МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Навчально-науковий інститут новітніх освітніх технологій
Кафедра комп'ютерних наук

ЮХТА Олександр Андрійович

Математичне та програмне забезпечення ідентифікації здобувачів освіти коледжу за студентським квитком / Mathematical Tools and Software for Identification of College Students by Student ID

спеціальність: 121 - Інженерія програмного забезпечення
освітньо-професійна програма - Інженерія програмного забезпечення

Кваліфікаційна робота

Виконав студент групи
ІПЗзмл-21
О. А. Юхта

_____

Науковий керівник:
к.е.н., доцент, Л. І. Гончар

_____

Кваліфікаційну роботу
допущено до захисту:

"___" _____ 20___ р.

Завідувач кафедри
_____ **А. В. Пукас**

ТЕРНОПІЛЬ - 2022

ЗМІСТ

ВСТУП 3
РОЗДІЛ 1 АНАЛІЗ МЕТОДІВ ТА ПІДХОДІВ ДО КОДУВАННЯ ІНФОРМАЦІЇ .. 7
1.1. Теоретичні аспекти представлення та кодування інформації 7
1.2. Застосування штрих-кодів для кодування інформації 11
1.3. Класифікація штрих-кодів, аналіз принципів побудови їх різновидів 16
1.4. Формалізація вимог до програмної системи 32
Висновки до розділу 1 33
РОЗДІЛ 2 МАТЕМАТИЧНЕ ТА АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ РОЗПІЗНАВАННЯ ШТРИХ-КОДІВ 34
2.1. Основні проблеми задачі читання штрих-кодів 34
2.2. Алгоритми виявлення та обробки штрих-кодів 35
2.2.1. Метод, заснований на основних морфологічних операціях 38
2.2.2. Метод на основі сканування зображень 38
2.2.3. Метод на основі фільтрації чорного циліндра 39
2.3. Узагальнена модель розшифрування штрих-коду на базі камери 40
2.4. Опис та реалізація алгоритму зчитування штрих-кодів 43
Висновки до розділу 2 48
РОЗДІЛ 3 ОПИС ПРОГРАМНОЇ СИСТЕМИ ТА ЇЇ РЕАЛІЗАЦІЇ 50
3.1. Архітектура розробленої системи 50
3.2. Опис структури та організації бази даних 52
3.3. Опис розробленого програмного забезпечення 55
Висновки до розділу 3 60
ВИСНОВКИ 61
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 62
ДОДАТКИ 69

ВСТУП

Використання смартфонів справило революцію у світі у багатьох відношеннях, включаючи систему освіти, з численними потенційними та реалізованими перевагами. Хоча функції смартфонів, такі як обмін текстовими повідомленнями, мультимедіа та підключення до Інтернету, можуть здатися суто розважальними, їх можна використовувати в навчальному закладі для різноманітних цілей. Смартфон може бути як засобом безпосередньої комунікації студентів та викладачів, допоміжним інструментом у проведенні навчальних занять для поширення матеріалу, виконання деякого типу завдань та роботи у синхронному режимі, так і забезпечувати впровадження та виконання більш глобальних функцій освітнього процесу.

Відвідування студентів відіграє важливу роль в оцінці успішності здобувачів освіти та навчального закладу загалом. На жаль, у коледжах, як і у багатьох закладах вищої освіти, зазвичай немає автоматизованого додатка для ведення обліку відвідування навчальних занять. Тому існує потреба в інструменті для систематичного ведення обліку відвідуваності студентів у зв'язку із тенденцією зростання кількості здобувачів освіти коледжів та ЗВО в цілому. Це і визначає ***актуальність теми***.

З допомогою смартфону можна впровадити систему ідентифікації студентів з мінімальними потрібними для цього засобами та ресурсами. Крім того, саме на базі такої системи можна організувати управління відвідуванням студентів, щоб прискорити процес обліку відвідування викладачами, безпосередніми керівниками та адміністрацією і отже скоротити час, людські помилки та кількість зайвої роботи порівняно з ручною системою ведення присутності на заняттях. Відстеження відвідування та активностей здобувачів є двома важливими проблемами у більшості закладів вищої освіти, оскільки вони є частиною процесу оцінки студентів, а також щорічних перевірок [1].

За останні кілька десятиліть, поряд з розвитком інформаційних технологій, в наше довкілля було внесено багато значних покращень, таких як розумні міста,

штучний інтелект, розумний автомобіль. Оскільки все навколо продовжує постійно вдосконалюватись, то доцільно покращити використання сучасної системи ідентифікації та відстеження відвідування. У минулі часи і дотепер викладачі робили переклички або роздавали листи-бланки присутності, щоб записувати відвідування як доказ. Тим не менш, сьогодні студенти дуже розумні та хитрі, у них може бути кілька способів відзначити свою присутність, навіть якщо вони відсутні на лекційному та практичному занятті. Тепер, в епоху Інтернету речей, існує набагато більше пристроїв, які можна використовувати як сучасний спосіб обліку відвідування здобувачів освіти. Мета обліку відвідування полягала в тому, щоб переконатися, що вони можуть отримати знання та досвід в університеті чи коледжі, але не витрачати гроші на “канікули” всередині закладу освіту, а це означає, що точність даних про відвідування має бути строгою та чіткою [2].

Проблему ідентифікації здобувачів освіти, для яких потрібно реалізувати контроль за відвідуванням занять, можна вирішити за допомогою студентських квитків.

Студентський квиток – це документ з персональними даними про особу здобувача освіти, в тому числі з фотокарткою. Кожен здобувач освіти при вступі до університету, інституту, академії чи коледжу отримує цей документ. Крім персональних даних, студентський квиток містить на собі штрих-код – машиночитане представлення інформації у візуальному форматі [3].

Незважаючи на те, що в літературі описано безліч методів вирішення проблеми відстеження студентів та їх активностей, доцільно обрати саме технологію штрих-коду, тому що вона дешева та відносно проста у реалізації. Ще більше здешевити вартість реалізації допоможе використання смартфону замість спеціального сканера зчитування штрих-коду [4]. Здобувач освіти може представити свій студентський квиток зі штрих-кодом, який можна зчитати з допомогою камери смартфону, таким чином провівши візуальну ідентифікацію та звірку даних. Паралельно з цим за унікальним ідентифікатором, зчитаним через штрих-код, можна отримати доступ до запису студента у базі даних

здобувачів освіти навчального закладу та програмно внести відмітки про відвідування того чи іншого заняття. Після цього, записавши відповідні дані, з бази можна отримувати різноманітні витяги зі звітністю для викладача, куратора групи, дирекції.

Метою дослідження є автоматизація процесу ідентифікації здобувачів освіти та обліку їх відвідування шляхом розробки математичного та програмного забезпечення розпізнавання і декодування штрих-кодів.

