

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
ДЕРЖАВНИЙ ЗАКЛАД  
„ЛУГАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ  
ІМЕНІ ТАРАСА ШЕВЧЕНКА”

Навчально-науковий інститут математики та інформаційних технологій  
Кафедра математики систем та інформатики

**Попов Денис Дмитрович**

**ТЕХНОЛОГІЯ ІДЕНТИФІКАЦІЇ АВТОМОБІЛЬНИХ НОМЕРІВ**

Магістерська робота  
за спеціальністю 122 Комп’ютерні науки

Особистий підпис – \_\_\_\_\_

Науковий керівник – \_\_\_\_\_ д-р філософії Владислав КОЗУБ

В.о.зав. кафедри – \_\_\_\_\_ д-р техн.наук, професор Юрій КОЗУБ

Полтава - 2025

## АНОТАЦІЯ

*Попов Д.Д.* Технологія ідентифікації автомобільних номерів. Кваліфікаційна робота другого (магістерського) рівня освіти. Луганський національний університет імені Тараса Шевченка. 2025.

В першому розділі наведено аналіз засобів та платформ, що використовуються для створення ідентифікаторів графічних зображень.

Другий розділ роботи приквятий аналізу алгоритмів обробки графічних сигналів. Попередній аналіз зображення полягає у побудові схематичного образу за допомогою виділення відрізків та кіл на зображенні. На наступному етапі проводиться аналіз перетворень схематичних зображень до нормалізованого вигляду для подальшого розпізнавання символів.

В третьому розділі наведено структуру та фрагменти коду Windows додатку.

## **ANNOTATION**

*Popov D.D.* Vehicle number identification technology. Qualification work of the second (master's) level of education. Luhansk Taras Shevchenko National University. 2025.

The first section provides an analysis of the tools and platforms used to create an identifier for graphic images.

The second section of the work is devoted to the analysis of algorithms for processing graphic signals. The preliminary analysis of the image consists in constructing a schematic image by selecting segments and circles in the image. The next stage analyzes the transformations of schematic images to a normalized form for further character recognition.

The third section provides the structure and code fragments of the Windows application.

## ЗМІСТ

ВСТУП .....	5
1 ПРИЗНАЧЕННЯ ТА МЕТА СТВОРЕННЯ СИСТЕМИ .....	6
1.1 Мета роботи .....	6
1.2. Постановка задачі .....	7
1.3 Техніко-математичний опис задачі .....	7
1.3.2 Бібліотека OpenALPR.....	8
1.3.3 Бібліотека OpenCV.....	9
1.3.4 Комплект розробника Java Development Kit 8 .....	10
1.4 Опис мови програмування .....	11
2 АЛГОРИТМ ІДЕНТИФІКАЦІЇ ГРАФІЧНИХ ЗОБРАЖЕНЬ.....	12
2.1 Алгоритми попереднього пошуку.....	12
2.2 Алгоритми нормалізації .....	21
2.2.1 Афінні перетворення .....	21
2.2.2 Гомографія.....	29
2.2.3 Методи контрастування .....	31
2.3. Алгоритми розпізнавання символів .....	35
2.3.1. Алгоритм кластеризації k-means .....	35
2.3.2 Метод опорних векторів.....	46
3 ОПИС WINDOWS - ДОДАТКУ .....	49
3.1 Опис структурної схеми додатку .....	49
3.2 Пояснення до програмного коду .....	51
3.3 Контрольний приклад.....	61
ВИСНОВКИ.....	64
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	65

## **ВСТУП**

Застосування систем штучного інтелекту знаходить все більшого поширення. Одним з перспективних напрямків використання таких систем є використання в сфері безпеки.

Метою роботи є дослідження методів розпізнавання графічних об'єктів та розробка Windows-додатку для розпізнавання автомобільних номерних знаків з зображення.

Для досягнення поставленої мети необхідно виконати наступні завдання.

Провести аналіз існуючих методів розпізнавання графічних об'єктів.

Розробити застосунок розпізнавання автономерів із урахуванням країни походження авто.

Виконати завдання, використовуючи об'єктно-орієнтовану мову програмування Java, бібліотеку аналізу зображення та відео для ідентифікування автомобільних номерних знаків OpenALPR та бібліотеку алгоритмів комп'ютерного зору, обробки зображень і числових алгоритмів загального призначення OpenCV.

## **1 ПРИЗНАЧЕННЯ ТА МЕТА СТВОРЕННЯ СИСТЕМИ**

### **1.1 Мета роботи**

В наш час щорічно збільшується кількість транспортних засобів на дорогах. З цим збільшується відповідальність щодо безпечного руху, а також безпеки оточуючих. Інформаційні технології не стоять на місці і їх також впроваджують в автомобільний світ. Технологія розпізнавання автомобільних номерів може бути дуже корисною для працівників поліції, а саме для моніторингу дорожньо-транспортної обстановки, автоматизувати безпечний в'їзд транспорту на територію підприємства, контролювати кількість транспортних засобів на території об'єкту.

Метою випускної роботи є розробка алгоритму та створення Windows-додатку з розпізнавання автомобільних номерних знаків з зображення алгоритмами бібліотек OpenALPR та OpenCV. Розроблюваний Windows-додаток повинен виконувати такі функції:

- а) завантажувати зображення з комп'ютеру;
- б) обирати необхідну країну чи регіон, відповідно до номерного знаку автомобіля;
- в) виконувати алгоритм аналізу та розпізнавання номерного знаку автомобіля;
- г) виводити інформацію про номерний знак автомобіля у головній формі програми у текстовому форматі;
- д) видаляти з форми зображення та текстової інформації про номерний знак автомобіля.

Ціллю дипломної роботи є розробка Windows-додатку з розпізнавання

зображень автомобільних номерів різних країн та з'ясування обмежень алгоритму.

Головна задача Windows-додатку, що розробляється - розпізнавання з зображення автомобільного номерного знаку та його вивід у текстовому форматі.

## **1.2. Постановка задачі**

У Windows-додатку передбачено завантаження зображення з комп'ютеру та розпізнавання автомобільних номерних знаків для таких країн та регіонів як США, Австралія та країни Євросоюзу,. Передбачена функція виводу номерного знаку у текстовому форматі. Окрім виводу найбільш ймовірного номерного знаку, також виводиться інформація про розмір завантаженого зображення, час виконання алгоритму, кількість знайдених номерних знаків та перелік варіантів номерних знаків з точністю, які ймовірно можуть бути реальним номерним знаком, що відповідає зображенню.

## **1.3 Техніко-математичний опис задачі**

Windows-додаток повинен бути реалізований на базі технологій:

- а) інтегроване середовище розробки IntelliJ IDEA 15;
- б) бібліотека для аналізу зображення та відео для ідентифікування автомобільних номерних знаків OpenALPR;
- в) бібліотека алгоритмів комп'ютерного зору, обробки зображень і числових алгоритмів загального призначення OpenCV;
- г) комплект розробника застосунків на мові Java, який включає до себе компілятор Java (javac), стандартні бібліотеки класів Java, приклади, документацію, різноманітні утиліти і виконавчу систему Java - Java Development Kit 8.

### **1.3.1 Інтегроване середовище розробки IntelliJ IDEA**

IntelliJ IDEA — комерційне інтегроване середовище розробки для різних мов програмування (Java, Python, Scala, PHP та ін.) від компанії JetBrains. При написанні дипломної роботи була використана повнофункціональна комерційна версія «Ultimate Edition».

Комерційна версія «Ultimate Edition» підтримує інструменти для проведення тестування TestNG і JUnit, системи контролю версій CVS, Subversion, Mercurial і Git, засоби складання Maven і Ant, мови програмування Java, Scala, Clojure, Groovy і Dart. Підтримується розробка застосунків для мобільної платформи Android. До складу входить модуль візуального проектування GUI-інтерфейсу Swing UI Designer, XML-редактор, редактор регулярних виразів, система перевірки коректності коду, система контролю за виконанням завдань і доповнення для імпорту та експорту проектів з Eclipse. Доступні засоби інтеграції з системами відстеження помилок JIRA, Trac, Redmine, Pivotal Tracker, GitHub, YouTrack, Lighthouse.

Також, на відміну від безкоштовної версії, підтримує додаткові мови програмування (наприклад, PHP, Ruby, Python, JavaScript, CoffeeScript, HTML, CSS, SQL), підтримує технології Java EE, UML-діаграм, підрахунок покриття коду, можливість роботи з фреймворками (Rails, Grails, Google Web Toolkit, Spring, Play Framework і Hibernate), засобами інтеграції з Perforce, Microsoft Team Foundation Server і Rational ClearCase.

### **1.3.2 Бібліотека OpenALPR**

OpenALPR є бібліотекою для аналізу зображення та відео для ідентифікування автомобільних номерних знаків, яка написана на мові програмування C ++ з прив'язками в C #, Java, Node.js і Python. Бібліотека аналізує зображення та відео потоки для ідентифікації номерних знаків. Вихідні дані - текстове представлення будь-яких символів номерного знака.



Програмне забезпечення може бути використане в самих різних формах. Наприклад, з OpenALPR ви можете:

- а) визначити номерні знаки з відео камери. Результати доступні для перегляду, пошуку і може видавати попередження. Сховище даних може бути в хмарі або повністю зберігається у вашій локальній мережі;
- б) визначити номерні знаки з потоків камери і відправити результати на вашій власний додаток;
- в) запустити відео файл і зберегти результати номерного знака в базі даних CSV і SQLite;
- г) аналізувати фотографії з командного рядка;
- д) інтегрувати визначений номерний знак у додаток безпосередньо у коді (C / C ++, C #, VB.NET, Java, Python, Node.js);
- е) запускати OpenALPR як веб-сервіс. JPG зображення передається по HTTP POST, веб-служба OpenALPR відповідає за інформацію номерного знака на зображенні.

### **1.3.3 Бібліотека OpenCV**

OpenCV - бібліотека функцій та алгоритмів комп'ютерного зору, обробки зображень і чисельних алгоритмів загального призначення з відкритим кодом. Бібліотека надає засоби для обробки і аналізу вмісту зображень, у тому числі розпізнавання об'єктів на фотографіях (наприклад, осіб і фігур людей, тексту тощо), відстежування руху об'єктів, перетворення зображень, застосування методів машинного навчання і виявлення загальних елементів на різних зображеннях.

Бібліотека розроблена Intel і нині підтримується компаніями Willow Garage та Itseez. Код ядра бібліотеки написаний мовою C++ і поширюється під ліцензією BSD. Окрім API на C++ підготовлені біндинги для різних мов

програмування, таких як Python, Java, Ruby, Matlab, Lua та інших. Бібліотеку можна вільно використовувати в академічних та комерційних цілях.

Бібліотека містить понад 2500 оптимізованих алгоритмів, серед яких повний набір як класичних так і практичних алгоритмів машинного навчання і комп'ютерного зору. Алгоритми OpenCV застосовують у таких сферах:

- а) аналіз та обробка зображень;
- б) системи з розпізнавання обличчя;
- в) ідентифікації об'єктів;
- г) розпізнавання жестів на відео;
- д) відстежування переміщення камери;
- е) побудова 3D моделей об'єктів;
- ж) створення 3D хмар точок зі стерео камер;
- о) склеювання зображень між собою, для створення зображень всієї сцени з високою роздільною здатністю; п) система взаємодії людини з комп'ютером; р) пошуку схожих зображень із бази даних; с) усування ефекту червоних очей при фотозйомці зі спалахом; т) стеження за рухом очей; у) аналіз руху; ф) ідентифікація об'єктів; х) сегментація зображення; ц) трекінг відео; ч) розпізнавання елементів сцени і додавання маркерів для створення доповненої реальності.

#### **1.3.4 Комплект розробника Java Development Kit 8**

Java Development Kit, скорочено JDK - безкоштовний розповсюджуваний Oracle (раніше Sun) комплект розробника застосунків на мові Java, який включає до себе компілятор Java (javac), стандартні бібліотеки класів Java, приклади, документацію, різноманітні утиліти і виконавчу систему Java (JRE). В склад JDK не входить інтегроване середовище розробки на Java (IDE), тому розробник, що використовує тільки JDK, повинен використовувати текстовий редактор і компілювати та виконувати свої програми через утиліти командного рядка. Усі середовища розробки на Java,

такі, як Eclipse, Netbeans, IntelliJ IDEA, Borland JBuilder, спираються на сервіси JDK, що надаються, і викликають для компіляції Java-програм компілятор з комплекту JDK. Тому ці середовища розробки або включають в комплект постачання одну з версій JDK або вимагають для своєї роботи попереднє встановлення JDK на машині розробника.

Повні вихідні тексти JDK, включаючи вихідні тексти самого Java-компілятора доступні вільно.

#### **1.4 Опис мови програмування**

Java - об'єктно-орієнтована мова програмування, випущена компанією Sun Microsystems у 1995 році як основний компонент платформи Java. Зараз мовою займається компанія Oracle, яка придбала Sun Microsystems у 2009 році. Синтаксис мови багато в чому схожий на C та C++. У офіційній реалізації Java-програми компілюються у байт-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи.

Oracle надає компілятор Java та віртуальну машину Java, які задовольняють специфікації Java Community Process, під ліцензією GNU General Public License.

## 2 АЛГОРИТМ ІДЕНТИФІКАЦІЇ ГРАФІЧНИХ ЗОБРАЖЕНЬ

Структура алгоритму:

- а) попередній пошук номеру - виявлення області в якій міститься номер;
- б) нормалізація номера - визначення точних меж номера, нормалізація контрасту;
- в) розпізнавання тексту - читання всього, що знайшлося в нормалізованому зображенні.

Це базова структура. Звичайно, в ситуації, коли номер лінійно розташований і добре освітлений, а в розпорядженні відмінний алгоритм розпізнавання тексту, перші два пункти відпадуть. У деяких алгоритмах можуть об'єднуватися пошук номера і його нормалізація.

