. Наведемо код цієї програми (результати оброблення – на рис. 4.1):

```
import cv2

cap = cv2.VideoCapture(0)
```

```

# Check if the webcam is opened correctly
if not cap.isOpened():
    raise IOError("Cannot open webcam")

while True:
    ret, frame = cap.read()
    frame = cv2.resize(frame, None, fx=1.0, fy=1.0, inter-
polation=cv2.INTER_AREA)

    cv2.imshow('Input', frame)

    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    cv2.imshow('Gray', frame_gray)

    edges = cv2.Canny(frame_gray, 0, 99)
    cv2.imshow('Edges', edges)

    c = cv2.waitKey(1)
    if c == 27:
        break

cap.release()
cv2.destroyAllWindows()

```



Рис. 4.1. Оброблення відео з web-камери

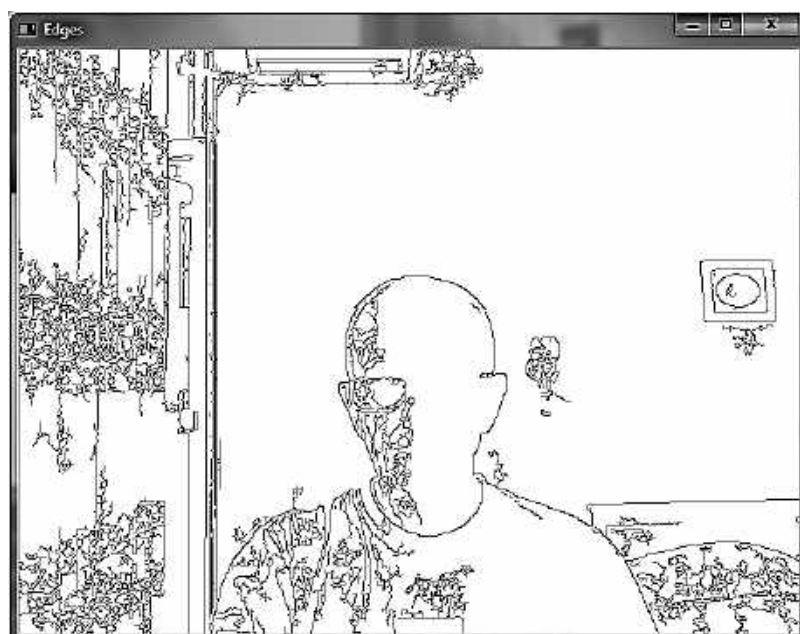


Рис. 4.1. Закінчення

Опишемо основні особливості написання цього програмного коду. По-перше, необхідно створити пристрій захоплення відео. Це здійснюється за допомогою функції `cap = cv2.VideoCapture(0)`. При цьому, якщо параметр захоплення дорівнює 0, за умовчанням використовується вбудована web-камера. При значенні параметра захоплення, що дорівнює 1, можна використовувати зовнішню камеру з USB-підключенням до комп'ютера.

Важливим чинником також є масштаб відображення поточного фрейма при перегляді відеопослідовності. Для цього необхідно в аргументі функції `cv2.resize(frame, None, fx=0.5, fy=0.5 ...)` указати потрібні розміри (наприклад, `fx=1.0, fy=1.0`).

Цикл читання відеоданих у цьому коді `ret, frame = cap.read()` продовжуватиметься доти, доки не станеться натиснення клавіші `Esc` на клавіатурі – це перерве процедуру відеозахоплення. Ця дія в програмі відображується умовою `if z == 27:`. Як відомо, ASCII значення `Esc` становить 27. Команда `cap.release()` закриває web-камеру.

У цьому програмному коді також виконуються перетворення початкової відеопослідовності на напівтонову за допомогою функції `frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)` і виділення меж об'єктів у кадрі за допомогою функції `edges = cv2.Canny(frame_gray, thresh)`. Нагадаємо, що параметр `thresh` визначає міру деталізації контурів меж розділення об'єкта і навколишнього фону.

Зверніть увагу на те, що всі процедури захоплення відображення й перетворення відеопослідовності, яка приймається з web-камери, виконуються в реальному масштабі часу. Це свідчить про високу ефективність використання функцій бібліотеки **OpenCV**.

4.2. Оброблення відео в інтерактивному режимі

Наведений вище приклад коду, що забезпечує виділення контурів об'єктів за методом Кенні, дає змогу ефективно вирішити поставлене завдання, але має істотний недолік – при зміні умов освітлення сцени необхідно підрегулювати параметр `thresh`, а отже, потрібно перервати роботу програми і ввести нові значення цього параметра. Набагато зручніше було б здійснювати регулювання параметрів оброблення відеоданих інтерактивно, не перериваючи процесу реєстрації.