Для досягнення мети необхідне вирішення наступних ***завдань***:

1. Виконати огляд кодування інформації за допомогою штрих-кодів, розглянути їх класифікацію.
2. Розглянути існуючі методи, алгоритми розпізнавання, зчитування та декодування штрих-кодів, зокрема, за допомогою камери смартфону.
3. На основі існуючих підходів до опрацювання штрих-кодів реалізувати можливість зчитування штрих-кодів для сканування студентських квитків.
4. Виконати розробку архітектури програмного засобу та організацію системи збереження даних, що використовуються для ідентифікації та обліку здобувачів освіти за їх квитками.
5. Розробити програмне забезпечення, що реалізує ідентифікацію здобувачів освіти за студентськими квитками та роботу зі сховищем даних для фіксації їх перебування у навчальному закладі та формування звітності щодо відвідування.

Об'єктом дослідження є технології, алгоритмічне та математичне забезпечення зчитування та декодування штрих-кодів. ***Предметом дослідження*** є процес розробки системи ідентифікації здобувачів освіти та обліку відвідування занять на базі використання штрих-кодів студентських квитків.

Методи дослідження базуються на використанні алгоритмів локалізації та декодування штрих-кодів заснованих на морфологічних операціях, black top-hat, методу Телкіна і Кафлана [5-7], а також організації сховища даних засобами Firebase.

Сучасні зчитувачі штрих-кодів на основі камери не завжди працюють належним чином, якщо зображення має низьку роздільну здатність, не у фокусі або розмиті рухом. Однією з головних причин є те, що практично всі існуючі алгоритми виконують певний вид бінаризації, або за допомогою порогових значень відтінків сірого, або шляхом пошуку країв смуги. ***Новизна*** полягає у модифікації існуючих підходів до процесу розпізнавання штрих-кодів з метою покращення точності та швидкості розпізнавання завдяки обмеженій бінаризації зображення [8].

Практична значущість отриманих результатів полягає в розробці математичної моделі та програмного забезпечення з метою вдосконалення процесу обліку здобувачів освіти, що дозволить спростити контроль за присутністю їх на заняттях та полегшити формування відповідної звітності викладачам та адміністрації закладу вищої освіти.

Публікації

1. Юхта О.А., Ройко О.Ю. Алгоритмічне забезпечення розпізнавання штрих-кодів на базі камери. Науковий журнал «Комп’ютерно-інтегровані технології: освіта, наука, виробництво», Луцьк: ЛНТУ, 2022. Випуск №48. С.124-128.

РОЗДІЛ 1
АНАЛІЗ МЕТОДІВ ТА ПІДХОДІВ ДО КОДУВАННЯ ІНФОРМАЦІЇ

1.1. Теоретичні аспекти представлення та кодування інформації

Теорія кодування є одним із найважливіших і прямих застосувань теорії інформації. Її можна розділити на теорію кодування джерела та теорію канального кодування. Використовуючи статистичний опис даних, теорія інформації кількісно визначає кількість бітів, необхідних для опису даних, що є інформаційною ентропією джерела.

Теорія кодування – це вивчення властивостей кодів та їх відповідної придатності для конкретних застосувань. Коди використовуються для стиснення даних, криптографії, виявлення та виправлення помилок, передачі та зберігання даних. Коди вивчаються різними науковими дисциплінами – теорією інформації, електротехнікою, математикою, лінгвістикою та інформатикою. Метою вивчення є розробка ефективних і надійних методів передачі даних. Зазвичай це передбачає видалення надлишковості та виправлення або виявлення помилок у переданих даних [9].

Код – це система правил для перетворення інформації в іншу форму, іноді скорочену або таємну, для передачі через канал зв’язку або зберігання на носії інформації. Такою інформацією для перетворення може бути літера, слово, набір символів, звук, зображення або жест. Раннім прикладом щодо використання кодів є винахід мови, який дозволяв людині за допомогою неї повідомляти іншим те, що вона думала, бачила, чула або відчувала. Але мова обмежує діапазон спілкування відстанню, яку може передати голос і обмежує аудиторію лише присутніми під час промови. Винахід писемності, який перетворив усну мову на візуальні символи, розширив діапазон спілкування в просторі та часі [10].

Цифрові дані, в теорії інформації та інформаційних системах, – це інформація, представлена у вигляді рядка дискретних символів, кожен із яких може приймати одне зі скінченної кількості значень деякого алфавіту, таких як літери чи цифри. Прикладом є текстовий документ, який складається з рядка

буквено-цифрових символів. Проте, комп'ютер не може зберігати літери, цифри, зображення чи щось інше. Єдине, що він може зберігати та з чим працювати, – це біти. Біт може мати лише два значення: так чи ні, істина чи хиба, 1 чи 0, або будь-яке інше, якими ви бажаєте назвати ці два значення. Оскільки комп'ютер працює з електрикою, фактичний біт – це проблиск електрики, сигнал, який або є, або його немає. Для людей це зазвичай представлено за допомогою 1 і 0 відповідно. Таке представлення у вигляді двійкових даних через двійкові цифри (біти) є найбільш поширеною формою цифрових даних у сучасних інформаційних системах.

Оскільки символи (наприклад, буквено-цифрові) не є неперервними, цифрове представлення символів набагато простіше, ніж перетворення неперервної або аналогової інформації в цифрову. Замість дискретизації та квантування, як при аналого-цифровому перетворенні, використовуються такі методи, як опитування та кодування [11-15].

У теорії інформації та інформатиці код зазвичай розглядається як алгоритм, який однозначно представляє символи з деякого вихідного алфавіту за допомогою закодованих рядків, які можуть бути в іншому цільовому алфавіті. Розширення коду для представлення послідовностей символів над вихідним алфавітом отримують шляхом конкатенації закодованих рядків.

Для прикладу, відображення $C = \{a \mapsto 0, b \mapsto 01, c \mapsto 011\}$ – це код, вихідним алфавітом якого є набір $\{a, b, c\}$, а цільовим – набір $\{0,1\}$. Використовуючи розширення коду, закодований рядок 0011001 можна згрупувати в кодові слова як $0\ \ 011\ \ 0\ \ 01$, а їх, у свою чергу, можна декодувати до послідовності вихідних символів $acab$.

Отже, процес кодування перетворює інформацію з джерела в символи для передачі або зберігання. Декодування є зворотнім процесом, перетворюючи кодові символи назад у форму, яку розуміє одержувач.

Використовуючи терміни з теорії формальної мови, точне математичне визначення цього поняття виглядає наступним чином: нехай S і T є двома кінцевими наборами, які називаються вихідним і цільовим алфавітами

відповідно. Код $C: S \to T^*$ є сумарною функцією, що відображає кожен символ із S на послідовність символів над T. Розширення C' для C є гомоморфізмом S^* на T^*, який природним чином відображає кожну послідовність вихідних символів на послідовність цільових символів [10].