### 2.1 Алгоритми попереднього пошуку

Перетворення Хафа - це метод знаходження ліній, кіл та інших простих аналітичних кривих на зображенні (Хаф розробив дане перетворення для застосування в фізичних експериментах. Впровадження перетворення для вирішення завдань комп'ютерного зору здійснили Duda і Hart). Спочатку перетворення Хафа застосовувалося для знаходження ліній, тому що є відносно швидким способом знаходження прямих ліній на бінарному зображенні. Надалі дане перетворення узагальнили для більш складних завдань, ніж просто пошук ліній. Теоретичною підосновою перетворення Хафа для ліній є те, що будь-яка точка на бінарному зображенні, можливо, є частиною деякого безлічі ліній. Якщо описати кожен ліній по її нахилу  $a$  і зрушенню  $b$ , то точка на оригінальному документі перетворюється на безліч точок на площині  $(a, b)$ , які відповідають усім лініям, що проходять через цю точку. Якщо перетворити кожен ненульовий піксель у вхідному зображенні в

такий набір точок у вихідному зображенні і підсумувати всі подібні внесення, тоді лінія, що з'явилася на вхідному зображенні (тобто на площині  $(x, y)$ ) буде відповідати локальному максимуму в вихідному зображенні (тобто на площині  $(a, b)$ ). Оскільки підсумовується внесок від кожної точки, площину  $(a, b)$  прийнято називати накопичувальною площиною (надалі просто накопичувач). Може виявитися, що площину виду нахил-зрушення не кращий спосіб представлення всіх ліній, що проходять через точку (через значно різної щільності ліній в залежності від нахилу і від того, що інтервал можливих нахилів лежить в діапазоні від -да до +да). З цієї причини, в чисельних розрахунках використовується інша параметризація перетворення зображення. Бажана параметризація представляє кожну лінію як точку в полярних координатах  $(\rho, \theta)$ , при чому ця лінія проходить через зазначену точку, але лінія повинна бути перпендикулярна до радіус-вектору від початку координат до цієї точки на лінії. Рівняння для такої прямої:

$$\rho = x \sin \theta + y \cos \theta \quad (2.1)$$

Рис. 2.1 – Знаходження ліній за допомогою перетворення Хафа.

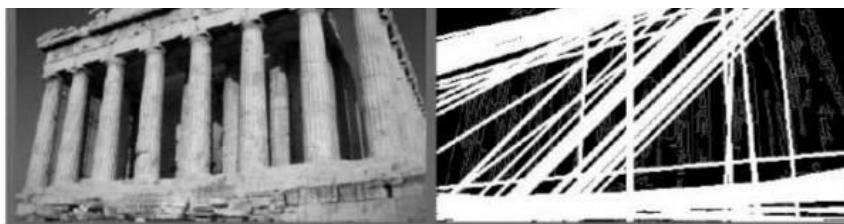
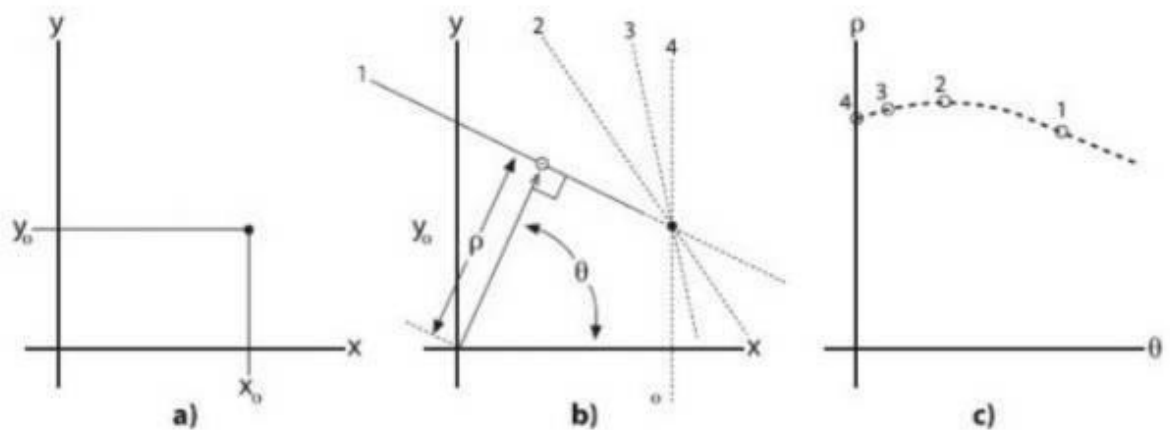


Рис. 2.2 - Утворення кривої характеристичної форми

Алгоритм перетворення Хафа не є явним для користувача. Замість цього він просто повертає локальний максимум на площині  $(\rho, \theta)$ . Однак, необхідно розуміти процес перетворення, щоб усвідомлювати призначення вхідних аргументів функції, що виконує перетворення Хафа для ліній. OpenCV підтримує два види перетворень Хафа для ліній: звичайне перетворення Хафа (SHT) і прогресивне (покращене) імовірнісне перетворення Хафа (PPHT). Раніше вже було розглянуто алгоритм SHT. PPHT є його різновидом і також обчислює протяжність кожної лінії на додаток до їх нахилу. Алгоритм названий "імовірнісним", тому що замість додавання кожної можливої точки на накопичувальну площину, він додає тільки частина з них. Ідея полягає в тому, що якщо існує максимум на площині, то, в будь-якому випадку, хоча б часткове попадання в цей максимум буде достатньою умовою, щоб виявити лінію; як результат - істотне зниження часу виконання обчислень. Для обох випадків в OpenCV існує одна функція, яка, в залежності від вхідних параметрів, приймає рішення про виконання того чи іншого методу.

```
CvSeq* cvHoughLines2( CvArr* image ,void*
line_storage ,int method
, double rho ,double theta ,int threshold
, double param1 = 0 ,double param2 = 0);
```

Перший аргумент - це вихідне зображення. Воно повинно бути 8-бітовим, але при цьому буде трактуватися як бінарне (тобто всі ненульові значення будуть сприйматися як рівні одиниці). Другий аргумент - покажчик на місце, де буде зберігатися результат, який може бути або сховищем в пам'яті (CvMemoryStorage), або матрицею розміру  $N \times 1$  (при цьому кількість рядків  $N$  буде також ще обмежувати максимальну кількість повертаються ліній). Наступний аргумент `method` може бути `CV_HOUGH_STANDARD`, `CV_HOUGH_PROBABLISTIC`, `CV_HOUGH_MULTI_SCALE` для SHT, PPHT і багатопараметричного варіанти SHT (MSHT) відповідно. Наступні два

аргументи,  $\rho$  та  $\theta$ , встановлюють бажане дозвіл для ліній (тобто дозвіл накопичувальної площини). Параметр  $\rho$  обчислюється в пікселях, а параметр  $\theta$  в радіанах, тому накопичувальну площину можна розглядати як двомірну гистограму з осередками розмірністю  $\rho$  пікселів на  $\theta$  радіан. Значення параметра  $\text{threshold}$  визначає величину, при досягненні якої повідомляється про знаходження лінії. Цей останній параметр на практиці кілька мудрований, причому він не нормалізується, тому очікується, що сам розробник буде його масштабувати з урахуванням зростання розмірності вхідного зображення для алгоритму SHT. Цей параметр, насправді, визначає кількість точок, яке має містити лінія, щоб бути доданою в повертається список ліній.

Спочатку виконаний прохід за допомогою детектора меж Canny ( $\text{param1} = 50$ ,  $\text{param2} = 150$ ), результат показаний в відтінках сірого. Потім вироблено прогресивне розподіл усіх перетворень Хафа ( $\text{param1} = 50$ ,  $\text{param2} = 10$ ), результат показаний білим. Перетворення Хафа в основному правильно виявляє чіткі лінії.

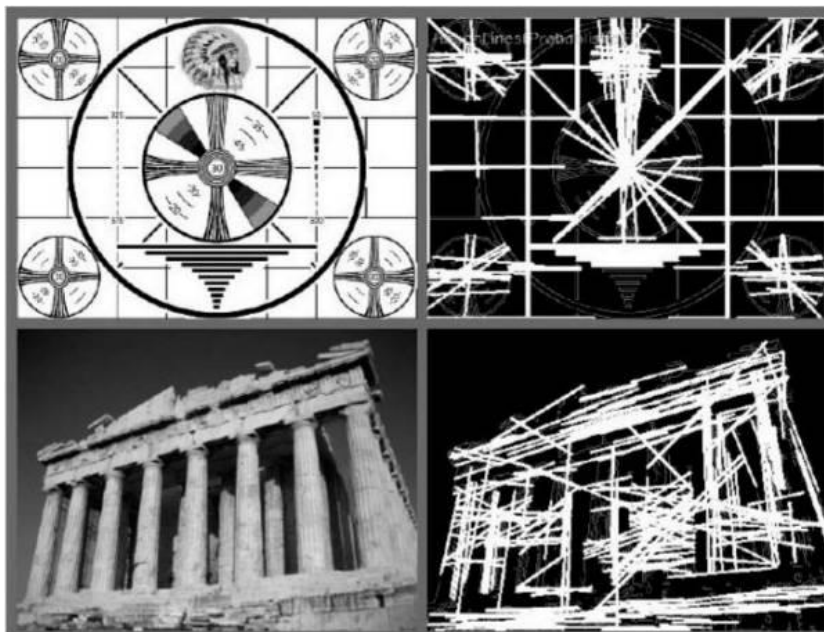


Рис. 2.3 - Результат обробки зображення після проходу за допомогою детектора меж Canny та прогресивного розподілу усіх перетворень Хафа

Аргументи `param1` і `param2` алгоритм SHT не використовують. Для алгоритму PPHT `param1` задає мінімальну довжину для повертаємого сегмента лінії, а аргумент `param2` задає відстань між колінеарними сегментами, необхідне для того, щоб алгоритм не склеїв їх разом в один сегмент більшої довжини. Для багатопараметричного варіанта SHT ці два параметри застосовуються для вказівки найвищого дозволу, до якого параметри повертаємих ліній повинні бути обчислені.

Багатопараметричний SHT спочатку обчислює положення ліній з урахуванням дозволів `rho` і `theta`, а потім починає уточнення результатів до ступеня параметрів `param1` і `param2` відповідно (тобто кінцеве дозвіл `rho` - це `rho` поділене на `param1`, а кінцевий дозвіл для `theta` дорівнює `theta` діленому на `param2`).

Значення, що повертається функції залежить від вхідних параметрів. Якщо параметр `line storage` матриця, тоді повертається значення буде NULL. У цьому випадку, матриця повинна бути типу `CV_32FC2` для SHT або багатопараметричного SHT і `CV_32SC4` для PPHT. У перших двох випадках величини `p` і `0` для кожної лінії будуть поміщені в двох каналах масиву. У разі PPHT, чотири канали будуть містити значення `x` і `y` для початкової та кінцевої точок повертаємого сегмента. І для всіх цих випадків, кількість рядків в масиві буде оновлено функцією `cvHoughLines2()` до кількості повертаються ліній.

Якщо параметр `line_storage` містить покажчик на сховище в пам'яті, тоді повертається значенням буде покажчик на структуру послідовності `CvSeq`. В цьому випадку, можна отримати кожну лінію або сегмент лінії з послідовності за допомогою подібної команди:

```
float* line = (float*) cvGetSeqElem( lines, i );
```

де `lines` - це значення, що повертається функції `cvHoughLines2()`, а `i` - індекс запитуваної лінії. У цьому випадку, `line` буде покажчиком на дані



запитуваної лінії, при чому `line [0]` і `line [1]` будуть дійсними числами, що відповідають значенням  $p$  і  $0$  (для SHT і MSHT), або покажчиком на структуру з парних значень `CvPoint` для початкової та кінцевої точок сегмента (для алгоритму PRHT).

Перетворення Хафа для кіл працює майже аналогічно тільки що описаного перетворенню Хафа для ліній. Якщо виконати точно аналогічні дії - накопичувальна площину буде заміщена площиною обсягу з трьома вимірами: одне для  $x$ , одне для  $y$  і останнім для радіуса кола  $r$ . Це призводить до значно більших витрат пам'яті і значно меншій швидкості виконання. Реалізація перетворення Хафа для окружності в OpenCV йде від цієї проблеми, застосовуючи кілька хитрий метод, званий градієнтним методом Хафа.

Градієнтний метод Хафа працює наступним чином. Спочатку зображення проходить фазу пошуку країв (використання функції `cvCanny()`). Потім для кожної ненульовий точки країв зображення шукається локальний градієнт (обчислюється шляхом розрахунку похідних Собеля першого і другого порядку для  $x$  і  $y$  за допомогою функції `cvSobel()`). Використовуючи цей градієнт, кожна точка лінії позначається як нахил - від встановленого мінімуму до зазначеного максимального відстані - ітераційно змінюючи накопичувач. У той же час, запам'ятовується розташування кожної ненульовий точки на зображенні країв. Центри-кандидати потім вибираються з тих точок (двовірного) накопичувача, величини яких вище заданого порогу і, одночасно, більше всіх їх безпосередніх сусідів. Ці центри-кандидати сортуються в порядку убутання їх величини в накопичувачі, так, що центри з найбільшою кількістю пікселів вибираються першими. Далі, для кожного центру, аналізуються всі ненульові пікселі. Ці пікселі упорядковано відповідно до їх відстанню від центру. Обробляючи від найменших відстаней до найбільших радіусів, вибирається єдиний радіус, який найкраще підходить ненульовим пікселям. Центр зберігається, якщо він має достатню кількість ненульових пікселів на краях на зображенні і якщо відстань від будь-якого раніше обраного центру задовольняє заданому значенню. Ця реалізація дозволяє

алгоритму виконуватися набагато швидше і, що ще більш важливо, дозволяє обійти проблему розростання тривимірного накопичувача, що призводить до значно великим зашумлення і нестабільно відбиваному результату. З іншого боку, цей алгоритм має кілька недоліків.