Наведений нижче програмний код виконує таке завдання – здійснюється детектування вихідної відеопослідовності методом виділення меж об'єктів за методом Кенні (рис. 4.2). Регулювання значень параметра `thresh` виконується інтерактивно з використанням функцій

```
thrs1 = cv2.getTrackbarPos('thrs1', 'edge') i
thrs2 = cv2.getTrackbarPos('thrs2', 'edge').
```

Інтерактивна робота з відеопотоком (Interacting with a live video stream). Наведемо приклади інтерактивної взаємодії з відеопотоком, що приймається.

Приклад 1. Програмний код наведено нижче, а результати на рис. 4.2:

```
#!/usr/bin/env python
'''
This sample demonstrates Canny edge detection.

Usage:
    edge.py [<video source>]

    Trackbars control edge thresholds.
'''

# Python 2/3 compatibility
from __future__ import print_function

import cv2
import numpy as np

# relative module
import video

# built-in module
import sys

if __name__ == '__main__':
    print(__doc__)

    try:
        fn = sys.argv[1]
    except:
        fn = 0

    def nothing(*arg):
        pass

    cv2.namedWindow('edge')
    cv2.createTrackbar('thrs1', 'edge', 2000, 5000, nothing)
    cv2.createTrackbar('thrs2', 'edge', 4000, 5000, nothing)

    cap = video.create_capture(fn)
    while True:
        flag, img = cap.read()
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```

    thrs1 = cv2.getTrackbarPos('thrs1', 'edge')
    thrs2 = cv2.getTrackbarPos('thrs2', 'edge')
    edge = cv2.Canny(gray, thrs1, thrs2,
apertureSize=5)
    vis = img.copy()
    vis = np.uint8(vis/2.)
    vis[edge != 0] = (0, 255, 0)
    cv2.imshow('edge', vis)
    ch = cv2.waitKey(5)
    if ch == 27:
        break
cv2.destroyAllWindows()

```



Рис. 4.2. Інтерактивне детектування меж об'єктів за методом Кенні

У наведеному програмному коді для створення повзунів-регуляторів використовуються функції `cv2.createTrackbar`.

Приклад 2. У цьому випадку після захоплення відео й початку відображення в поточному кадрі за допомогою мишки виділяємо область інтересу і ведемо відображення в ній негативного відео (рис. 4.3):

```
import cv2
import numpy as np

def draw_rectangle(event, x, y, flags, params):
    global x_init, y_init, drawing, top_left_pt, bottom_right_pt

    if event == cv2.EVENT_LBUTTONDOWN:
        drawing = True
        x_init, y_init = x, y

    elif event == cv2.EVENT_MOUSEMOVE:
        if drawing:
            top_left_pt = (min(x_init, x), min(y_init, y))
            bottom_right_pt = (max(x_init, x), max(y_init, y))
            img[y_init:y, x_init:x] = 255 - img[y_init:y, x_init:x]

    elif event == cv2.EVENT_LBUTTONUP:
        drawing = False
        top_left_pt = (min(x_init, x), min(y_init, y))
        bottom_right_pt = (max(x_init, x), max(y_init, y))
        img[y_init:y, x_init:x] = 255 - img[y_init:y, x_init:x]

if __name__ == '__main__':
    drawing = False
    top_left_pt, bottom_right_pt = (-1, -1), (-1, -1)

    cap = cv2.VideoCapture(0)

    # Check if the webcam is opened correctly
    if not cap.isOpened():
        raise IOError("Cannot open webcam")

    cv2.namedWindow('Webcam')
    cv2.setMouseCallback('Webcam', draw_rectangle)

    while True:
        ret, frame = cap.read()
        img = cv2.resize(frame, None, fx=0.5, fy=0.5,
```

```

interpolation=cv2.INTER_AREA)
    (x0,y0), (x1,y1) = top_left_pt, bottom_right_pt
    img[y0:y1, x0:x1] = 255 - img[y0:y1, x0:x1]
    cv2.imshow('Webcam', img)

    c = cv2.waitKey(1)
    if c == 27:
        break

cap.release()
cv2.destroyAllWindows()