Кодування використовується для перетворення даних, щоб дані могли підтримуватися та використовуватися різними системами. Кодування працює подібно до перетворення температури за Цельсієм у Фаренгейт, оскільки вона просто перетворюється в іншу форму, але вихідне значення завжди залишається незмінним. Кодування використовується в основному в двох областях:

1. Кодування в електроніці: в електроніці кодування стосується перетворення аналогових сигналів у цифрові.

2. Кодування в обчислювальній техніці: в обчислювальній техніці кодування – це процес перетворення даних в еквівалентний шифр шляхом застосування до даних певного коду, літер і цифр.

Кодування символів кодує символи в байти. Воно інформує комп’ютер про те, як інтерпретувати нулі та одиниці в справжні символи, числа та символи. Комп'ютер розуміє тільки двійкові дані; отже, необхідно перетворити ці символи в цифрові коди. Для цього кожен символ перетворюється на двійковий код, і для цього текстові документи зберігаються з типами кодування. Це можна зробити, поєднавши числа з символами. Якщо кодування символів не застосувати, наприклад, до деякого веб-сайту, то він не відображатиме символи та текст у належному форматі. Таким чином, це зменшить читабельність і машина не зможе правильно обробляти дані. Крім того, кодування символів забезпечує належне представлення кожного символу в комп’ютерному чи двійковому форматі.

Кодування HTML використовується для відображення сторінки HTML у належному форматі. За допомогою кодування веб-браузер дізнається, який набір символів використовувати. У розмітці HTML використовуються різні символи, наприклад <, >. Щоб закодувати ці символи як вміст, нам потрібно використовувати кодування.

Кодування URL використовується для перетворення символів у такий формат, щоб їх можна було передавати через Інтернет. Він також відомий як процентне кодування. Кодування URL-адреси виконується для її надсилання в Інтернет за допомогою набору символів ASCII. Символи, відмінні від ASCII, замінюються на %, за якими йдуть шістнадцяткові цифри.

Unicode – це стандарт кодування для універсального набору символів. Він дозволяє кодувати, представляти та обробляти текст, представлений більшістю мов або систем письма, доступних у всьому світі. Він надає кодову точку або номер для кожного символу в кожній підтримуваній мові. Він може представляти приблизно всі можливі символи в усіх мовах. Певна послідовність бітів відома як одиниця кодування. Стандарт UNICODE може використовувати 8, 16 або 32 біт для представлення символів за допомогою схем UTF-8, UTF-16 та UTF-32 відповідно.

Кодування Base64 використовується для кодування двійкових даних у еквівалентні символи ASCII. Кодування Base64 використовується в системі пошти, оскільки такі поштові системи, як SMTP, не можуть працювати з двійковими даними, адже вони приймають лише текстові дані ASCII. Воно також використовується в простій автентифікації HTTP для кодування облікових даних. Крім того, для передачі двійкових даних у файли cookie та інші параметри, щоб зробити дані нечитабельними для запобігання підробці. Якщо зображення або інший файл передано без кодування Base64, він буде пошкоджений, оскільки поштова система не може працювати з двійковими даними. Base64 представляє дані в блоках по 3 байти, де кожен байт містить 8 біт; отже, він представляє 24 біти. Ці 24 біти розділені на чотири групи по 6 бітів. Кожна з цих груп або фрагментів перетворюється на еквівалентне значення Base64.

Американський стандартний код для обміну інформацією (ASCII) – це тип кодування символів, що був першим стандартом кодування символів, випущеним у 1963 році. Код ASCII використовується для представлення англійських символів у вигляді чисел, де кожній літері присвоюється число з Від 0 до 127. Більшість сучасних схем кодування символів базується на ASCII, хоча

вони підтримують багато додаткових символів. Це однобайтове кодування лише з використанням нижніх 7 бітів. У файлі ASCII кожен алфавітний, цифровий або спеціальний символ представлений 7-розрядним двійковим числом. Кожен символ клавіатури має еквівалентне значення ASCII [13-14].

Зображення, аудіо та відео кодуються для економії місця для зберігання у більш ефективному та стисненому форматі. Ці закодовані файли містять однаковий вміст із зазвичай однаковою якістю, але в стисненому розмірі, тому їх можна зберігати на меншому просторі, легко передавати та завантажувати [13].

1.2. Застосування штрих-кодів для кодування інформації

Традиційні методи введення даних у комп’ютер за допомогою клавіатури мають низку недоліків. Завжди існує ризик запису помилкових даних. Крім того, введення даних відбувається досить повільно, а також потрібна обов'язкова повна та неперервна участь людини у процесі.

Такі недоліки можуть особливо відчуватися при великих обсягах даних, що вводяться. В такому випадку на допомогу приходять автоматичні системи введення даних, такі, як модулі пам'яті, засновані на програмованих постійних запам’ятовувачах, магнітні карти, радіомітки, системи відео-розпізнавання символів, штрих-коди і т.д [15].

Останні десятиліття велике поширення набула технологія штрихового кодування. Це пов’язано з надійністю, компактністю і дешевизною символу, що наноситься, з можливістю вибору обладнання для роботи з ним. Штрихове кодування широко застосовується, наприклад, у роздрібній торгівлі, поштових та вантажних перевезеннях, для ідентифікації, трекінгу у різноманітних сферах, а також у квитках та навіть документах [16].

Штрих-код – це спосіб представлення даних у візуальному машиночитаному форматі. Спочатку штрих-коди представляли дані шляхом зміни ширини, відстані та розміру паралельних ліній. Ці штрих-коди, які зараз зазвичай називають лінійними або одновимірними (1D), можна сканувати спеціальними оптичними сканерами, які називаються зчитувачами штрих-кодів,

яких існує декілька типів. Пізніше були розроблені двовимірні (2D) варіанти з використанням прямокутників, крапок, шестикутників та інших візерунків, які називаються матричними кодами або 2D штрих-кодами, хоча вони не використовують смуги як такі. 2D штрих-коди можна зчитувати за допомогою спеціально розроблених 2D оптичних сканерів, які існують у кількох різних формах. Двовимірні штрих-коди також можна зчитувати цифровою камерою, підключеною до мікрокомп'ютера, на якому працює програмне забезпечення, яке робить фотографічне зображення штрих-коду та аналізує зображення, щоб деконструювати та декодувати двовимірний штрих-код. Мобільний пристрій із вбудованою камерою, як-от смартфон, може забезпечувати зчитування штрих-кодів обох типів.