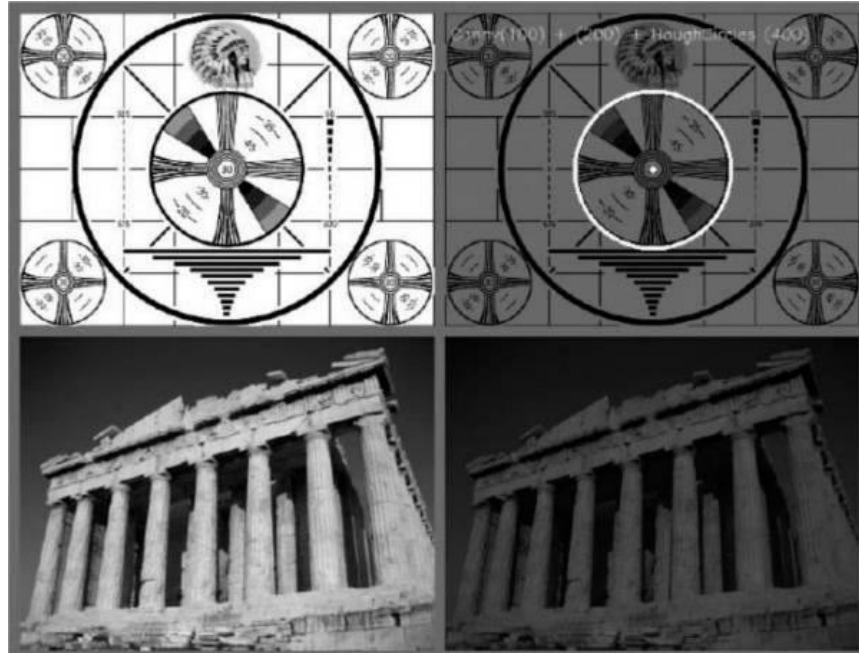


Рис. 2.4 - Перетворення Хафа для кола знаходить кілька кіл на тестовому шаблоні і не знаходить нічого на фотографії

По-перше, застосування похідних Собеля для обчислення локального градієнта - і припущення, що він може бути розглянутий як локальний тангенс - чисельністю не стабільне рішення. Він може давати вірний результат "більшу частину часу", при цьому на виході не виключено присутність деякої кількості шуму.

По-друге, кожен ненульовий піксель на гранях на зображенні розглядається як центр-кандидат, тому для малого значення порога накопичувача час виконання алгоритму різко зростає.

По-третє, оскільки для кожного центру вибирається тільки одна окружність, то в разі існування концентричних кіл, використовується тільки одна з них.

І нарешті, оскільки центри розглядаються в порядку зростання значень з накопичувача і до того ж нові центри не додаються, якщо вони занадто близькі до вже доданим центрам, в разі концентричних кіл (або близько концентричних), перевага може надаватися тим, у яких найбільший розмір. Беручи до уваги все вище сказане, розглянемо функцію OpenCV, яка реалізує алгоритм Хафа для кіл:

```
CvSeq* cvHoughCircles( CvArr* image ,void* circle_storage ,int method ,double
dp ,double min_dist ,double param1 = 100 ,double param2 = 300 ,int min_radius = 0 ,int
max_radius = 0 );
```

Функція перетворення Хафа для кіл `cvHoughCircles()` має параметри, схожі на параметри в функції для ліній. Початкове зображення повинно бути 8-бітовим. Значної відмінності між `cvHoughCircles()` і `cvHoughLines2()` в тому, що остання функція запитує бінарне зображення. Функція `cvHoughCircles()` внутрішньо (автоматично) сама викликає функцію `cvSobel()` (внутрішньо викликається функція `cvSobel()`, замість `cvCanny()`). Це пов'язано з тим, що `cvHoughCircles()` необхідно визначити напрямки градієнта для кожного пікселя, а це складно виконати для бінарної карти країв), тому цілком вистачає і зображення у відтінках сірого.

Аргумент `circle storage` може бути або масивом, або сховищем в пам'яті - залежить від переваг розробника. Якщо масив, то він повинен мати одну колонку типу `CV_32FC3`; три канали використовуються для зберігання значень положення і радіуса кола. Якщо сховище в пам'яті, то окружності будуть перетворені в послідовності OpenCV, а функція `cvHoughCircles()` поверне покажчик на цю послідовність. (При вказівці в `circle_storage` покажчика на масив, повертається значенням функції буде `NULL`). Аргумент `method` завжди має дорівнювати `CV_HOUGH_GRADIENT`.

Аргумент `dp` є дозволом накопичувача. Цей аргумент дозволяє створювати накопичувач більш низьких дозволів, ніж вихідне зображення. Якщо аргумент `dp` дорівнює 1, тоді дозвіл буде таким же, як і у вихідного

зображення; якщо аргумент `dp` більше 1, тоді дозвіл буде в `dp` разів менше (у разі `dp = 2`, буде половинним). Значення `dp` не може бути менше 1.

Аргумент `min_dist` задає мінімальне відстань між двома колами, щоб алгоритм розглядав їх як дві різні кола.

Для застосовуваного методу `CV_HOUGH_GRADIENT`, такі два параметри, `param1` і `param2`, є порогами для країв (Canny) і накопичувача відповідно. Внутрішній виклик функції `cvCanny()` встановлює в якості верхнього порогу значення рівне `param1`, а нижній поріг встановлює рівним половині величиною `param1`. Параметр `param2` застосовується до накопичувача в точно такому ж сенсі, як і параметр `threshold` для `cvHoughLines()`.

Останні два параметри - це мінімально і максимально можливі значення радіуса для кіл, які шукаються. Це означає, що вони є радіусами кіл, для яких накопичувач має уявлення.

Використання `cvHoughCircles()` для отримання послідовності заданих кіл на зображення у відтінках сірого:

```
#include <cv.h>    #include <highgui.h>    #include
<math.h>

int main(int argc, char** argv) {
    IplImage* image = cvLoadImage( argv[1]
    ,CV_LOAD_IMAGE_GRAYSCALE );

    CvMemStorage* storage = cvCreateMemStorage( 0 );
    cvSmooth( image, image, CV_GAUSSIAN, 5, 5 );
    CvSeq* results = cvHoughCircles( image ,storage
    ,CV_HOUGH_GRADIENT ,2
    ,image->width/10 );
    for( int i = 0; i < results->total; i++ ) {
        float* p = (float*) cvGetSeqElem( results, i );
        CvPoint pt = cvPoint( cvRound( p[0] ), cvRound( p[1] )
        cvCircle( image ,pt
```

```

, cvRound( p[2] ) , CV_RGB( 0xff, 0xff, 0xff ) );
}
cvNamedWindow( "cvHoughCircles", 1 ) ;
cvShowImage( "cvHoughCircles", image ); cvWaitKey( 0
);
}

```

Незалежно від усіх можливих застосовуваних прийомів, описати кола можна тільки за допомогою трьох ступенів свободи ( $x$ ,  $y$  і  $r$ ), в той час як для ліній достатньо тільки два ( $p$  і  $0$ ). Як результат, алгоритм пошуку кіл, в порівнянні з алгоритмом пошуку ліній, витрачає на виконання більше часу і пам'яті. Беручи до уваги цей факт, непогано було б обмежувати радіус кола наскільки це можливо в тій чи іншій ситуації. Хоча `cvHoughCircles()` вловлює центри досить точно, він іноді не здатний знайти правильний радіус. Тому в додатках, де потрібно тільки знаходження центру (або там, де для знаходження радіуса можна застосувати інші механізми), величину радіуса, що повертається `cvHoughCircles` можна ігнорувати. Перетворення Хафа було розширено довільними формами Балларда в 1981, в основному розглядаючи форми, як набори градієнтних граней

## 2.2 Алгоритми нормалізації

### 2.2.1 Афінні перетворення

В даному розділі мова піде про геометричних перетворень зображень. Ці перетворення включають в себе розтягування і поворот в різних напрямках, при цьому мова може йти як про однорідних, так і про неоднорідних змінах розміру (останнє більш відомо, як деформація).

Функції, які можуть розтягувати, стискати, деформувати і / або повертати зображення іменуються геометричними трансформаціями. Для площини є два різновиди геометричних перетворень: перетворення, яке використовує матрицю  $2 \times 3$  і іменується афінним перетворенням; і

перетворення, яке використовує матрицю  $3 \times 3$  і іменується перспективним перетворенням або гомографією. Для останнього перетворення потрібно уявити, що воно засноване на методі для обчислення шляху, по якому в тривимірному вимірюванні спрямований погляд конкретного спостерігача, за умови, що прямо поглянути на цю площину він не може.

Афінний перетворення - це будь-які перетворення, які можуть бути виражені у вигляді послідовного множення матриць і складання з деяким вектором. У OpenCV стандартної для таких перетворень є матриця розміру  $2 \times 3$ .

Результатом афінного перетворення є  $AX + B$ , що в точності еквівалентно розширенню вектора  $X$  до вектора  $X'$  і подальшого лівого множення на вектор  $T$ . Афінний перетворення можуть бути представлені таким чином: будь-який паралелограм  $ABCD$  на площині може бути відображений в будь-який інший паралелограм  $A'B'C'D'$ ; якщо площі цих паралелограмів нерівні нулю, тоді мається на увазі Афінний перетворення однозначно визначено для трьох вершин обох паралелограмів. Так само для розуміння афінного перетворення можна представити зображення у вигляді великого гумового листа, який в результаті деформації натисканням або розтягуванням (аж до перевертання паралелограма) за кути може бути перетворений в різноманітні паралелограми.

Маючи кілька зображень одного об'єкта з різних ракурсів, можна застосувати фактичне перетворення для зв'язування різних ракурсів. У цьому випадку, найчастіше для моделювання ракурсів використовуються афінні перетворення, тому що задіюється мала кількість параметрів, що полегшує процес обчислення. При цьому недоліком є те, що реальні перспективні перетворення можуть бути змодельовані лише за допомогою гомографії, так як афінні перетворення дають уявлення, які не можуть вмістити всі можливі зв'язки між ракурсами. З іншого боку, для невеличких змін положення спостерігача, результуюче спотворення буде афінним, тому в деяких випадках афінних перетворень цілком достатньо.

За рахунок афінного перетворення прямокутник може бути перетворений в паралелограм. При цьому, в результаті перетворень (обертання і/або масштабування), сторони повинні зберігати паралельність. Перспективні перетворення забезпечують більшу гнучкість; прямокутник може бути перетворений в трапецію. Так як паралелограм - це окремий випадок трапеції, то і афінні перетворення - це підмножина перспективного перетворення.

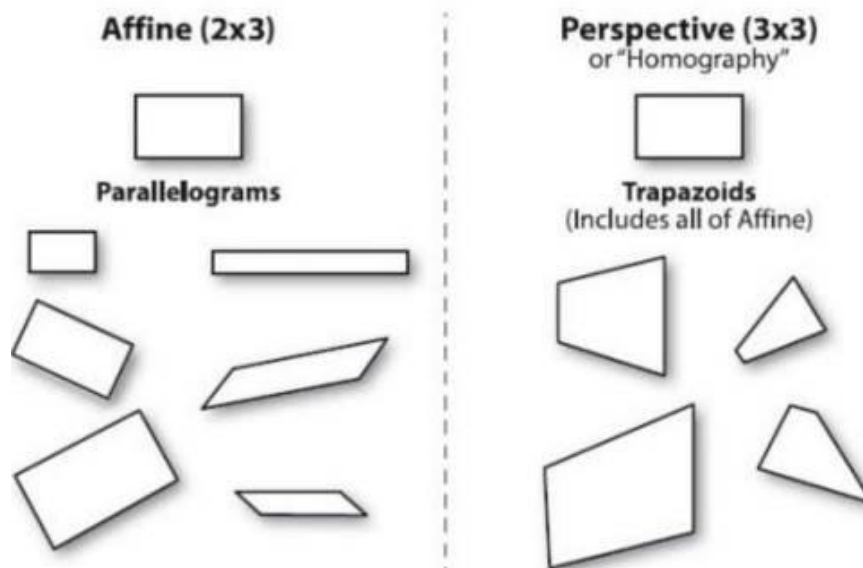


Рис. 2.5 - Афінні і перспективні перетворення

Є дві ситуації, при яких використовуються афінні перетворення. У першому випадку для конвертування валюти над зображенням (або його частиною); у другому випадку для конвертування валюти за списком точок. У першому випадку очевидно, що вхідні і вихідні формати - це зображення, і для перетворення явно потрібно, щоб пікселі були щільним поданням нижчого зображення. Це означає, що перетворювач зображення повинен виробляти інтерполяцію, так що кінцеве зображення буде виглядати натурально і згладжено. Функція OpenCV, що відповідає за афінні перетворення, іменується

```

cvWarpAffine() .
void cvWarpAffine(
    const CvArr* src ,CvArr* dst
    ,const CvMat* map_matrix
    ,int flags = CV_INTER_LINEAR | CV_WARP_FILL_OUTLIERS
    ,CvScalar fillval = cvScalarAll(0)
    );

```

Параметри `src` і `dst` вказують на масив або зображення, які можуть бути одно- або трьохканальними довільного типу. Оскільки обертання зображення призводить до розширення описує його прямокутника, результатом стане відсікання зображення. Обійти цю проблему можна або за рахунок стиснення зображення, або за рахунок копіювання першого зображення в область інтересу ROI цільового зображення більшого розміру, яке потім використовується як вихідне зображення для проведення перетворення). Параметр `map_matrix` - це матриця розміру 2x3 по якій обчислюється необхідне перетворення. Передостанній параметр `flags` визначає метод інтерполяції, рівно, як і наступні допоміжні опції (комбінування усіх можливих значень прапора проводиться за допомогою OR).