```

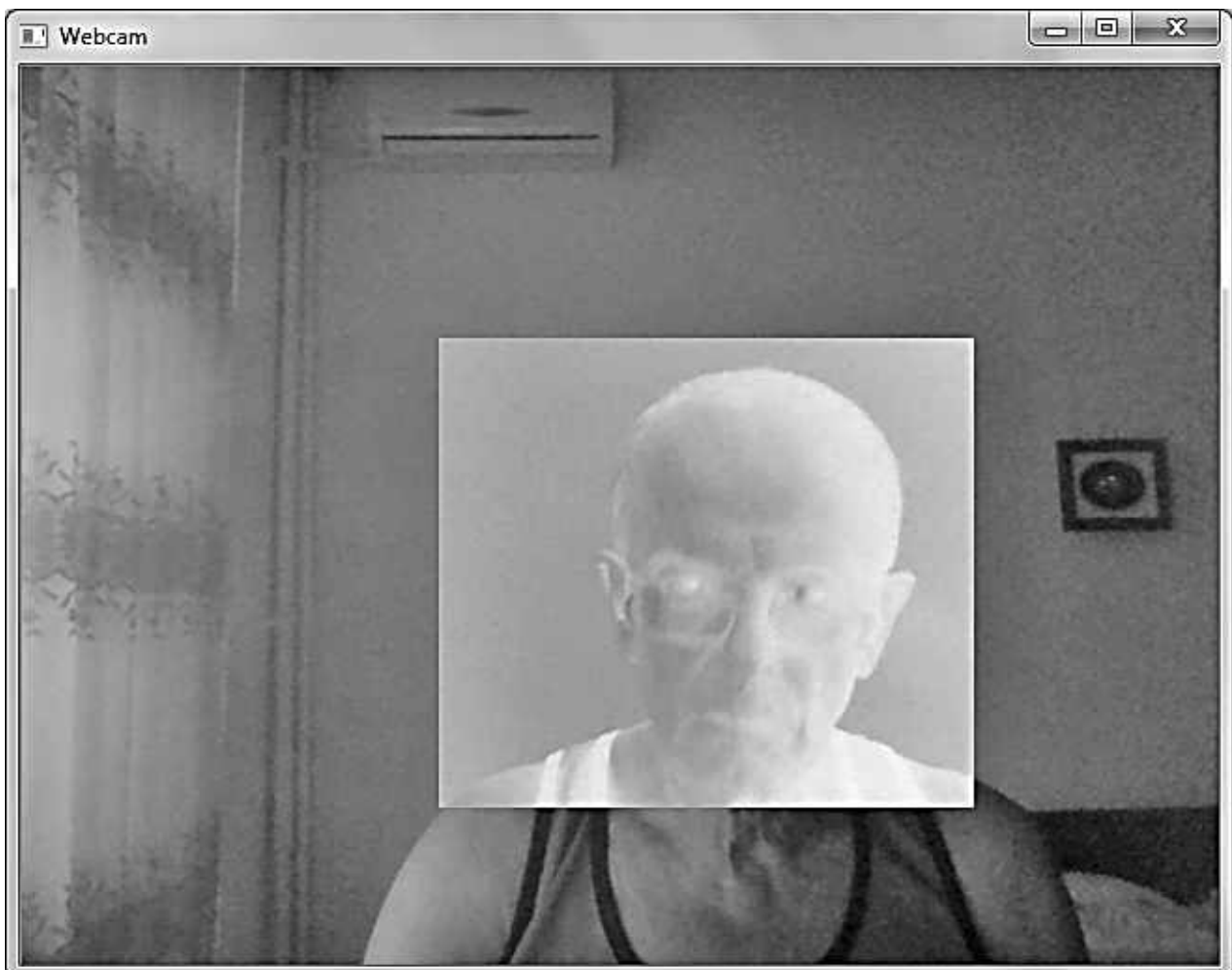


Рис. 4.3. Інтерактивне виділення області інтересу й відображення в ній негативного відео

Зазначимо, що в цьому випадку достатньо клацнути лівою кнопкою мишки поза полем області інтересу для відміни режиму перегляду в ній негативного зображення. Потім, не перериваючи процесу захоплення відеопотоку, можна виділити нову область інтересу і переглядати негативне зображення в ній далі.

4.3. Оцінювання руху об'єкта по відеопотоку

Одним із найактуальніших завдань технічного зору є визначення координат мобільного об'єкта в просторі на основі аналізу послідовності зображень відеокамери. Для вирішення завдання в послідовному потоці кадрів відео необхідно обчислювати, як і наскільки змістився мобільний об'єкт. Щоб уникнути надлишкових обчислень, на поточному зображенні необхідно відслідковувати лише деякі точки, зміна положення яких у наступному кадрі визначає зміну положення рухомого об'єкта.