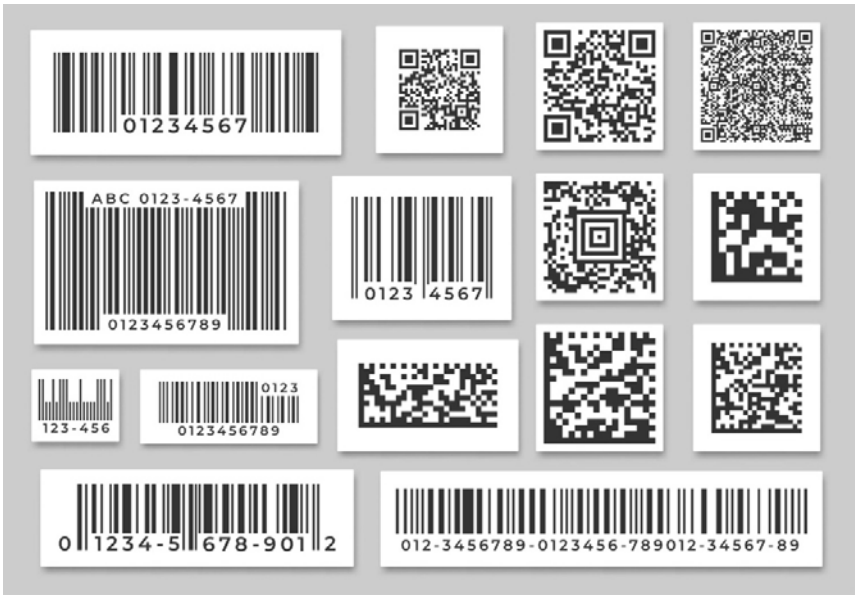

Рис.1.1. Різновиди штрих-кодів

Перший штрих-код із дизайном, схожим на яблучко, був винайдений у 1948 році двома студентами Університету Дрекселя – Норманом Дж. Вудлендом і Бернардом Сільвером. Вони були зацікавлені у вирішенні проблем супермаркетів, які вкрай потребували кращого методу управління власними запасами та перевірки клієнтів. Штрих-код був запатентований у США в 1952 році. Винахід був заснований на азбуці Морзе, яка була поширена на тонкі та товсті смуги, з яких і складаються штрих-коди.

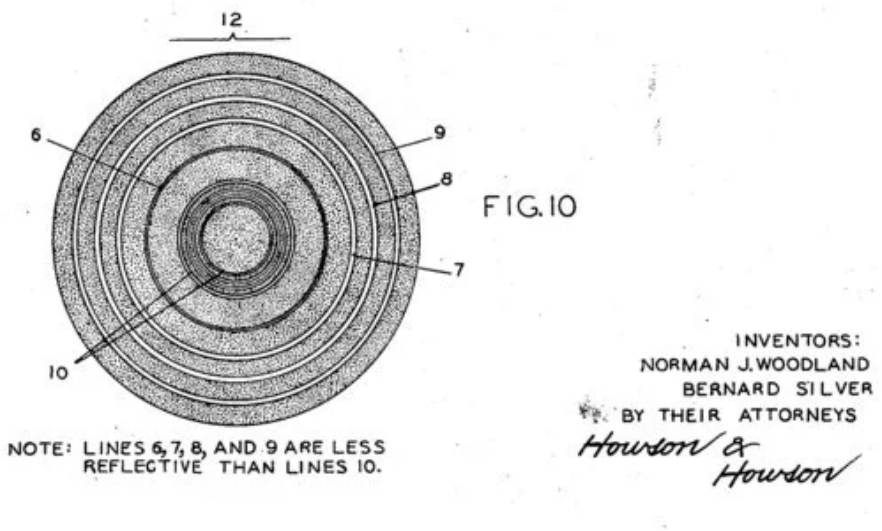

Рис.1.2. Перший запатентований штрих-код

Проте, минуло понад двадцять років, перш ніж цей винахід став комерційно успішним. Штрих-кодами користувались в лабораторіях, але в принципі це було надзвичайно непрактично через обмеження тогочасних технологій. Але у 1960-х роках з’явилася перша практична реалізація лінійного штрих-коду. Асоціація американських залізниць виступила спонсором проекту, і Sylvania (Нідхем, штат Массачусетс) побудувала систему KarTrak ACI для автоматичної ідентифікації вагонів. Ця схема передбачала розміщення кольорових смуг у різних комбінаціях на сталевих пластинах, які були прикріплені до боків залізничного рухомого складу. Для кожного вагону використовувалися два номерні знаки, по одному з кожного боку, з розташуванням кольорових смуг, що кодували інформацію про право власності, тип обладнання та ідентифікаційний номер. Номерні знаки зчитував сканер біля колії, розташований, наприклад, на в'їзді на класифікаційний майданчик, коли вагон проїжджав повз. Сканер штрих-кодів KarTrak був машиною розміром з холодильник. Він активувався при наближенні поїзда, освітлював вагони світлом потужністю 500 Вт, а датчик інтерпретував відбиття. Дані роздруковувалися на телетайпі, папері чи магнітній стрічці. У деяких випадках дані могли бути записані безпосередньо на комп’ютер. KarTrak випередив трьох інших конкурентів і став стандартом США в 1968 році. Усі північноамериканські залізниці прийняли KarTrak. Але приблизно через десять років проект було

залишено, оскільки система виявилася ненадійною після тривалого використання через низьку точність зчитування, відсутність технічного обслуговування штрих-кодів, високу вартість комп'ютерів і низку банкрутств залізниць.

Штрих-коди стали комерційно успішними, коли їх почали використовувати для автоматизації касових систем супермаркетів. Технологічна гонка розгорнулася між RCA та IBM протягом 1970-х років.

RCA придбала оригінальний патент на штрих-код, але IBM зрештою виграла гонку винаходом лінійного штрих-коду UPC. Штрих-код Лорера з вертикальними смугами друкувався краще, ніж круглий штрих-код, розроблений Вудлендом та Сільвером. Їх використання поширилося на багато інших завдань, які загалом називають автоматичною ідентифікацією та збором даних (AIDC), а вперше сканування нового штрих-коду відбулося у супермаркеті у червні 1974 року.

Відповідність між повідомленнями та штрих-кодами називається символікою. Специфікація символіки включає кодування повідомлення в смуги та пробіли між смугами, будь-які необхідні початкові та кінцеві маркери, розмір тихої зони, яка повинна бути перед і після штрих-коду, і обчислення контрольної суми.

Лінійні символіки можна класифікувати головним чином за двома властивостями:

1. Роздільність (дискретна проти неперервної).

Символи в дискретній символіці складаються з n смуг і $n-1$ пробілів. Між символами є додатковий пробіл, але він не передає інформації та може мати будь-яку ширину, якщо його не плутати з кінцем коду. Символи в неперервній символіці складаються з n штрихів і n пробілів і зазвичай стикаються один з одним, причому один символ закінчується пробілом, а наступний починається смугою, або навпаки. Для завершення коду потрібен спеціальний кінцевий шаблон, який має смуги на обох кінцях.

2. Розмірність (двохширинна проти багатоширинної).