`CV_WARP_FILL_OUTLIERS` - найчастіше, в результаті перетворень, зображення `src` не вписується в зображення `dst`, тому на кінцеве зображення потрібно перенести пікселі початкового зображення, яких насправді немає. Якщо даний прапор обраний, то ці втрачені значення заповнюються значенням параметра `fillval`.

`CV_WARP_INVERSE_MAP`. Цей прапор призначений для зворотного перетворення `dst` в `src`, замість стандартного `src` в `dst`.

Важливо знати, що використання `cvWarpAffine()` призводить до значних накладних витрат. Альтернативою є використання `cvGetQuadrangleSubPix()`. Ця функція функціонально обмежена, проте, має ряд переваг. Зокрема, вона має менші накладні витрати і обробляє окремий випадок, коли вихідне



зображення 8 бітове, а кінцеве зображення речовий. Так само ця функція може обробляти і багатоканальні зображення.

```
void cvGetQuadrangleSubPix( const CvArr* src ,CvArr*
dst
, const CvMat* map_matrix
);
```

`cvGetQuadrangleSubPix()` обчислює всі точки `dst` шляхом (інтерпольованого) їх зображення з точок `src`, за рахунок застосування афінної перетворення, що в свою чергу має на увазі множення на матрицю `map_matrix` розміром  $2 \times 3$ . Перетворення розташування пікселів в `dst` до однорідних координатах для виконання множення здійснюватися автоматично.

Особливість `cvGetQuadrangleSubPix()` полягає в тому, що функція виробляє додаткове відображення. Варто звернути увагу, що відображення  $(x, y)$  в  $(x'', y'')$  має наступний ефект - навіть якщо перетворення  $M$  одиничне перетворення - точки в центрі кінцевого зображення будуть відповідати точкам, узятим з початку відліку вихідного зображення. Якщо `cvGetQuadrangleSubPix()` потрібна точка поза зображення, буде задіяна реплікація для відновлення цих значень.

OpenCV надає дві функції для обчислення матриці `map matrix`. Перша використовується, коли вже є два зображення, про які відомо, що вони співвідносяться таким собі афінних перетворень, або апроксимувати їх таким чином:

```
CvMat* cvGetAffineTransform(
const CvPoint2D32f* pts_src ,const CvPoint2D32f*
pts_dst ,CvMat* map_matrix
);
```

`src` і `dst` - це масиви, що містять три двовимірні  $(x, y)$  точки, а `map_matrix` - обчислюється з цих точок афінная матриця.

Аргументи `pts_src` і `pts_dst` функції `cvGetAffineTransform()` - це всього лише масиви з трьох точок, що визначають два паралелограма. Найпростіший

спосіб визначити Афінний перетворення - встановити покажчик pts\_src на три кути вихідного зображення - наприклад, на верхній і нижній лівий і на верхній правий. Відображення вихідного зображення в кінцеве зображення повністю визначається зазначенням в pts\_dst місця, в якому по трьох точках буде відображено кінцеве зображення. Після твори відображення цих трьох незалежних кутів, всі інші точки можуть бути перетворені відповідно.

Отримання параметрів матриці функції cvWarpAffine() здійснюватися в такий спосіб: на початку відбувається побудова двох трикомпонентних масивів точок (для кутів паралелограма), а потім перетворення їх в дійсну матрицю перетворень за рахунок виклику cvGetAffineTransform(). Потім виконуються афінніє перетворення з наступним обертанням зображення. Для вихідного зображення створюється масив точок іменований srcTri[] із заданими точками: (0, 0), (0, height -1) і (width -1, 0). На завершення вказується в яке відповідне розташування будуть відображені точки з масиву srcTri[] в масив dstTri[]. Приклад використання афінних перетворень:

```
#include <cv.h> #include <highgui.h>

int main( int argc, char** argv ) {
    CvPoint2D32f srcTri[3], dstTri[3]; CvMat* rot_mat =
cvCreateMat( 2, 3, CV_32FC1 ); CvMat* warp_mat =
cvCreateMat( 2, 3, CV_32FC1 ); IplImage *src, *dst;
    if( argc == 2 && ((src=cvLoadImage(argv[1], 1)) != 0
) ) { dst = cvCloneImage( src ) ; dst->origin = src-
>origin;
        cvZero( dst ) ; // Обчислення матриці перетворення
// srcTri[0].x = 0; //Верхній лівий кут джерела
srcTri[0].y = 0;
        srcTri[1].x = src->width - 1; //Верхній правий кут
джерела
        srcTri[1].y = 0; srcTri[2].x = 0; //Зсув нижнього
лівого кута джерела
```

```

srcTri[2].y = src->height - 1;
dstTri[0].x = src->width * 0.0; //Верхній лівий кут
приймача
dstTri[0].y = src->height * 0.33;
dstTri[1].x = src->width * 0.85; //Верхній лівий кут
приймача
dstTri[1].y = src->height * 0.25;
dstTri[2].x = src->width * 0.15; //Зсув нижнього
лівого кута приймача
dstTri[2].y = src->height * 0.7;
cvGetAffineTransform( srcTri, dstTri, warp_mat ) ;
cvWarpAffine( src, dst, warp_mat ) ; 29

cvCopy( dst, src ); // Обчислення матриці обертання //
CvPoint2D32f center = cvPoint2D32f( src->width/2
,src->height/2
);
double angle = -50.0; double scale = 0.6;
cv2DRotationMatrix( center, angle, scale, rot_mat );
// Виконання перетворення //
cvWarpAffine(src, dst, rot_mat);
// Відображення результату //
cvNamedWindow( "Affine_Transform", 1 );
cvShowImage( "Affine_Transform", dst );
cvWaitKey();
}
cvReleaseImage( &dst );
cvReleaseMat( &rot_mat );
cvReleaseMat( &warp_mat );
return 0;

```

```
}
```

Другою функцією обчислення матриці `map matrix` є `cv2DRotationMatrix()`, яка обчислює матрицю перетворень за допомогою обертання навколо заданої точки та необов'язкового масштабування. Це всього лише один з можливих варіантів афінних перетворень, проте, дуже важливий, тому що має альтернативне уявлення, яке легше розуміти і використовувати:

```
CvMat* cv2DRotationMatrix( CvPoint2D32f center
, double angle , double scale , CvMat* map_matrix
) ;
```

Перший аргумент `center` - точка обертання. Наступні два аргументи задають величину обертання і масштабний коефіцієнт. Останній аргумент - матриця `map_matrix` дійсного типу розмірністю 2x3. Ці методи отримання `map_matrix` можна комбінувати для отримання, наприклад, повернутого, масштабованого і деформованого зображення.

Для обробки щільних перетворень використовується `cvWarpAffine()`. Для розрядженого перетворення найкраще використовувати `cvTransform()`:

```
void cvTransform(
const CvArr* src , CvArr* dst
, const CvMat* transmat , const CvMat* shiftvec =
NULL
) ;
```

Аргумент `src` - це масив  $N \times 1$  с  $D_s$  каналами, де  $N$  - кількість точок для перетворення, а  $D_s$  - розмірність цих точок. Аргумент `dst` повинен бути того ж розміру, але може мати відмінне кількість каналів  $D_d$ . Матриця перетворень `transmat` - це матриця розмірністю  $D_s \times D_d$ , яка потім застосовується до кожного елементу з `src`, з подальшим розміщенням результату в `dst`. Якщо необов'язковий вектор `shiftvec` НЕ NULL, то він повинен бути масивом розмірністю  $D_s \times 1$ , елементи якого додаються до результату до його приміщення в `dst`.

У разі афінного перетворення, існує два способи використання `cvTransform()`, вибір залежить від уявлення перетворення. У першому випадку, перетворення розкладається на частини розміром  $2 \times 2$  (яка виконує обертання, масштабування і деформацію) і розміром  $2 \times 1$  (яка виконує перетворення). Вхідні дані: масив  $N \times 1$  з двома каналами, `transmat` локальне гомогенне перетворення, а `shiftvec` містить всі необхідні зрушення. У другому випадку, застосовується звичайне уявлення матриці афінної перетворення розмірністю  $2 \times 3$ . У цьому випадку, `src` - це трьохканальний масив, для якого потрібно встановити всі елементи третього каналу в 1. Вихідний масив буде двоканальним.

### 2.2.2 Гомографія

У комп'ютерному зорі плоска гомографія визначається як проєктивне відображення з однієї площини в іншу. Таким чином, відображення точок на двовимірну плоску поверхню фотоприймача камери є прикладом плоскою гомографії. Даний приклад можна показати у вигляді перемноження матриць за умови, що будуть використані однорідні координати для відображення точок  $Q$  і  $q$  на фотоприймачі. Якщо визначити

$$Q = [X \ Y \ Z \ 1]^T \quad (2.1)$$

$$q = [x \ y \ 1]^T \quad (2.2)$$

тоді дію гомографії можна виразити таким чином:

$$q = dHq \quad (2.3)$$

Введений параметр  $s$  - це довільний масштабний коефіцієнт (призначений для того, щоб явно показати, що гомографія визначається

тільки цим коефіцієнтом). Умовно цей коефіцієнт винесено з  $H$  і надалі при розгляді матеріалу дане правило буде виконуватися.

За рахунок невеликого застосування геометрії і матричної алгебри можна знайти рішення для цієї матриці перетворення. Найбільш важливим моментом є те, що  $H$  складається з двох частин: матеріальних перетворень, які по суті визначають положення об'єкта на даній площині; і проекції, яка виводить матрицю вбудованих параметрів камери.

По частині матеріальних перетворень - це сумарна дія деякого обертання  $R$  і деякого зсуву  $t$ , які відносяться до площини, на якій розглядається площину зображення. Без обмеження спільності, необхідно вибрати певну площину об'єкта так, щоб  $Z = 0$ . Це необхідно, тому що при розбитті матриці обертання на три  $3 \times 1$  шпальти (тобто), один з цих стовпців не буде потрібно. Бібліотека використовує кілька зображень одного об'єкта для обчислення перетворень і обертань для кожного уявлення, а також вбудованих параметрів камери (які однакові для всіх уявлень).

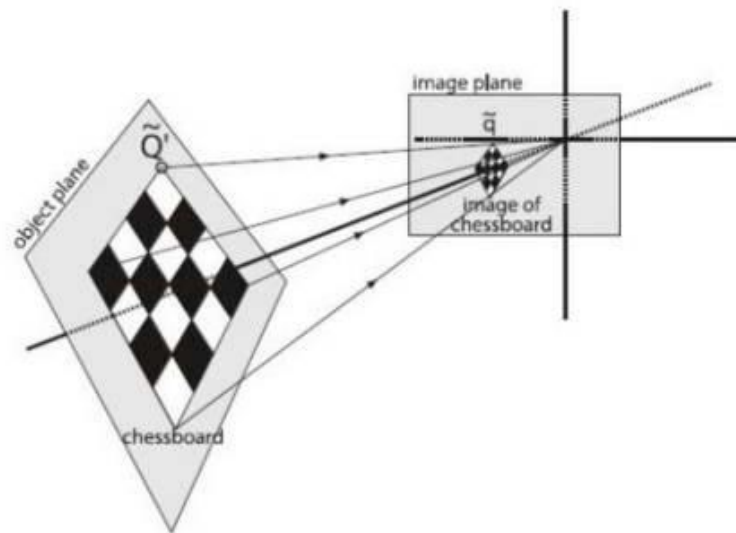


Рис. 2.6 - Подання плоского об'єкта, побудоване гомографією

Обертання описується трьома кутами, а переміщення трьома зсувами; відповідно є шість невідомих для кожного уявлення. Це нормально, тому що відомо, що плоский об'єкт (наприклад, шахівниця) дає вісім рівнянь - тобто відображення квадрата в чотирикутник можна описати чотирма  $(x, y)$  точками.

Кожен новий кадр дає вісім рівнянь за рахунок шести нових зовнішніх невідомих, так що при наявності достатньої кількості зображень є можливість обчислити будь-яку кількість власних невідомих.

Варто звернути увагу на той факт, що існує можливість обчислити  $H$ , нічого не знаючи про вбудованих параметрах камери. Насправді, обчислення безлічі гомографій з кількох вистав - це метод, який використовує OpenCV для обчислення внутрішніх параметрів камери. OpenCV надає зручну функцію `cvFindHomographyO`, яка приймає список відповідей і повертає матрицю гомографії, яка найкращим чином описує ці відповідності. Необхідно як мінімум чотири точки, щоб знайти  $H$ , однак, завжди є можливість надати набагато більшу кількість точок (за умови розгляду шахівниці розміру більшого, ніж  $3 \times 3$ ). Використання більшої кількості точок набагато вигідніше, тому що завжди є шум і інші невідповідності, вплив яких необхідно зводити до мінімуму.

```
void cvFindHomography(
    const CvMat* src_points ,const CvMat* dst_points
    ,CvMat* homography
```

Вхідні масиви `src_points` і `dst_points` можуть бути матрицями  $N \times 2$  або  $N \times 3$ . У першому випадку координати пікселів, в другому однорідні координати. Останній аргумент `homography` це матриця  $3 \times 3$ , заповнена за допомогою даної функції таким чином, щоб мінімізувати помилку зворотної проєкції. Масштабування гомографії може бути застосоване до дев'ятого параметру гомографії, але зазвичай масштабування виконується за рахунок множення всієї матриці гомографії на коефіцієнт масштабування.