Існує декілька способів визначення зміщень між двома сусідніми кадрами. Найбільш продуктивним є підхід, який дає змогу визначати зміщення не для фрагментів, а для кожного окремого пікселя. Зазвичай у літературі такий метод називають «оптичним потоком». Інколи застосовують спрощену версію такого алгоритму – обчислюють абсолютну різницю (похідні) для послідовності попереднього й поточного кадрів. Наведемо приклад використання такого алгоритму (рис. 4.4):

```
import cv2

# Compute the frame difference
def frame_diff(prev_frame, cur_frame, next_frame):
    # Absolute difference between current frame and next
    frame
    diff_frames1 = cv2.absdiff(next_frame, cur_frame)

    # Absolute difference between current frame and # pre-
    vious frame
    diff_frames2 = cv2.absdiff(cur_frame, prev_frame)

    # Return the result of bitwise 'AND' between the #
    above two resultant images
    return cv2.bitwise_and(diff_frames1, diff_frames2)

# Capture the frame from webcam
def get_frame(cap):
    # Capture the frame
    ret, frame = cap.read()

    # Resize the image
    frame = cv2.resize(frame, None, fx=scaling_factor,
                        fy=scaling_factor, interpola-
tion=cv2.INTER_AREA)

    # Return the grayscale image
    return cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)

if __name__ == '__main__':
    cap = cv2.VideoCapture(0)
```

```

scaling_factor = 1.0

prev_frame = get_frame(cap)
cur_frame = get_frame(cap)
next_frame = get_frame(cap)

# Iterate until the user presses the ESC key
while True:
    # Display the result of frame differencing
    cv2.imshow("Object Movement",
frame_diff(prev_frame, cur_frame,
next_frame))

    # Update the variables
    prev_frame = cur_frame
    cur_frame = next_frame
    ext_frame = get_frame(cap)

    # Check if the user pressed ESC
    key = cv2.waitKey(10)
    if key == 27:
        break

cv2.destroyAllWindows()