Двохширинний штрих-код, який також називають двійковим штрих-кодом, містить смуги та пробіли двох ширин – широкі та вузькі. Точна ширина широких смуг і проміжків не є критичною; як правило, дозволено бути десь у 2-3 рази більшою за ширину вузьких еквівалентів. Деякі інші символіки використовують смуги двох різних висот, або факт наявності чи відсутності смуг. Зазвичай вони також вважаються двійковими штрих-кодами. Смуги та пробіли в багатоширинній символіці є кратними базової ширини, яка називається модулем. Більшість таких штрих-кодів використовують чотири ширини з 1, 2, 3 і 4 модулів [17-19].

Найбільш поширеними серед багатьох двовимірних символік є матричні коди, які містять квадратні або точкові модулі, розташовані на сітці. Двовимірні символіки також мають круглі та інші форми, а також можуть використовувати стеганографію, приховуючи модулі всередині зображення.

Намагаючись вибрати доцільний штрих-код для використання, потрібно враховувати деякі фактори. Перш за все, слід ознайомитися з різними типами штрих-кодів, їхніми можливостями та вимогами, щоб забезпечити правильний вибір штрих-коду для конкретного сценарію. Щоб здійснити вибір конкретного типу штрих-коду для поставленої задачі, потрібно зрозуміти його набір символів. Загалом існує чотири різні типи наборів символів:

1. Числовий – набір числових символів включає лише цифри (0-9).
2. Алфавітно-цифровий – алфавітно-цифровий набір символів включає цифри та букви (0-9 та A-Z).
3. Набір кодованих символів GS1 AI 82 □ включає цифри (0-9), букви (A-Z) і спеціальні символи, які можна знайти в стандарті.
4. Повний ASCII – набір символів системи кодів ASCII, що включає будь-який символ таблиці ASCII (значення 0-127).

Що стосується даного дослідження, то символіка буде визначена як представлення будь-якого символу ASCII послідовністю 0 та 1. Щоб перетворити його на зображення штрих-коду, 0 та 1 замінюються білими та чорними модулями відповідно певного фіксованого розміру.

Можливості штрих-кодів варіюються від використання десятків символів для кодування за допомогою конкретного 1D штрих-коду до тисяч символів у 2D штрих-кодах. Відповідно до цього, в основному існує два таких типи штрих-кодів:

1. Одновимірні (1D, лінійні) штрих-коди, що складаються з вертикальних чорних ліній на повністю білому тлі. Лінії мають різну ширину з певними проміжками, що у комплексі призводить до формування певного візерунка зі смуг та пробілів.

2. Двовимірні (2D) штрих-коди, що зазвичай складаються з квадратних, прямокутних або інших форм візерунків двох вимірів. Знову ж таки, візерунки найчастіше чорні на повністю білому тлі.

Як правило, 2D штрих-коди зберігають більше даних і підтримують більший набір символів, ніж 1D штрих-коди. Однак більше не завжди означає краще. Вибір штрих-коду має ґрунтуватися на ефективності максимізації набору символів із невеликим простором для розширення, а не на подвоєнні чи потроєнні набору символів, який ви можете використовувати. Інакше це може вплинути на швидкість обробки [20].

Детальніше конкретні типи штрих-кодів, що відносяться до категорій одновимірних та двовимірних, будуть розглянуті у наступному пункті.

1.3. Класифікація штрих-кодів, аналіз принципів побудови їх різновидів

Штрих-код – це візерунок із паралельних ліній різної ширини. Вони використовуються для вбудовування певних даних, які можуть бути зчитані сканером штрих-кодів, який потім декодує інформацію за допомогою алгоритму. Візерунки та лінії, які утворюють штрих-код, являють собою певні символи, які можна використовувати для зберігання певної інформації, такої як тип продукту, партія продукту та інше. Хоча штрих-коди існують уже кілька десятиліть, у цьому просторі відбулося багато інновацій. Сьогодні існують одновимірні штрих-коди, а також двовимірні штрих-коди кількох різних типів. Одновимірні

штрих-коди можуть упакувати обмежену кількість інформації, а двовимірні штрих-коди – до кількох тисяч символів інформації. Отже, штрих-код – це машиночитане представлення цифр і символів.

Хоча єдиного стандартизованого формату штрих-кодів не існує, є кілька важливих специфікацій, які визначають, як вони розроблені та створені. Етикетки зі штрих-кодом можуть відрізнятися за розміром, місткістю, лінійністю, матеріалом, а також за тим, чи потрібна контрольна сума. Розмір етикетки часто визначається специфікаціями скануючого обладнання та передбачуваним застосуванням. У деяких випадках може знадобитися певна орієнтація етикетки щодо розміщення скануючого обладнання чи іншого технологічного обладнання.

Лінійність штрих-коду вимірює довжину сканованої області та є функцією ємності штрих-коду залежно від конкретного стилю. Місткість штрих-коду, тобто кількість потенційних комбінацій символів, залежить від щільності штрих-коду та набору символів, який підтримується. Одне вимірювання щільності штрих-коду називається x-розмірністю, а у випадку лінійного штрих-коду – шириною найвужчої смуги. Для двовимірних штрих-кодів x-розмірність – це розмір кожного його квадрата.

Штрих-коди можуть мати фіксовану або змінну довжину. У кодах фіксованої довжини стандарт визначає, скільки символів представлено в коді, тоді як інший тип може кодувати довільну кількість символів.

Контрольна сума – це стандартизована частина деяких форматів штрих-кодів, яка використовується для перевірки правильності інформації, відсканованої з коду. У лінійній конфігурації це завжди число в дальньому правому куті штрих-коду, і сканер виконає серію обчислень над цифрами, які передують йому, а тоді порівнює цей результат з останньою цифрою. Якщо сума правильна, сканер часто подаватиме звуковий сигнал, щоб переконатися, що сканування відбулося правильно. Знання цих схожостей та відмінностей між форматами штрих-кодів може допомогти вибрати найбільш оптимальні типи штрих-кодів для вирішення конкретних задач, потреб та цілей [21].

Одновимірні (лінійні) штрих-коди можна умовно поділити на дві підкатегорії – цифрові (числові) та алфавітно-цифрові. Перші складаються лише з цифр, а другі можуть містити комбінацію цифр і літер (букв). Сьогодні існує близько 30 основних форматів штрих-кодів, які широко використовуються на основі лінійних і двовимірних типів. Кожен із цих основних форматів знайшов застосування в окремих додатках, які можуть використовувати їхні унікальні якості та властивості. Розглянемо детальніше найбільш популярні та часто застосовувані види штрих-кодів, а також їх основні характеристики.

Суто цифрові штрих-коди – це одновимірні штрих-коди, які кодують тільки числа. Одновимірні або 1D штрих-коди систематично представляють дані, змінюючи ширину та відстань між паралельними лініями, і їх можна називати лінійними або одновимірними. До них належать деякі з традиційних та найбільш добре розпізнаних типів штрих-кодів, таких як типи кодів UPC і EAN. Існує більше десятка різних типів лише числових символік штрих-кодів.