### 2.2.3 Методи контрастування

Камери і детектори зображень повинні, як правило, займатися обробкою не лише контрасту, а й світла від датчиків зображення. У стандартній камері затвор і діафрагма об'єктива поперемінно подають на датчики або занадто

багато, або занадто мало світла. Часто діапазон контрастів занадто великий для датчиків; отже, існує компроміс між збором даних темних областей (наприклад, тіней), які вимагають збільшення часу експозиції, і світлих областей, які вимагають малого часу експозиції, щоб уникнути "пересвітлення".

Після того, як знімок зроблений, можливо спробувати розширити динамічний діапазон зображення. Найбільш часто використовуваним методом для цього є корекція гистограмм. Корекція гистограм - це старий математичний метод, застосовуваний в обробці зображень. На рис. 2.7 видно, що ліве зображення бліде, тому що діапазон змін значень невеликий. Це видно по гистограмі інтенсивності справа. Оскільки представлено 8-бітове зображення, то значення інтенсивності можуть змінюватися в діапазоні від 0 до 255, однак, гистограма показує, що фактичні значення інтенсивності згруповані поблизу середини доступного діапазону. Корекція гистограми - це метод розтяжки цього діапазону.

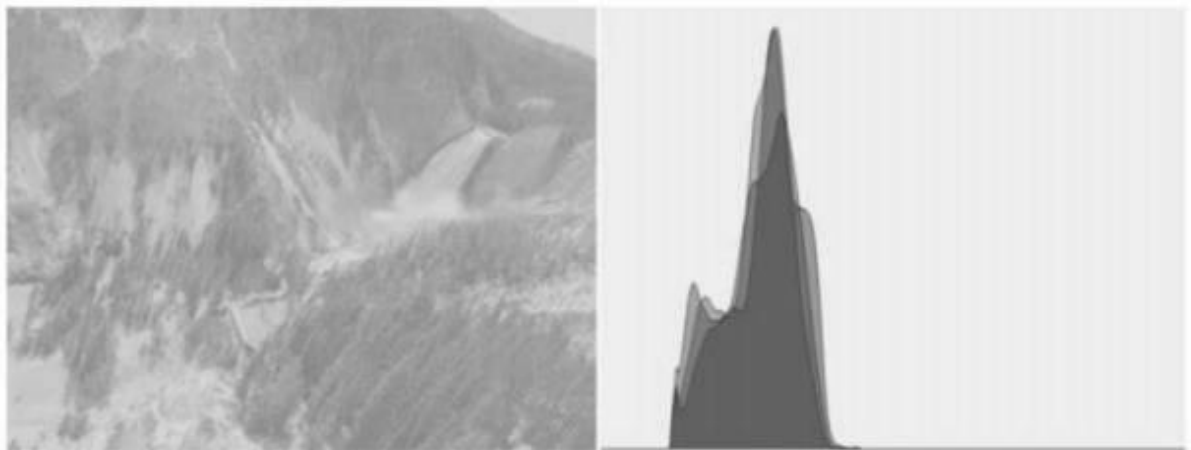


Рис. 2.7 - Погана контрастність зображення, що підтверджує гистограма значень інтенсивності справа

Що лежить в основі корекції гистограмм математика включає в себе відображення одного розподілу (що враховує значення інтенсивності



гістограми) на інший розподіл (в ширину  $i$ , в ідеалі, рівномірно розподіляючи значення інтенсивності). Тобто потрібно розтягнути у-значення початкового розподілу якомога рівномірніше в новому розподілі. Виявляється, що є хороше рішення проблеми поширення розподілу значень: функція повторного відображення повинна бути інтегральною функцією розподілу. Приклад інтегральної функції щільності показаний на рис. 2.8 для випадку з Гауссовим розподілом.

Функцію розподілу можна використовувати для перепризначення початкового розподілу в якості рівномірного поширення розподілу (рис. 2.9), просто переглядаючи кожне  $n$ -значення в первісному розподілі і відстежуючи напрям рівномірного розподілу.

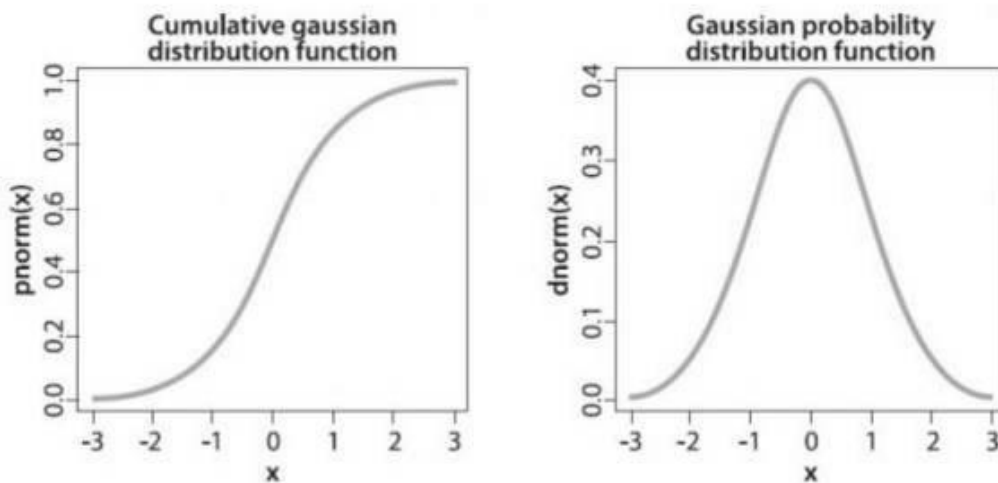


Рис. 2.8 - Інтегральна функція розподілу (зліва) і Гаусовий розподіл (праворуч)

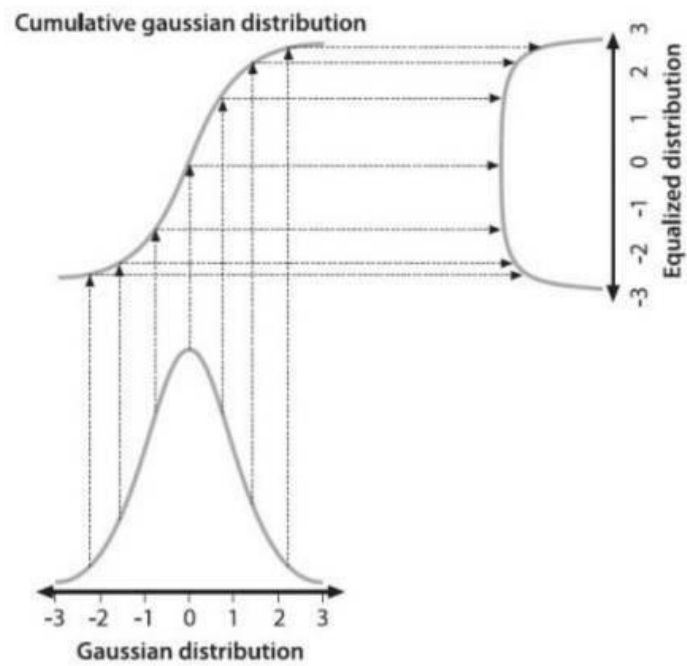


Рис. 2. 9 - Використання інтегральної функції щільності для вирівнювання розподілу Гаусса

Для безперервного розподілу результат буде точним, але для цифрових/дискретних розподілів результати можуть бути далеко не однорідними.

В результаті застосування процесу вирівнювання до рисунка призводить до вирівнювання гістограми розподілу інтенсивності. Весь цей процес "зашитий" в одній функції:

```
void cvEqualizeHist( const CvArr* src ,CvArr* dst
```

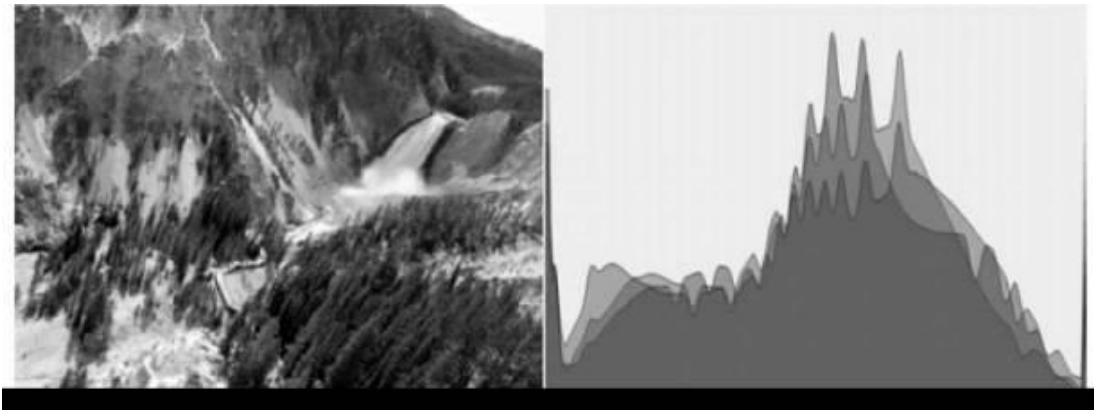


Рис. 2.10 - Результат вирівнювання гістограми

У `cvEqualizeHist()` вихідне і кінцеве зображення повинні бути одноканальними, 8-бітними, одного і того ж розміру. У разі кольорових зображень, кожен канал обробляється окремо.

## 2.3. Алгоритми розпізнавання символів

### 2.3.1. Алгоритм кластеризації k-means

Метод k-середніх - метод кластеризації, прагне мінімізувати сумарне квадратичне відхилення точок кластерів від центрів цих кластерів.

Кластеризація - завдання машинного навчання, що складається в розбитті заданої вибірки об'єктів (даних) на непересічні підмножини / групи

(кластери) на основі близькості їх ознак/значень. Таким чином кожен кластер складається з схожих об'єктів.

Кластеризація дозволяє:

- краще зрозуміти дані (виявивши структурні групи),
- компактне зберігання даних,
- виявлення нових об'єктів.

У OpenCV, алгоритм k-means реалізований в модулі `sxcore`, тому що він був реалізований задовго до появи бібліотеки ML. Алгоритм k-means знаходить кластери в наборі даних і реалізується функцією `cvKMeans2()`.

Алгоритм роботи k-means:

- отримує вхідні дані і бажане число кластерів
- випадковим чином вибирає центри кластерів
- асоціює кожну точку даних з центром найближчого кластера
- переміщення центр кластера в центр ваги асоційованих з ним точок даних.
- повертається на крок 3 поки не досягнута збіжність (тобто центр кластера не рухається)

Проблеми алгоритму k-means:

- не гарантує визначення кращого з можливих розташувань центрів кластерів (досягнення глобального мінімуму сумарного квадратичного відхилення).

Однак, гарантує збіжність до якого-небудь рішення, тобто ітерації не зациклений в нескінченному циклі.

- не визначає скільки кластерів варто використовувати (тобто число кластерів треба знати заздалегідь)
- результат залежить від вибору вихідних центрів кластерів
- k-means передбачає, що просторова коваріаційна складова, або не має значення, або вже була пронумерована.

Поділ векторів на задане число кластерів:

```
CVAPI(int) cvKMeans2 (
```

```

    const CvArr* samples ,int cluster_count ,CvArr*
labels ,CvTermCriteria termcrit ,int attempts
CV_DEFAULT(1) ,CvRNG* rng CV_DEFAULT(0) ,int flags
CV_DEFAULT(0)
    ,CvArr* centers CV_DEFAULT(0)
    ,double* compactness CV_DEFAULT(0)
);

```

samples - матриця прикладів (float) - одна строчка - один вектор;

cluster count - число кластерів;

labels - повертається вектор (int), який зберігає індекс кластера кожного вектора;

termcrit - критерій завершення ітерацій;

attempts - число спроб досягнення кращої компактності;

rng - зовнішній ГВЧ;

flags - прапорець - 0 або:

#define CV\_KMEANS\_USE\_INITIAL\_LABELS 1 \_centers - повертається масив центрів кластерів.