```

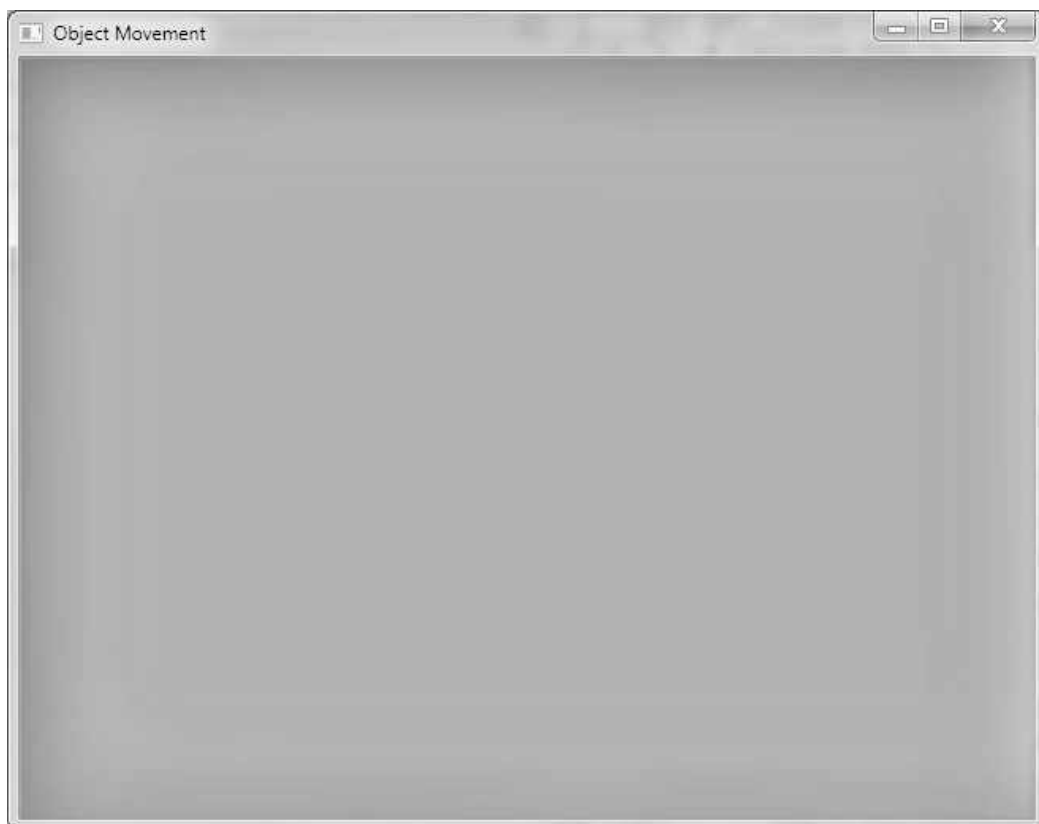


Рис. 4.4. Алгоритм оцінювання руху за різницею сусідніх кадрів

Розглянутий алгоритм має один істотний недолік – за відсутності руху в кадрі або при русі з малою швидкістю виникають труднощі візуалізації результату. Це добре видно на рис. 4.4.

Алгоритм оптичного потоку (Optical Flow) є найбільш універсальним. Детектування руху за допомогою алгоритму оптичного потоку зображено на рис. 4.5. Візуалізація результатів оброблення в цьому випадку є очевиднішою.

Розглянемо далі його програмний код:

```
import cv2
import numpy as np

def start_tracking():
    # Capture the input frame
    cap = cv2.VideoCapture(0)

    # Downsampling factor for the image
    scaling_factor = 1.0

    # Number of frames to keep in the buffer when you
    # are tracking. If you increase this number,
    # feature points will have more "inertia"
    num_frames_to_track = 5

    # Skip every 'n' frames. This is just to increase the
    speed.
    num_frames_jump = 2

    tracking_paths = []
    frame_index = 0

    # 'winSize' refers to the size of each patch. These
    patches
    # are the smallest blocks on which we operate and
    track
    # the feature points. You can read more about the pa-
    rameters
    # here: http://goo.gl/ulwqLk
    tracking_params = dict(winSize = (11, 11), maxLevel =
2,
                           criteria = (cv2.TERM_CRITERIA_EPS |
cv2.TERM_CRITERIA_COUNT, 10, 0.03))

    # Iterate until the user presses the ESC key
    while True:
        # read the input frame
        ret, frame = cap.read()
```

```

    # downsample the input frame
    frame = cv2.resize(frame, None, fx=scaling_factor,
                        fy=scaling_factor, interpolation=cv2.INTER_AREA)

    frame_gray = cv2.cvtColor(frame,
                                cv2.COLOR_BGR2GRAY)
    output_img = frame.copy()

    if len(tracking_paths) > 0:
        prev_img, current_img = prev_gray, frame_gray
        feature_points_0 = np.float32([tp[-1] for tp
in
tracking_paths]).reshape(-1, 1, 2)

        # Compute feature points using optical flow.
You can
        # refer to the documentation to learn more
about the
        # parameters here: http://goo.gl/t6P4SE
        feature_points_1, _, _ =
cv2.calcOpticalFlowPyrLK(prev_img,
current_img, feature_points_0,
                        None, **tracking_params)
        feature_points_0_rev, _, _ =
cv2.calcOpticalFlowPyrLK(current_img, prev_img, fea-
ture_points_1,
                        None, **tracking_params)
        # Compute the difference of the feature points
        diff_feature_points = abs(feature_points_0-
feature_points_0_rev).reshape(-1, 2).max(-1)

        # threshold and keep the good points
        good_points = diff_feature_points < 1

        new_tracking_paths = []

        for tp, (x, y), good_points_flag in
zip(tracking_paths,
                                feature_points_1.reshape(-1, 2),
good_points):
            if not good_points_flag:
                continue

            tp.append((x, y))

            # Using the queue structure i.e. first in,
            # first out

```

```

        if len(tp) > num_frames_to_track:
            del tp[0]

        new_tracking_paths.append(tp)

        # draw green circles on top of the output
image      cv2.circle(output_img, (x, y), 3, (0, 255,
0), -1)

        tracking_paths = new_tracking_paths

        # draw green lines on top of the output image
        cv2.polylines(output_img, [np.int32(tp) for tp
in
tracking_paths], False, (0, 150, 0))

        # 'if' condition to skip every 'n'th frame
        if not frame_index % num_frames_jump:
            mask = np.zeros_like(frame_gray)
            mask[:] = 255
            for x, y in [np.int32(tp[-1]) for tp in track-
ing_paths]:
                cv2.circle(mask, (x, y), 6, 0, -1)

            # Extract good features to track. You can
learn more
            # about the parameters here:
http://goo.gl/BI2Kml
            feature_points =
cv2.goodFeaturesToTrack(frame_gray,
                        mask = mask, maxCorners = 500,
qualityLevel = 0.3,
                        minDistance = 7, blockSize = 7)

            if feature_points is not None:
                for x, y in
np.float32(feature_points).reshape (-1, 2):
                    tracking_paths.append([(x, y)])

            frame_index += 1
            prev_gray = frame_gray

            cv2.imshow('Optical Flow', output_img)

            # Check if the user pressed the ESC key
            c = cv2.waitKey(1)
            if c == 27:
                break