Довжина одновимірного штрих-коду безпосередньо залежить від того, скільки інформації він містить. Відповідно, користувачі повинні обмежити кількість символів, які містить кожен код, – від 8 до 15.

Коди UPC – одна із найпоширеніших символік штрих-кодів і, можливо, найбільш впізнаваний тип штрих-кодів споживачами через широке використання кодів UPC у роздрібних магазинах – переважно в США, Великобританії, Австралії, Новій Зеландії та деяких інших країнах. 12-значні коди UPC (UPC-A) містять основну інформацію про особу виробника та ідентифікаційний номер продукту. Позиція кожної цифри вказує на тип інформації, до якої відносяться ці числа. Це стандартизований процес, який дає змогу розшифровувати UPC-коди, які не належать до певної компанії. Існують також базові варіації UPC-E, які містять лише 6 цифр. Абревіатура UPC фактично означає універсальний код продукту. У контексті роздрібної торгівлі призначення цього штрих-коду полягає в тому, щоб полегшити користувачам ідентифікацію конкретних характеристик продукту (наприклад, його розмір або колір), коли товар сканується під час оформлення замовлення та на касі. Окрім

підвищення ефективності процесу оформлення замовлення, UPC-коди допомагають оптимізувати відстеження запасів у магазинах і на складах. Ці штрих-коди забезпечують точне та ефективне відстеження продукту на всьому шляху від виробництва до розповсюдження.

Рис.1.3. Штрих-код UPC-A

Структура коду UPC-A:

1. Початкова тиха зона.
2. Захисний шаблон (початковий символ).
3. Шість знаків символів.
4. Центральний захисний шаблон (центральний символ).
5. Шість знаків символів, включаючи контрольну цифру.
6. Захисний шаблон (стоп-символ).
7. Кінцева тиха зона.

Щільність: середньої щільності. UPC друкує смуги та пробіли чотирьох різних розмірів і займає половину простору коду Interleaved 2 of 5.

Набір символів: числовий.

Місткість: 12 символів.

Виявлення помилок: має контрольну цифру за модулем 10 для виявлення помилки.

Виправлення помилок: UPC не підтримує виправлення помилок.

Використання: унікально ідентифікує продукт для роздрібного продажу.

Рис.1.4. Штрих-код UPC-E

Структура коду UPC-E:

1. Початкова тиха зона.

2. Захисний шаблон (початковий символ).

3. Шість знаків символів.

4. Захисний шаблон (стоп-символ).

5. Кінцева тиха зона.

Щільність: середньої щільності. UPC друкує смуги та пробіли чотирьох різних розмірів і займає половину простору коду Interleaved 2 of 5.

Набір символів: числовий.

Місткість: 12 символів, нулі відсутні.

Виявлення помилок: має контрольну цифру за модулем 10 для виявлення помилки.

Виправлення помилок: UPC не підтримує виправлення помилок.

Використання: дозволяє використовувати штрих-коди UPC на менших упаковках, де код UPC-A не поміщається.

Коди EAN – використовуються для маркування споживчих товарів у всьому світі для сканування в місцях продажу, переважно в Європі. Вони дуже схожі на коди UPC, головною відмінністю є їхнє географічне застосування. Хоча EAN-13 (що містить 13 цифр) є форматом за замовчуванням, для продуктів, де простір обмежений, як-от маленькі цукерки, використовується штрих-код типу EAN-8 (містить 8 цифр). Головною перевагою кодів EAN є їх гнучкість. EAN-13

– це штрих-код високої щільності, який може кодувати відносно великі обсяги даних у невеликій області, тоді як коди EAN-8 ідеально підходять для ідентифікації дуже маленьких продуктів або активів. Коди EAN легко зчитуються 1D-сканерами, що робить процес сканування швидким і безперебійним.

Структура коду EAN-13: ідентична до UPC-A.

Щільність: середньої щільності. EAN друкує смуги та пробіли чотирьох різних розмірів і займає половину простору коду Interleaved 2 of 5.

Набір символів: числовий.

Місткість: 13 символів.

Виявлення помилок: має контрольну цифру за модулем 10 для виявлення помилки.

Виправлення помилок: EAN не підтримує виправлення помилок.

Використання: для маркування продуктів, що продаються в роздрібних магазинах або під час розповсюдження.

Рис.1.5. Штрих-код EAN-8

Структура коду EAN-8:

1. Початкова тиха зона.
2. Захисний шаблон (початковий символ).
3. Чотири знаки символів.
4. Центральний захисний шаблон (центральний символ).
5. Чотири знаки символів, включаючи контрольну цифру.
6. Захисний шаблон (стоп-символ).

7. Кінцева тиха зона.

Щільність: середньої щільності. EAN друкує смуги та пробіли чотирьох різних розмірів і займає половину простору коду Interleaved 2 of 5.

Набір символів: числовий.

Місткість: 8 символів.

Виявлення помилок: має контрольну цифру за модулем 10 для виявлення помилки.

Виправлення помилок: EAN не підтримує виправлення помилок.

Використання: на невеликих упаковках, коли код EAN-13 завеликий.

Коди ITF (Interleaved 2 of 5) – використовуються для маркування пакувальних матеріалів у всьому світі. Оскільки вони витримують високі допуски друку, вони ідеальні для друку на гофрокартоні. Штрих-коди ITF кодують 14 цифрових символів і використовують повний набір таблиці ASCII. Хоча штрих-код Interleaved 2 of 5 може кодувати лише цифри, але не літери, для нього не потрібна контрольна цифра.

Рис.1.6. Штрих-код ITF

Структура коду ITF:

1. Початкова тиха зона.

2. Початковий шаблон (вузька смуга, вузький пробіл, вузька смуга, вузький пробіл).

3. Одна або кілька пар символів, які представляють дані (включаючи необов’язкову контрольну цифру).

4. Стоп-шаблон (широка смуга, вузький пробіл, вузька смуга).

5. Кінцева тиха зона.

Щільність: висока, кодує дані як по ширині смуг, так і пробілів.

Набір символів: числовий.

Місткість: 14 символів.

Виявлення помилок: має контрольну цифру за модулем 10 для виявлення помилки.

Виправлення помилок: ITF не підтримує виправлення помилок.

Використання: зазвичай використовується для об’ємної упаковки, такої як ящик або картонна коробка.

Коди Codabar – використовуються спеціалістами з логістики та охорони здоров’я, фотолабораторіями та бібліотеками. Їх головна перевага полягає в тому, що коди легко друкувати на будь-якому принтері – навіть на друкарській машинці. Таким чином, користувачі можуть створювати багато кодів Codabar, використовуючи послідовні номери без використання комп’ютера. Codabar – це окрема символіка з самоперевіркою, яка кодує до 16 різних символів із додатковими 4 символами початку/зупинки. До переваг також можна віднести легкість сканування та самоперевірки, що зменшує кількість помилок при введенні коду. Однак коди Codabar поступово втрачають позиції на користь нових форм коду, які дозволяють зберігати більше даних у набагато меншій формі. Незважаючи на це, Codabar все ще широко використовується в логістиці, охороні здоров’я та навіть у школах, де код наноситься на корінці бібліотечних книг.