Вираховується після кожної спроби і краще (мінімальне) значення вибирається і відповідні labels повертаються:

```

Summ( || samples(i) - centers(labels(i)) ||L2 )
#include "cxcore.h" #include "highgui.h"
#define MAX_CLUSTERS 5
int main( int argc, char* argv[] ) {
    // зображення для показу точок
    IplImage* img = cvCreateImage( cvSize( 500, 500 ),
8, 3 );
    // таблиця кольорів кластерів CvScalar
color_tab[MAX_CLUSTERS]; color_tab[0] = CV_RGB(255, 0,
0); color_tab[1] = CV_RGB(0,255,0); color_tab[2] =

```

```

CV_RGB(100,100,255); color_tab[3] = CV_RGB(2 55, 0, 2
55); color_tab[4] = CV_RGB(2 55, 2 55, 0);
    // ініціалізація стану ГПСЧ CvRNG rng =
cvRNG(0xffffffff);
    cvNamedWindow( "clusters", 1 );
    for(;;){
        int k, cluster _____ count =
cvRandInt(&rng)%MAX_CLUSTERS
        int i, sample_count = cvRandInt(&rng)%1000 + 1;
        CvMat* points = cvCreateMat( sample_count, 1,
CV_32FC2 );
        CvMat* clusters = cvCreateMat( sample_count, 1,
CV_32SC1 ); // генерація випадкового гаусового
розподілення точок
        for( k = 0; k < cluster_count; k++ ){ CvPoint
center; CvMat point_chunk;
            center.x = cvRandInt(&rng)%img->width; center.y =
cvRandInt(&rng)%img->height; cvGetRows( points,
&point_chunk,
            k*sample_count/cluster_count, k == cluster_count -
1 ?
            sample_count :
            (k+1)*sample_count/cluster_count ); cvRandArr(
&rng, &point_chunk,
            CV_RAND_NORMAL,
            cvScalar(center.x,center.y,0,0), cvScalar(img->
width/6, img->
height/6,0,0) );
        }
        // точки перемішуються

```

```

    for( i = 0; i < sample_count/2; i++ ){
        CvPoint2D32f* pt1 = (CvPoint2D32f*)points-
        >data.fl +
        cvRandInt(&rng)%sample_count; CvPoint2D32f* pt2 =
        (CvPoint2D32f*)points-
        >data.fl +
        cvRandInt(&rng)%sample_count; CvPoint2D32f temp;
        CV_SWAP( *pt1, *pt2, temp );
        } // визначення кластерів
        cvKMeans2( points, cluster_count, clusters,
        cvTermCriteria( CV_TERMCRIT_EPS+CV_TERMCRIT_ITER,
        10, 1.0 )); cvZero( img ) ;
        // показуємо точки
        for( i = 0; i < sample_count; i++ ){
            CvPoint2D32f pt = ((CvPoint2D32f*)points-
            >data.fl)[i];
            // індекс кластера
            int cluster_idx = clusters->data.i[i]; cvCircle(
            img, cvPointFrom32f(pt), 2,
            color_tab[cluster_idx], CV_FILLED );
        }
        cvReleaseMat( &points ); cvReleaseMat( &clusters );
        // показуємо
        cvShowImage( "clusters", img );
        int key = cvWaitKey(330); if( key == 27 ){ // 'ESC'
break;
        }
        }
        // оновлюємо
Ресурси

```

```

cvReleaseImage(&img);
// видаляємо вікна
cvDestroyAllWindows(); return 0;
}

```

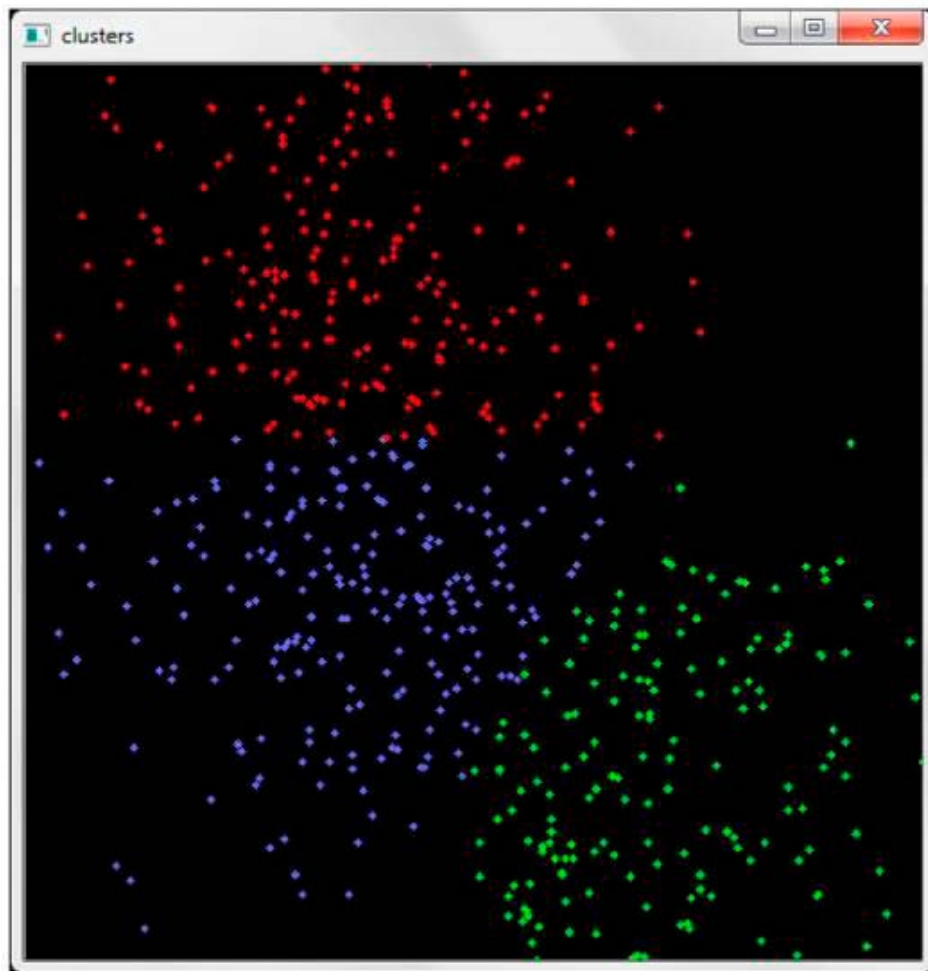


Рис. 2.11 - Кластеризація

Повертає масив строк:

```

CVAPI(CvMat*) cvGetRows( const CvArr* arr, CvMat*
submat,
int start_row, int end_row, int delta_row
CV_DEFAULT(1));
CV_INLINE CvMat* cvGetRow( const CvArr* arr, CvMat*
submat, int
row ) {

```



```

return cvGetRows( arr, submat, row, row + 1, 1 );
} arr - вихідний масив;
submit - покажчик повертання заголовку масиву; startrow
- індекс (від 0) початкового рядка; end_row - індекс
останнього рядка (не включаючи);
delta_row - індекс кроку (тобто вибирається кожна
delta_row^ рядок між [start_row: end_row)).
Повертає масив стовпців:
CVAPI(CvMat*) cvGetCols( const CvArr* arr, CvMat*
submat,
int start_col, int end_col );
CV_INLINE CvMat* cvGetCol( const CvArr* arr, CvMat*
submat, int
col ) {
return cvGetCols( arr, submat, col, col + 1 );
}
arr - вихідний масив;
submit - покажчик повертає заголовок масиву; start_col
- індекс (від 0) початкового стовпця;
end_col - індекс останнього стовпчика (не включаючи).
#include <cv.h> #include <highgui.h>
#define MAX_CLUSTERS 5
int main( int argc, char* argv[] ) {
// для збереження зображення
IplImage* image=0, *gray=0, *bin=0, *dst=0;
//
// завантаження зображення
//
char img_name[] = "Image0.jpg";
// ім'я картинки задається першим параметром

```

```

char* image_filename = argc >= 2 ? argv[1] : img_name;
// отримуємо картинку
image = cvLoadImage(image_filename, 1);
printf("[i] image: %s\n", image_filename); if(!image){
printf("[!] Error: cant load test image: %s\n",
image_filename);
return -1;
}
// показуємо картинку cvNamedWindow("image");
cvShowImage("image", image);
// створення зображень dst = cvCloneImage(image);
gray = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U,
1); bin = cvCreateImage( cvGetSize(image),
IPL_DEPTH_8U, 1);
cvConvertImage(image, gray, CV_BGR2GRAY); cvCanny(gray,
bin, 50, 200, 3);
cvNamedWindow("cvCanny"); cvShowImage("cvCanny", bin);
// для збереження контурів
CvMemStorage* storage = cvCreateMemStorage(0); CvSeq*
contours=0, *seq=0;
// знаходимо контури
int contoursCont = cvFindContours(bin, storage,
&contours,
sizeof(CvContour), CV_RETR_LIST, CV_CHAIN_APPROX_SIMPLE,
cvPoint(
if(!contours){
printf("[!] Error: cant find contours!\n"); return -2;
}
printf("[i] contours: %d \n", contoursCont); // рисуємо
контури

```

```

for(seq = contours; seq!=0; seq = seq->h_next){
cvDrawContours(image, seq, CV_RGB(255,216,0),
CV_RGB(0,0,250), 0, 1, 4); // рисуємо контур }
cvNamedWindow("contours"); cvShowImage("contours",
image);
cvConvertImage(bin, dst, CV_GRAY2BGR); //
// таблиця кольорів кластерів //
CvScalar color_tab[MAX_CLUSTERS]; color_tab[0] =
CV_RGB(2 55,0,0); color_tab[1] = CV_RGB(0,2 55,0);
color_tab[2] = CV_RGB(100,100,255); color_tab[3] =
CV_RGB(2 55,0,2 55); color tab[4] = CV RGB(255, 255,
0);
cvNamedWindow("clusters"); {
int cluster _____ count = MAX_CLUSTERS;
int sample_count = contoursCont; int i, j, k;
CvMat* points = cvCreateMat( sample_count, 1,
CV_32FC2 );
CvMat* clusters = cvCreateMat( sample_count, 1,
CV_32SC1 );
// визначаємо точки контурів
for(k=0, seq = contours; seq!=0; seq = seq->h_next,
k++){
CvPoint2D32f center = cvPoint2D32f(0, 0);
#if 0
float radius=0;
// визначаємо коло, що охоплює
контур
cvMinEnclosingCircle(seq, Фег, &radius);
#else
// знаходимо центр мас int count=0;

```

```

for(i=0; i<seq->total; ++i ) {
CvPoint* p = CV_GET_SEQ_ELEM(
CvPoint, seq, i );
center.x+=p->x; center.y+=p->y; count++;
}
center.x=center.x/count; center.y=center.y/count;
#endif
// відмічаємо точку
cvCircle( image, cvPointFrom32f(center), 2, CV_RGB(2
00,0,0), CV_FILLED );
// заносимо в масив
cvSet1D( points, k, cvScalar(center.x,
center.y, 0) );
}
cvShowImage("contours", image); // визначення кластерів
cvKMeans2( points, cluster_count, clusters,
cvTermCriteria( CV_TERMCRIT_EPS+CV_TERMCRIT_ITER, 10,
1.0 ));
// показуємо точки
for( i = 0; i < sample_count; i++ ){
CvPoint2D32f pt = ((CvPoint2D32f*)points-
>data.fl)[i];
// індекс кластера
int cluster_idx = clusters->data.i[i]; cvCircle( dst,
cvPointFrom32f(pt), 2,
color_tab[cluster_idx], CV_FILLED );
}
cvReleaseMat( &points ); cvReleaseMat( &clusters );
// показемо
cvShowImage( "clusters", dst );

```

```

int key = cvWaitKey(0); //if( key == 27 ){ // ESC //
break;
//}
}
//
// вивільнюємо ресурси //
cvReleaseImage(&image); cvReleaseImage(&gray);
cvReleaseImage(&bin); cvReleaseImage(&dst);
cvReleaseMemStorage(&storage);
// видаляємо вікна
cvDestroyAllWindows(); return 0;
}

```



Рис. 2.12 - Робота алгоритму k-means на конкретному прикладі

### 2.3.2 Метод опорних векторів

Метод опорних векторів - набір схожих алгоритмів навчання з учителем, що використовуються для задач класифікації і регресійного аналізу. Належить до сімейства лінійних класифікаторів, може також розглядатися як спеціальний випадок регуляризації по Тихонову. Особливою властивістю методу опорних векторів є невпинне зменшення емпіричної помилки класифікації і збільшення зазору, тому метод також відомий як метод класифікатора з максимальним зазором.

Основна ідея методу - переклад вихідних векторів в простір більш високої розмірності і пошук розділяє гіперплощини з максимальним зазором в цьому просторі. Дві паралельні гіперплощини будуються по обидва боки гіперплощини, що розділяє наші класи. Розділяє гіперплощиною буде гіперплощиною, що максимізує відстань до двох паралельних гіперплощостей. Алгоритм працює в припущенні, що чим більша різниця або відстань між цими паралельними гіперплощостями, тим менше буде середня помилка класифікатора.

Часто в алгоритмах машинного навчання виникає необхідність класифікувати дані. Кожен об'єкт даних представлений як вектор (точка) в  $P$ -вимірному просторі (послідовність  $P$  чисел). Кожна з цих точок належить тільки одному з двох класів. Нас цікавить, чи можемо ми розділити точки гіперплощиною розмірністю  $(P-1)$ . Це типовий випадок лінійної роздільності. Таких гіперплощостей може бути багато. Тому цілком природно вважати, що максимізація зазору між класами сприяє більш впевненою класифікації. Тобто чи можемо ми знайти таку гіперплощиною, щоб відстань від неї до найближчої точки було максимальним. Це б означало, що відстань між двома найближчими точками, що лежать по різні боки гіперплощини, максимально. Якщо така гіперплощиною існує, то вона нас буде цікавити найбільше; вона називається оптимальною розділяє гіперплощиною, а

відповідний їй лінійний класифікатор називається оптимально розділяє класифікатором.

Ми вважаємо, що точки мають вигляд:

$$\{(x_1, c_1), (x_2, c_2), \dots, (x_n, c_n)\} \quad (2.4)$$

де  $c$  приймає значення 1 або -1, в залежності від того, якого класу належить точка  $X$ . Кожне  $x$  - це  $P$ -мірний речовинний вектор, зазвичай нормалізований значеннями  $[0,1]$  або  $[-1,1]$ . Якщо крапки не будуть нормалізовані, то точка з великими відхиленнями від середніх значень координат точок дуже сильно вплине на класифікатор. Ми можемо розглядати це як навчальну колекцію, в якій для кожного елемента вже заданий клас, до якого він належить. Ми хочемо, щоб алгоритм методу опорних векторів класифікував їх таким же чином. Для цього ми будемо розділяти гіперплощину, яка має вигляд:

$$w \cdot x - b = 0 \quad (2.5)$$

Вектор  $w$  - перпендикуляр до розділяючої гіперплощини. Параметр  $b$  порівнює по модулю відстані від гіперплощини до початку координат. Якщо параметр  $b$  дорівнює нулю, гіперплощина проходить через початок координат, що обмежує рішення.