```

```
if __name__ == '__main__':  
    start_tracking()  
    cv2.destroyAllWindows()
```

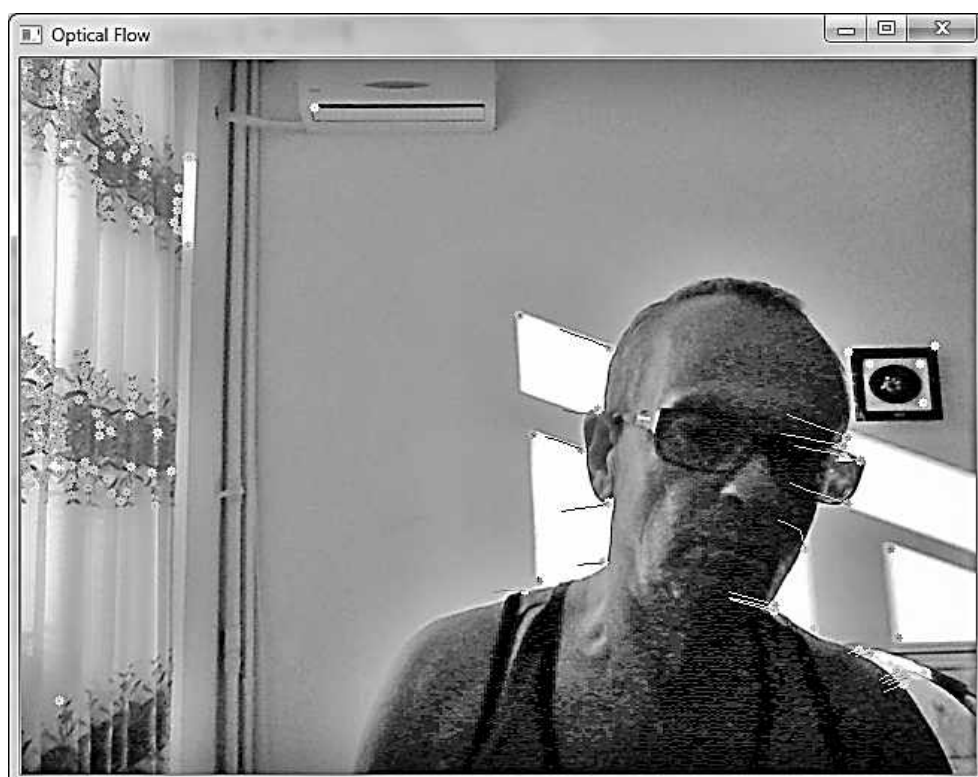
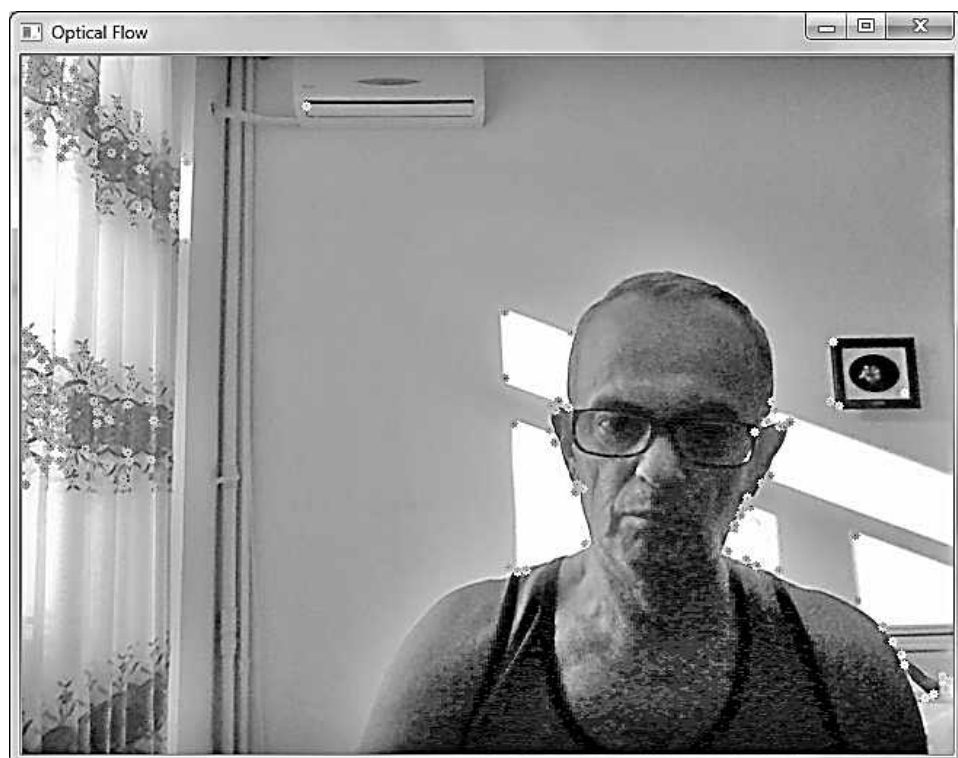