Рис.1.7. Штрих-код Codabar

Структура коду Codabar:

1. Початкова тиха зона.

2. Початковий символ.

3. Символи, які представляють дані (включаючи необов’язкову контрольну цифру).

4. Стоп-символ.

5. Кінцева тиха зона.

Щільність: низька.

Набір символів: числовий, також підтримуються символи двокрапка (:), коса риска (/), крапка (.), знак плюс (+).

Місткість: необмежена (зазвичай до 20 символів).

Виявлення помилок: це код самовизначення, більшість стандартів не визначають контрольну цифру.

Виправлення помилок: Codabar не підтримує виправлення помилок.

Використання: зазвичай використовуються фахівцями з логістики та охорони здоров’я, зокрема банками крові США, фотолабораторіями, бібліотеками тощо.

Що стосується алфавітно-цифрових одновимірних штрих-кодів, то їх найбільш популярними представниками є Code 39, Code 128 та Code 93.

Коди Code 39 – використовуються для маркування товарів у багатьох галузях промисловості та часто зустрічаються в автомобільній промисловості та Міністерстві оборони США. Це був перший розроблений буквено-цифровий код. Він дозволяє використовувати як цифри, так і символи, а його назва походить від того, що він міг кодувати лише 39 символів, хоча в останній версії набір символів було збільшено до 43. Він схожий на штрих-код Code 128, але не такий компактний.

Рис.1.8. Штрих-код Code 39

Структура коду Code 39:

1. Початкова тиха зона.

2. Початковий символ (зазвичай це зірочка *).

3. Одна або кілька пар символів, які представляють дані (включаючи необов’язкову контрольну цифру).

4. Стоп-символ (зазвичай це зірочка *).

5. Кінцева тиха зона.

6. Міжсимвольні пробіли шириною в один модуль для розділення символів.

Щільність: помірна.

Набір символів: буквено-цифровий. Код 39 також підтримує такі символи, як знак відсотка (%), знак плюса (+), знак долара ($), похилу риску (/), крапку (.), дефіс (-).

Місткість: необмежена (зазвичай від 20 до 23 символів).

Виявлення помилок: це код самовизначення, контрольна цифра зазвичай не потрібна.

Виправлення помилок: Code 39 не підтримує виправлення помилок.

Використання: все ще широко використовується, особливо в нероздрібному середовищі.

Коди Code 128 – це компактні коди високої щільності, які використовуються в логістиці та транспортній галузі для розміщення замовлень та дистрибуції. Вони орієнтовані на продукти, не призначені для точок продажу. Оскільки вони підтримують будь-який символ набору символів ASCII 128, штрих-коди Code 128 можуть зберігати найрізноманітнішу інформацію. Ці штрих-коди можуть зберігати великі обсяги лінійних даних у компактній формі.

Рис.1.9. Штрих-код Code 128

Структура коду Code 128:

1. Початкова тиха зона.

2. Початковий символ.

3. Символи, які представляють дані.

4. Символ для перевірки.

5. Стоп-символ.

6. Кінцева тиха зона.

Щільність: висока.

Набір символів: вся таблиця ASCII.

Місткість: необмежена (зазвичай до 48 символів).

Виявлення помилок: має контрольне значення за модулем 103 для виявлення помилки.

Виправлення помилок: Code 128 має обов'язковий символ виправлення помилок.

Використання: може містити такі дані, як термін придатності, номер партії, продукту в партії, кількість, вага та багато інших атрибутів.

Коди Code 93 – використовуються в логістиці для ідентифікації пакунків у роздрібній торгівлі, маркування електронних компонентів і навіть надання додаткової інформації про доставку для пошти Канади. Як і Code 39, штрих-коди Code 93 мають повну підтримку ASCII. Code 93 діє як більш компактна та безпечна альтернатива Code 39, частково завдяки додатковим символам. Його невеликий розмір і резервування даних роблять його ідеальним для використання в багатьох галузях промисловості.

Рис.1.10. Штрих-код Code 93

Структура коду Code 93:

1. Початкова тиха зона.

2. Початковий символ.

3. Вхідні символи, що представляють дані.

4. Дві контрольні цифри.

5. Стоп-символ.

6. Кінцева тиха зона.

Щільність: помірна.

Набір символів: буквено-цифровий. Код 39 також підтримує такі символи, як знак відсотка (%), знак плюса (+), знак долара ($), похилу риску (/), крапку (.), дефіс (-).

Місткість: необмежена (зазвичай від 20 до 23 символів).

Виявлення помилок: має контрольну цифру за модулем 47 "C" і контрольний символ за модулем 47 "K" для виявлення помилки.

Виправлення помилок: Code 93 не підтримує виправлення помилок.

Використання: поштові сервіси, роздрібна торгівля, виробництво та логістика.

Символіки двовимірних штрих-кодів, також відомі як 2D штрих-коди, – це графічні зображення, які зберігають інформацію як у горизонтальній, так і у вертикальній площинах. Завдяки цій конструкції двовимірні штрих-коди можуть кодувати до 7089 символів – це значно більше, ніж може кодувати будь-який одновимірний штрих-код. Двовимірні штрих-коди пропонують можливість зберігати більше даних в одному коді та призводять до меншої кількості помилок при розшифровці кодів завдяки можливості створення резервів або механізмів самоперевірки, що стало можливим завдяки більшій ємності 2D-кодів для зберігання інформації. QR-коди є одним з таких типів штрих-коду.

2D штрих-коди часто використовуються в поєднанні зі смартфонами. Користувач просто фотографує або сканує 2D штрих-код за допомогою камери на телефоні, оснащеному зчитувачем 2D штрих-кодів, що також полегшує їх точну передачу за допомогою сервісів обміну повідомленнями. Крім того,

двовимірні штрих-коди більш безпечні, оскільки інформацію, що зберігається в такому форматі легко зашифрувати. Існує кілька типів двовимірних символік штрих-кодів, найбільш популярні з них – QR-коди, Data Matrix, Aztec Code, PDF 417.

QR-коди – означають код швидкого реагування та найчастіше використовуються для трекінгу та маркетингових ініціатив, таких як реклама, журнали та візитні картки. Вони гнучкі за розміром, пропонують високу відмовостійкість і швидко читаються, хоча їх не можна прочитати за допомогою лазерного сканера. QR-коди підтримують чотири різні режими даних: числові, буквено-цифрові, байтові/двійкові та кандзі. Вони є суспільним надбанням і вільні для використання.

Рис.1.11. QR-код

Структура QR-коду:

1. Квадратні модулі в звичайному квадратному масиві, які містять дані.
2. Візерунки пошуку.
3. Смуги синхронізації.
4. Візерунки вирівнювання.
5. Границя тихої зони.