Так як нас цікавить оптимальний розподіл, нас цікавлять опорні вектори і гіперплощини, паралельні оптимальній і найближчі до опорних векторів двох класів. Можна показати, що ці паралельні гіперплощини можуть бути описані наступними рівняннями (з точністю до нормування).

$$w \cdot x - b = 1 \quad (2.6)$$

$$w \cdot x - b = -1 \quad (2.7)$$

Якщо навчальна вибірка лінійно роздільна, то ми можемо вибрати гіперплощини таким чином, щоб між ними не лежала жодна точка навчальної вибірки і потім максимізувати відстань між гіперплоскостями. Ширину смуги між ними легко знайти з міркувань геометрії, таким чином наше завдання мінімізувати  $w$ . Щоб виключити всі точки зі смуги, ми повинні переконатися для всіх  $i$ , що

Алгоритм побудови оптимальної розділяючої гіперплоскості, запропонований в 1963 році Володимиром Вапніка і Олексієм Червоненкіса - алгоритм лінійної класифікації. Однак в 1992 році Бернхард Босер, Ізабель Гийон і Вапник запропонували спосіб створення нелінійного класифікатора, в основі якого лежить перехід від скалярних творів до довільних ядер, так званий *kernel trick* (запропонований вперше М. А. Айзерманом, Е. М. Броверманом і Л. В. Розоноером для методу потенційних функцій), що дозволяє будувати нелінійні роздільники. Результируючий алгоритм вкрай схожий на алгоритм лінійної класифікації, з тією лише різницею, що кожне скалярний твір в наведених вище формулах замінюється нелінійною функцією ядра (скалярним твором в просторі з більшою розмірністю). У цьому просторі вже може існувати оптимальна розділяє гіперплоскість. Так як розмірність одержуваного простору може бути більше розмірності вихідного, то перетворення, зіставляє скалярні твори, буде нелінійним, а це свідчить про те, відповідна в вихідному просторі оптимальної розділяє гіперплоскості, буде також нелінійною.

Варто відзначити, що якщо початковий простір має досить високу розмірність, то можна сподіватися, що в ньому вибірка виявиться лінійно роздільна.



### 3 ОПИС WINDOWS - ДОДАТКУ

#### 3.1 Опис структурної схеми додатку

Програма складається з декількох основних файлів:

- а) `LicencePlateRecognition.java` - файл, який містить основний програмний код. У цьому файлі здійснюється динамічне створення форми Windows- додатку та її компонентів, завантаження зображень, а також саме розпізнання автомобільних номерних знаків.
- б) `ALPR.iml` - модульна бібліотека, яка існує тільки як залежність для даного модулю. Це є архівом скомпільованого коду.
- в) `ALPR.ipr` - конфігураційний файл, який містить інформаційне ядро проекту.
- г) `ALPR.iws` - конфігураційний файл, який містить персональні налаштування робочого простору.

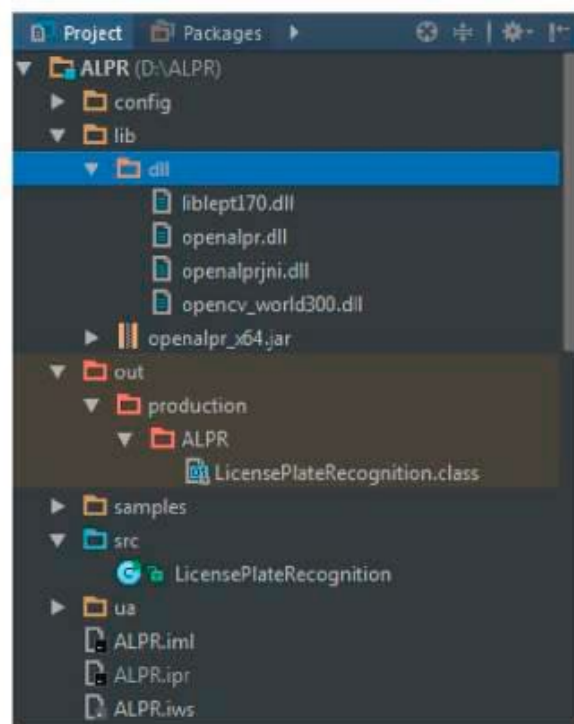


Рис. 3.1 - Структурна схема програми

У Windows-додатку розташовані такі компоненти як:

Компонент File - компонент, що використовується для створення каталогу або дерева каталогів. Також можна дізнатись властивості файлів (розмір, дату останньої зміни, режим читання/запису), визначити до якого типу (файл чи каталог) відноситься файл, видалити файл.

Компонент JLabel - компонент, який є в Java Swing. Використовується у тому випадку, коли треба відобразити будь-яке повідомлення для користувача або зробити текстову мітку для поля введення. Текст, який показує JLabel виділяти неможна, тільки побачити.

Компонент JTextArea - компонент, який використовується для створення поля введення і відображення тексту. Можна вводити і виводити необмежене число строк.

Компонент JFrame - компонент, який представляє собою вікно з рамкою і строкою заголовку (з кнопками «Згорнути», «У весь екран», «Закрити»). Воно може змінювати розміри і переміщуватись по екрану.

Компонент JSplitPane - компонент, який представляє собою панель, що розділена на дві області, межу між якими користувач може переміщувати.

Компонент JPanel - це елемент управління, що представляє собою прямокутний простір, на якому можна розміщати інші елементи. Елементи додаються та видаляються методами, які унаслідовані від класа Container.

Компонент JToolBar - компонент, який використовується для розміщення в строку чи стовпець компонент, що визначає найбільш виконувані дії.

Компонент JButton - компонент, який представляє собою кнопку. У кнопки є різні методи для її конфігурування - установка надпису, установка іконки, вирівнювання тексту, установка розмірі і так далі. Також для кнопки можна зробити слухача, тобто запрограмувати кнопку таким чином, що при натисненні на неї будуть виконуватись відповідні дії.

Компонент JFileChooser - компонент, який представляє собою діалог для вибору файлів з комп'ютеру. Також діалог JFileChooser дозволяє виконувати навігацію по файловій системі. JFileChooser надає тільки можливість вибору файлу чи директорії.

Компонент JMenuBar - це головне меню, яке розташовано у верхній часті вікна додатку у вигляді горизонтальної полоски. Меню може мати довільну вкладеність. Меню складається з пунктів меню.

Компонент JMenu - це меню, яке містить всередині себе компоненти JMenuItem або вкладені меню JMenu. Потрібний для організації ієрархії вкладеності меню.

### 3.2 Пояснення до програмного коду

Розбір змісту модуля: Підключення бібліотек до проекту:

```
import com.openalpr.jni.Alpr; import
com.openalpr.jni.AlprPlate; import
com.openalpr.jni.AlprPlateResult; import
com.openalpr.jni.AlprResults;
import javax.swing.*;
import javax.swing.border.Border;
import javax.swing.filechooser.FileNameExtensionFilter;
import java.awt.*;
import java.awt.event.*;
import java.io.File;
import java.nio.file.Files;
Головний клас проекту:
public class LicensePlateRecognition extends JFrame {
private File imageFile; private JLabel imageLabel;
private JTextArea resultTextArea; private String
country; private JLabel resultLabel;
```

```

private static String NL =
System.getProperty("line.separator");
public LicensePlateRecognition() throws
HeadlessException {
this.initComponents();
}
Динамічне створення візуальних компонентів проекту:
private void initComponents() {
this.initFrame(); this.initMenuBar();
this.initSplitPane(); this.initToolBar();
}
private void initSplitPane() {
this.imageLabel = new JLabel( (String)null,
SwingConstants.CENTER);
this.imageLabel.setOpaque(true); this.resultTextArea =
new JTextArea(25,40);
this.resultTextArea.setEditable(false);
this.resultLabel = new JLabel();
this.resultTextArea.setFont(this.resultTextArea.getFont
().deriveFont (Font.PLAIN, 12f));
JPanel mainPanel = new JPanel(new BorderLayout());
JPanel leftPanel = new JPanel();
leftPanel.setLayout(new BoxLayout(leftPanel,
BoxLayout.Y_AXIS));
//leftPanel.setBorder(createAppBorder());
JPanel imagePanel = new JPanel();
imagePanel.setLayout(new
FlowLayout(FlowLayout.CENTER));
imagePanel.add(this.imageLabel);
JScrollPane scrollPane = new JScrollPane(imagePanel);

```

```

scrollPane.setBorder(createAppBorder());
leftPanel.add(scrollPane);
JPanel rightPanel = new JPanel();
rightPanel.setLayout(new BoxLayout(rightPanel,
BoxLayout.Y_AXIS));
rightPanel.setBorder(createAppBorder());
JPanel strip = new JPanel(new
FlowLayout(FlowLayout.LEFT));
strip.add(new JLabel("Вероятнее всего:"));
this.resultLabel.setFont(this.resultLabel.getFont().der
iveFont(Font. BOLD, 14.0f));
strip.add(resultLabel); rightPanel.add(strip);
rightPanel.add(Box.createHorizontalStrut(5));
rightPanel.add(this.resultTextArea);
JSplitPane splitPane = new
JSplitPane(JSplitPane.HORIZONTAL_SPLIT, rightPanel,
leftPanel); splitPane.setBorder(null);
splitPane.setContinuousLayout(true);
splitPane.setOneTouchExpandable(true);
mainPanel.add(splitPane, BorderLayout.CENTER);
this.setContentPane(mainPanel);
}

private void initToolBar() {
JToolBar toolBar = new JToolBar();
JButton button = new JButton(new
AbstractAction("Завантажити зображення...")
Реалізація діалогу вибору зображення з комп'ютера:
@Override
public void actionPerformed(ActionEvent e) {

```

```

JFileChooser chooser = new JFileChooser();
chooser.setDialogTitle("Select image file");
chooser.setFileFilter(new
FileNameExtensionFilter("Image files", "png", "jpg",
"gif"));
chooser.setCurrentDirectory(new
File("").getAbsoluteFile());
if (
chooser.showOpenDialog(LicensePlateRecognition.this) ==
JFileChooser.APPROVE_OPTION )
{
File f = chooser.getSelectedFile();
if ( f != null && f.exists() && f.canRead() ) {
LicensePlateRecognition.this.imageFile = f;
LicensePlateRecognition.this.imageLabel.setIcon(new
ImageIcon(f.getAbsolutePath()));
LicensePlateRecognition.this.revalidate();
}
}
});
button.setFont(button.getFont().deriveFont(Font.BOLD));
toolBar.add(button);
toolBar.addSeparator(new Dimension(20,20));
JLabel label = new JLabel("Страна/Регион"), •
label.setFont(label.getFont().deriveFont(Font.BOLD));
toolBar.add(label);
toolBar.add(Box.createHorizontalStrut(5));
JComboBox<String> countryComboBox = new JComboBox<>(new
String[]{"eu", "us", "au"});

```

```

countryComboBox.setEditable(false);
countryComboBox.addItemListener(new ItemListener()
Створення списку країн. При виборі відповідної країни
змінюється і алгоритм розпізнавання автомобільного
номера:
@Override
public void itemStateChanged(ItemEvent e) {
LicensePlateRecognition.this.country =
String.class.cast(e.getItem()); }
});
countryComboBox.setRenderer(new
DefaultListCellRenderer() {
@Override
public Component getListCellRendererComponent(JList<?>
list, Object value,
int index, boolean isSelected, boolean cellHasFocus)
JLabel label =
JLabel.class.cast(super.getListCellRendererComponent(li
st, value,
index, isSelected, cellHasFocus));
label.setText(label.getText().toUpperCase()); return
label;
}
});
countryComboBox.setSelectedIndex(0);
countryComboBox.setMaximumSize(new Dimension(50, 20));
this.country =
countryComboBox.getModel().getElementAt(0);
toolBar.add(countryComboBox);
button = new JButton(new AbstractAction("ВЫПОЛНИТЬ")

```

Виконання алгоритму розпізнавання автомобільного номеру:

```
@Override
public void actionPerformed(ActionEvent e) {
    LicensePlateRecognition.this.process();
}
});
button.setFont(button.getFont().deriveFont(Font.BOLD));
toolBar.add(Box.createHorizontalStrut(20));
toolBar.add(button);
toolBar.add(Box.createHorizontalStrut(20));
//toolBar.add(Box.createHorizontalGlue());
button = new JButton(new AbstractAction("Очистить")
Очищує панель з зображенням та панель з інформацією про
розпізнаний автомобільний номер:
```

```
@Override
public void actionPerformed(ActionEvent e) {
    LicensePlateRecognition.this.imageLabel.setIcon(null);
    LicensePlateRecognition.this.imageFile = null;
    LicensePlateRecognition.this.resultTextArea.setText(null);
    LicensePlateRecognition.this.resultLabel.setText(null);
}
});
button.setFont(button.getFont().deriveFont(Font.BOLD));
toolBar.add(button);
this.getContentPane().add(toolBar,
BorderLayout.PAGE_START); }
```

Створення меню:

```
private void initMenuBar() {
```



```

JMenuBar menuBar = new JMenuBar(); JMenu fileMenu = new
JMenu("Файл"), •
fileMenu.add(new JMenuItem(new AbstractAction("Выход")
{
@Override
public void actionPerformed(ActionEvent e) {
LicensePlateRecognition.this.shouldClose();
}
}));
menuBar.add(fileMenu); this.setJMenuBar(menuBar);
}
private void initFrame() {
this.setTitle("Розпізнавання номерних знаків");
this.setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);
this.setLocationRelativeTo(null);
this.setSize(700,400);
this.addWindowListener(new WindowAdapter() {
@Override
public void windowClosing(WindowEvent e) {
LicensePlateRecognition.this.shouldClose();
}
});
}
При виході з програми з'являється вікно, яке запитує
чи дійсно користувач хоче вийти з програми:
private void shouldClose() {
if (JOptionPane.showConfirmDialog(this, "Це закриє
застосунок.\п\пВиконати?", "Вихід",
JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION )
System.exit(0);