Рис. 4.5. Работа алгоритму оптичного потоку

4.4. Використання колірних просторів для оцінювання руху

Ще одним ефективним методом оцінювання руху об'єктів у кадрах відео є метод використання колірних просторів (***Colorspace based tracking***). Програмна реалізація цього методу, наведена нижче, дає змогу детектувати лише об'єкт певного кольору (тут – синього), а решта площини кадру замінюється нульовими значеннями (чорний колір). Це дуже спрощує завдання локалізації положення об'єкта в кадрі.

Програма виділення колірного простору (рис. 4.6):

```
import cv2
import numpy as np

# Capture the input frame from webcam
def get_frame(cap, scaling_factor):
    # Capture the frame from video capture object
    ret, frame = cap.read()

    # Resize the input frame
    frame = cv2.resize(frame, None, fx=scaling_factor,
                       fy=scaling_factor, interpolation=cv2.INTER_AREA)

    return frame

if __name__ == '__main__':
    cap = cv2.VideoCapture(0)
    scaling_factor = 1.0

    # Iterate until the user presses ESC key
    while True:
        frame = get_frame(cap, scaling_factor)

        # Convert the HSV colorspace
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

        # Define 'blue' range in HSV colorspace
        lower = np.array([60,100,100])
        upper = np.array([180,255,255])

        # Threshold the HSV image to get only blue color
        mask = cv2.inRange(hsv, lower, upper)

        # Bitwise-AND mask and original image
        res = cv2.bitwise_and(frame, frame, mask=mask)
        res = cv2.medianBlur(res, 5)

        cv2.imshow('Original image', frame)
        cv2.imshow('Color Detector', res)
```

```
# Check if the user pressed ESC key
c = cv2.waitKey(5)
if c == 27:
    break

cv2.destroyAllWindows()
```



Рис. 4.6. Алгоритм виділення певного колірного простору

БІБЛІОГРАФІЧНИЙ СПИСОК

Гридин, В. Н. Адаптивные системы технического зрения / В. Н. Гридин, В. С. Титов, М. Н. Труфанов. – СПб. : Наука, 2009. – 442 с.

Лутц, М. Программирование на Python. Т. 1 / М. Лутц. – СПб. : Символ-Плюс, 2011. – 992 с.

Магда, Ю. С. Raspberry Pi. Руководство по настройке и применению / Ю. С. Магда. – СПб. : ДМК Пресс, 2014. – 188 с.

Методы компьютерной обработки изображений / под ред. В. А. Сойфера. – М. : Физматлит, 2003. – 784 с.

Обработка и анализ изображений в задачах машинного зрения : курс лекций и практ. занятий / Ю. В. Визильтер и др. – М. : Физматкнига, 2010. – 672 с.

Официальный сайт документации для работы с компьютером Raspberry Pi [Электронный ресурс]. – Режим доступа: <https://www.raspberrypi.org>.

Официальный сайт языка Python [Электронный ресурс]. – Режим доступа: <http://python.org/>.

Официальный сайт разработчиков библиотеки OpenCV [Электронный ресурс]. – Режим доступа: <http://opencv.org>.

Прохоренок, Н. А. Python. Самое необходимое / Н. А. Прохоренок. – СПб. : БХВ-Петербург, 2011. – 416 с.

Прэтт, У. Цифровая обработка изображений / У. Прэтт. – М. : Мир, 1982. – 791 с.

Саммерфилд, М. Программирование на Python 3. Подробное руководство. – СПб. : Символ-Плюс, 2009. – 608 с.