```

```

}
Механізм розпізнавання автомобільного номеру:
private void process() {
    if ( this.imageFile == null ) return;
    Alpr alpr = null;
    try
    {
        alpr = new Alpr(this.country, "config/openalpr.conf",
            "config/runtime_data");
        alpr.setTopN(5);
        byte[] imageData =
            Files.readAllBytes(this.imageFile.toPath());
        AlprResults results = alpr.recognize(imageData);
        StringBuilder sb = new StringBuilder();
        sb.append("Версія OpenALPR:
        ").append(alpr.getVersion()).append(NL);
        sb.append("Розмір зображення:
        ").append(results.getImgWidth()).append("x").append(res
            ults.getImgHeight()).append(NL);
        sb.append("Час
        виконання:").append(results.getTotalProcessingTimeMs())
            .append(" мс").append(NL), •
        sb.append(NL);
        sb.append("Номерів знайдено:
        ").append(results.getPlates().size()).append(NL);
        sb.append(NL);
        sb.append(String.format(" %-15s%-8s", "Номер #",
            "Точність")).append(NL), • for (AlprPlateResult
            plateResult : results.getPlates()
        )
    }

```

```

{
for (AlprPlate plate : plateResult.getTopNPlates()
)
{
if ( plate.isMatchesTemplate() )
sb.append(" * "); else
sb.append(" - ");
sb.append(String.format("%-15s%-8f",
plate.getCharacters(),
plate.getOverallConfidence())) .append(NL);
}
sb.append(NL);
}
if ( results.getPlates() != null &&
!results.getPlates().isEmpty())
this.resultLabel.setText(results.getPlates().get(0).get
BestPlate().getCharacters());
else
this.resultLabel.setText(null);
this.resultTextArea.setText(sb.toString());
}
catch (Exception e) {
e.printStackTrace();
}
finally {
if ( alpr != null ) alpr.unload();
}
}
private static Border createAppBorder() {
return

```

```

BorderFactory.createCompoundBorder (BorderFactory.create
EmptyBorder(5
,5,5,5),
BorderFactory.createCompoundBorder (BorderFactory.create
EtchedBorder(
), BorderFactory.createEmptyBorder(5,5,5,5))); }
public static void main (String...args) {
SwingUtilities.invokeLater(new Runnable() {
@Override
public void run() {
try {
UIManager.setLookAndFeel (UIManager.getSystemLookAndFeel
ClassName());
}
catch (Exception e) {
e.printStackTrace();
}
new LicensePlateRecognition().setVisible(true);
}
});

```

### 3.3 Контрольний приклад

При відкритті програма виглядає таким чином:

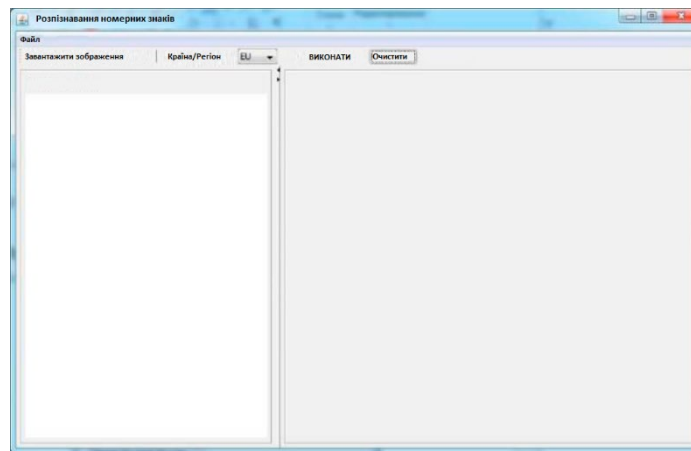


Рис. 3.2 - Вигляд програми при відкритті

При натисненні кнопки «Завантажити зображення...» з'являється діалог, після чого можна здійснити вибір необхідного зображення автомобіля для розпізнавання номеру:



Рис. 3.3- Діалог вибору зображення

Після вибору зображення з'являється на формі програми у правій панелі

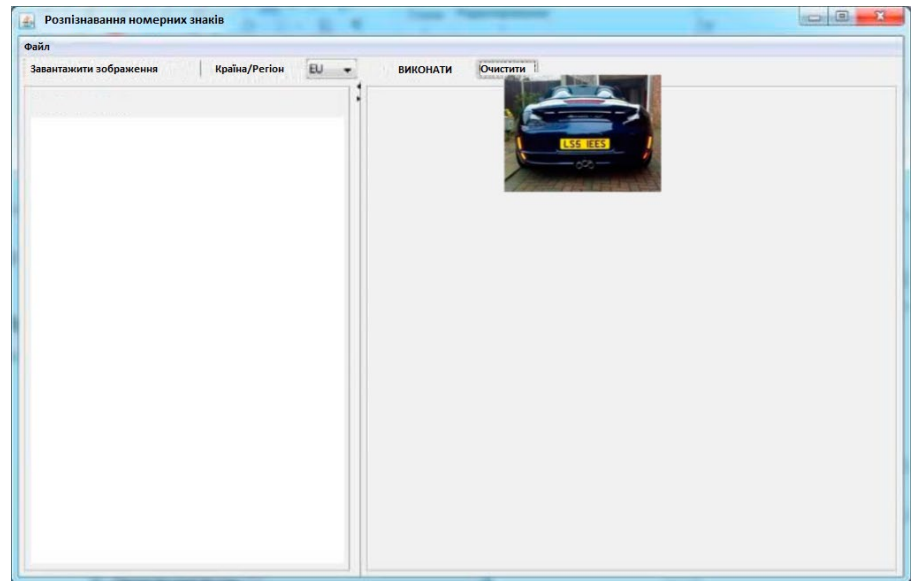


Рисунок 3.4 - Завантаження зображення у програму

Тепер необхідно обрати країну або регіон відповідно до номерного знаку автомобіля:

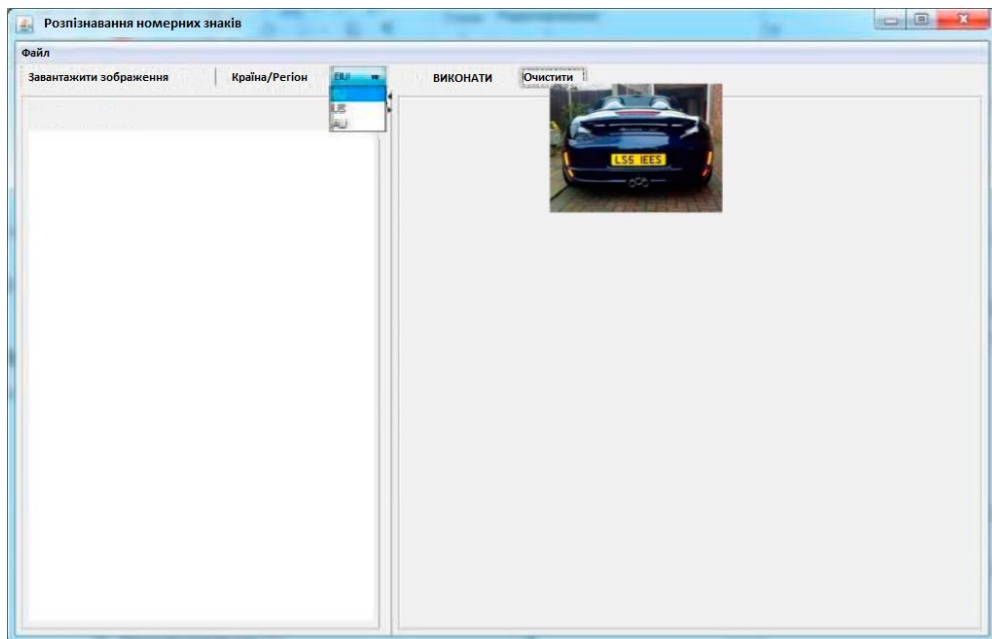


Рисунок 3.4 - Вибір країни або регіону

При натисненні кнопки «ВІКОНАТИ» у лівій панелі програми з'являється розпізнаний номер у текстовому форматі та інша інформація про роботу Windows-додатку:

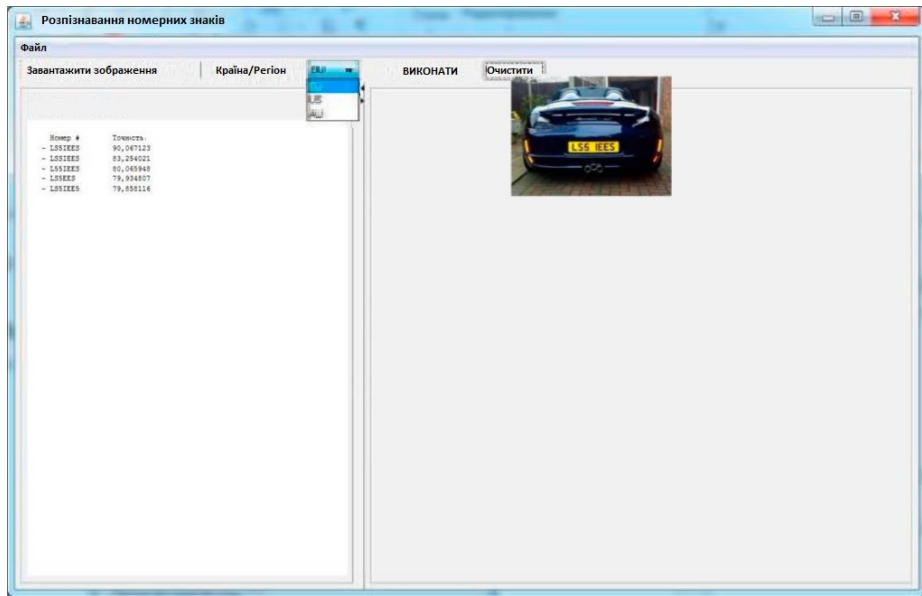


Рисунок 3.5 - Розпізнавання автомобільного номерного знаку

При натисненні кнопки «Очистити» зображення а також вся інформація пропадають:

## **ВИСНОВКИ**

В результаті виконання випускної роботи були виконані усі завдання, а саме проведено аналіз методів розпізнавання графічних об'єктів та розроблено Windows-додаток для розпізнавання автомобільних номерних знаків з зображення. Також, у майбутньому планується здійснити переробку інтерфейсу програми, зробити програму більш зручної, додати можливість розпізнавати автомобільні номерні знаки деяких інших країн та розпізнавання автомобільного номерного знаку з відео і створення бази даних для зберігання результатів на певний час.



## ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ali Arya, Farzin Farhadi-Niaki [Електронний ресурс] / An Implementation on Object Move Detection Using OpenCV - Режим доступу:  
<http://www.sce.carleton.ca/~ffniaki/An Implementation on Object Move Detection Using OpenCV.pdf> - Дата доступу: 09.09.20245.
2. OpenCV v2.4.1 documentation [Електронний ресурс] / Feature detection - Режим доступу:  
[http://docs.opencv.org/modules/imgproc/doc/feature\\_detection.html?highlight=canny#cv.Canny](http://docs.opencv.org/modules/imgproc/doc/feature_detection.html?highlight=canny#cv.Canny) - Дата доступу: 15.09.2024.
3. OpenCV v2.4.1 documentation [Електронний ресурс] / Miscellaneous Image Transformations - Режим доступу: [http://docs.opencv.org/modules/imgproc/5.doc/miscellaneous\\_transformations.html](http://docs.opencv.org/modules/imgproc/5.doc/miscellaneous_transformations.html) - Дата доступу: 18.09.2024.
4. OpenCV v2.4.1 documentation [Електронний ресурс] / Operations on Arrays - Режим доступу: [http://opencv.itseez.com/modules/core/doc/operations\\_on\\_arrays.html](http://opencv.itseez.com/modules/core/doc/operations_on_arrays.html) - Дата доступу: 04.10.2024.
5. OpenCV v2.4.1 documentation [Електронний ресурс] / Structural Analysis and Shape Description - Режим доступу: [http://docs.opencv.org/modules/imgproc/doc/structural\\_analysis\\_and\\_shape\\_descriptors.html?highlight=findcontours#cv.Fin dContours](http://docs.opencv.org/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=findcontours#cv.Fin dContours) - Дата доступу: 14.10.2024.
6. Onionesque Reality [Електронний ресурс] / A Note on the Graph Laplacian - Режим доступу: [onionesquereality.wordpress.com](http://onionesquereality.wordpress.com) - Дата доступу: 25.10.2024.
7. Learning OpenCV [Електронний ресурс] / Learning OpenCV. - Режим доступу: [locv.ru](http://locv.ru) - Дата доступу: 30.10.2024.
8. Robo Craft [Електронний ресурс] / Использование алгоритма Хафа для коррекции наклона фотографии - Режим доступу:  
<http://robocraft.ru/blog/computervision/502.html> - Дата доступу: 30.11.2024.

9. Лісова І.Д. Проектування та вдосконалення інформаційних систем і комп'ютерних мереж: методичні вказівки до економічного обґрунтування дипломних робіт / І.Д. Лісова - Одеса:ОНМУ, 2004. - 51 с.
10. Жидецький В.Ц. Охорона праці користувачів комп'ютерів. Навчальний посібник. - Вид. 2-ге, доп. / В.Ц. Жидецький - Львів:Афіша, 2000. - 176 с.
11. Скріпкин, К.Г. Економічна ефективність інформаційних систем: Лекції МГУ / До. Р. Ськріпкин. - М.: ДМК Прес, 2008. - 256 с.
20. Филимонов Г.В. Гражданская оборона объектов морского транспорта / Г.В. Филимонов, Ю.Ф. Кустов, А.И. Яковлев, М.А. Крутских. - М.: Транспорт, 1983. 58-60с.
21. Стеблюк М.І. Цивільна оборона: Підручник / М.І. Стеблюк - К.: Знання, 2006. - 29 с.
22. Депутат О.П. Цивільна оборона. Навчальний посібник / О.П. Депутат, І.В. Коваленко, І.С. Мужик — Львів: Афіша. 2000 —142-147 с.