Сергиенко, А. Б. Цифровая обработка сигналов / А. Б. Сергиенко. – СПб. : Питер, 2005. – 587 с.

Техническое зрение роботов / В. И. Мошкин и др. – М. : Машиностроение, 1990. – 272 с.

Хахаев, И. А. Практикум по алгоритмизации и программированию на Python / И. А. Хахаев. – М. : Альт Линукс, 2010. – 126 с.

Хуанг, Т. С. Быстрые алгоритмы в цифровой обработке изображений / Т. С. Хуанг, Дж.-О. Эклунд, Г. Дж. Нуссбаумер / под ред. Т. С. Хуанга. – М. : Радио и связь, 1984. – 224 с.

Федоров, Д. Ю. Основы программирования на примере языка Python : учеб. пособие / Д. Ю. Федоров. – СПб. : СПбГЭУ, 2016. – 176 с.

Форсайт, Дэвид А. Компьютерное зрение. Современный подход / Дэвид А. Форсайт, Жан Понс. – М. : Вильямс, 2004. – 928 с.

ЗМІСТ

ВСТУП.....	3
1. ОСНОВНІ РЕСУРСИ ОБРОБЛЕННЯ ЗОБРАЖЕНЬ І ВІДЕО В СИСТЕМАХ ТЕХНІЧНОГО ЗОРУ	4
1.1. Загальна інформація про бібліотеку OpenCV	4
1.2. Підключення бібліотеки OpenCV до Python для ОС Windows	6
1.3. Підключення OpenCV до одноплатового комп'ютера Raspberry Pi	8
2. ВИКОРИСТАННЯ БІБЛІОТЕКИ PILLOW ПРИ ОБРОБЛЕННІ ЗОБРАЖЕНЬ	11
2.1. Основні модулі пакета Pillow	11
2.2. Особливості використання функцій і методів модулів пакета Pillow	20
2.2.1. Функції для відкриття, копіювання і зберігання файлів зображень	20
2.2.2. Створення нового зображення	24
2.2.3. Отримання інформації про зображення	25
2.2.4. Перетворення зображень	25
3. АЛГОРИТМИ OPENCV ДЛЯ ОБРОБЛЕННЯ ЗОБРАЖЕНЬ У ПРОГРАМІ PYTHON	35
3.1. Уведення й візуалізація зображень і відеоданих	35
3.2. Перетворення зображень за допомогою функцій OpenCV	39
3.3. Алгоритми фільтрації зображень у бібліотеці OpenCV	47
3.4. Гістограми розподілу яскравості зображень	56
3.5. Бінаризація зображень з відсіканням за порогом яскравості	63
3.6. Афінні й проєктивні перетворення зображень	65
3.7. Детектування кутів на зображеннях у системах технічного зору	71
4. ВИКОРИСТАННЯ АЛГОРИТМІВ OPENCV ДЛЯ ОБРОБЛЕННЯ ВІДЕОДАНИХ	74
4.1. Оброблення відео з web-камери в реальному часі	74
4.2. Оброблення відео в інтерактивному режимі	77
4.3. Оцінювання руху об'єкта по відеопотоку	82
4.4. Використання колірних просторів для оцінювання руху	88
БІБЛІОГРАФІЧНИЙ СПИСОК	90

Навчальне видання

**Краснов Леонід Олександрович
Дергачов Костянтин Юрійович
Багінський Сергій Вікторович
Пявка Євгеній Валентинович**

**ОСНОВИ ПОБУДОВИ СУЧАСНИХ МОБІЛЬНИХ СИСТЕМ
ТЕХНІЧНОГО ЗОРУ**

Частина 2

ОБРОБЛЕННЯ ЗОБРАЖЕНЬ І ВІДЕОДАНИХ

Редактор Т. О. Іващенко

Зв. план, 2018

Підписано до друку 06.02.2018

Формат 60x84 1/16. Папір офс. № 2. Офс. друк

Ум. друк. арк. 5,1. Обл.-вид. арк. 5,75. Наклад 100 пр.

Замовлення 17. Ціна вільна

Видавець і виготовлювач

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

61070, Харків-70, вул. Чкалова, 17

<http://www.khai.edu>

Видавничий центр «ХАІ»

61070, Харків-70, вул. Чкалова, 17

izdat@khai.edu

Свідоцтво про внесення суб'єкта видавничої справи
до Державного реєстру видавців, виготовлювачів і розповсюджувачів
видавничої продукції сер. ДК № 391 від 30.02.2001