Щільність: висока.

Набір символів: вся таблиця ASCII.

Місткість: 7089 цифрових або 4296 буквено-цифрових символів.

Виявлення помилок: використовує алгоритм Ріда-Соломона, щоб додати інформацію про виявлення помилок і виправлення до вихідних даних.

Виправлення помилок: використовує алгоритм Ріда-Соломона для виправлення помилок. Доступні чотири рівні виправлення помилок, що дозволяє зчитувати штрих-код, навіть якщо пошкоджено до 30 відсотків коду.

Використання: платежі в додатках і онлайн, доповнена реальність, мобільні операційні системи, вхід на веб-сайти та сервіси, програми лояльності, віртуальні магазини, маркетингові та рекламні кампанії тощо.

Коди Datamatrix – використовуються для маркування невеликих предметів, товарів і документів. Завдяки своїй мініатюрній площі вони ідеально підходять для невеликих продуктів у логістиці та операційній сфері. Як і QR-коди, вони мають високу відмовостійкість і швидку читаність. Вони також розроблені таким чином, щоб їх можна було читати навіть у низькій роздільній здатності або з неідеальним положенням сканування.

Рис.1.12. Штрих-код Datamatrix

Структура коду Datamatrix:

1. Області даних, які містять квадратні модулі в масиві.

2. Візерунки вирівнювання (для більших символів).

3. Візерунки пошуку.

4. Границя тихої зони.

Щільність: висока.

Набір символів: вся таблиця ASCII.

Місткість: з усіх розмірів 144 x 144 має найбільшу ємність, яка теоретично може вмістити до 3116 цифр, 2335 буквено-цифрових символів або 1556 байт.

Виявлення помилок: використовує згорткове виправлення помилок і лише опційно виявлення помилок.

Виправлення помилок: використовує алгоритм Ріда-Соломона для виправлення помилок.

Використання: маркування невеликих предметів, таких як невеликі електронні компоненти та флакони з таблетками, часто зустрічається на упаковці, плакатах і на виробничих деталях.

Коди PDF417 – використовуються для застосувань, які вимагають зберігання величезних обсягів даних, таких як фотографії, відбитки пальців і підписи. Вони можуть зберігати понад 1,1 кілобайт машинозчитуваних даних, що робить їх набагато потужнішими, ніж інші двовимірні штрих-коди. Як і QR-коди, штрих-коди PDF417 є суспільним надбанням і вільні для використання. Завдяки ефективності даних коди PDF417 підходять для широкого спектру застосувань.

Рис.1.13. Штрих-код PDF417

Структура коду PDF417:

1. Початкова тиха зона.
2. Початковий шаблон.
3. Кодові слова індикатора лівого рядка.
4. Кодові слова даних.
5. Кодові слова індикатора правого рядка.
6. Стоп-шаблон.
7. Кінцева тиха зона.

Щільність: висока.

Набір символів: вся таблиця ASCII.

Місткість: 1850 буквено-цифрових символів, 2710 цифр, або 1108 байт.

Виявлення помилок: використовує алгоритм Ріда-Соломона, щоб додати інформацію про виявлення помилок і виправлення до вихідних даних.

Виправлення помилок: використовує алгоритм Ріда-Соломона для виправлення помилок. Доступні чотири рівні виправлення помилок, що дозволяє зчитувати штрих-код, навіть якщо пошкоджено до 50 відсотків коду.

Використання: є одним із найпоширеніших 2D штрих-кодів, найчастіше зустрічається в логістиці, транспорті (посадкові талони), державній ідентифікації (водійські права та ідентифікаційні картки), інвентаризації та документообігу (поштові посилки).

Коди Aztec – використовуються в галузі перевезень пасажирів. Штрих-коди все ще можна розшифрувати, навіть якщо вони мають погану роздільну здатність, що робить їх корисними, коли квитки погано надруковані або представлені на телефоні. Вони також можуть займати менше місця, оскільки їм не потрібна порожня тиха зона навколо, на відміну від деяких інших типів двовимірних штрих-кодів. Штрих-коди Aztec надзвичайно компактні. Вони можуть зберігати величезні обсяги даних, зберігаючи відносно невеликий розмір, і мають чудову функцію корекції помилок для запобігання помилкам сканування.

Рис.1.14. Штрих-код Aztec

Структура коду Aztec:

1. Візерунок пошуку.

2. Паттерни орієнтації.

3. Повідомлення про режим.

4. Опорна сітка.

5. Шари даних.

Щільність: висока.

Набір символів: вся таблиця ASCII.

Місткість: 3067 буквено-цифрових символів, 3832 цифр, або 1914 байт.

Виявлення помилок: використовує алгоритм Ріда-Соломона, щоб додати інформацію про виявлення помилок і виправлення до вихідних даних.

Виправлення помилок: використовує алгоритм Ріда-Соломона для виправлення помилок. Користувач може вказати відсоток області даних, яка буде використовуватися кодовими словами виправлення помилок; рекомендований рівень (який також є мінімальним за замовчуванням) становить 23 відсотки символьної ємності плюс ще три кодові слова.

Використання: транспортні квитки (залізничні квитки, посадкові талони на літак), реєстрація автомобілів і податкові системи, системи оплати рахунків компаній [22-25].

1.4. Формалізація вимог до програмної системи

Проведений аналіз методів кодування та представлення інформації, зокрема, за допомогою штрих-кодів, які є основною технологією для успішного функціонування розроблюваної програмної системи, дозволяє сформулювати перелік вимог до майбутнього програмного засобу та основних принципових моментів, що повинні мати місце в процесі його проєктування і бути реалізованими в результаті.

1. Програмний засіб повинен мати можливість зчитувати та розшифровувати різноманітні типи штрих-кодів за допомогою камери мобільного пристрою.

2. Програмне забезпечення повинне дозволяти справлятись з задачею зчитування з врахуванням таких умов, як рівень освітлення, ракурс, розміщення

у кадрі, відстань до об’єкта та якість вихідного зображення, що містить штрих-код.

3. Технології зчитування та розшифрування у програмному засобі повинні забезпечувати достатній та адекватний рівень швидкодії для процесу виявлення та обробки штрих-кодів.

4. Для забезпечення доступу до даних та виконання користувачами супутніх дій у програмній системі повинна бути реалізована база даних відповідної архітектури, налаштований зв’язок з нею для доступу до її записів.

Отже, підсумовуючи сформульовані вище вимоги, можна відмітити, що складові частини програмного комплексу, що розроблюється, повинні технологічно забезпечувати процес виявлення штрих-коду на зображенні камери у реальному часі, його захоплення, визначення належності до конкретного типу для подальшого розшифрування і здійснення доступу до даних за зашифрованою в них інформацією. Зчитування та розшифрування необхідні виконуватись коректно та однозначно, оскільки результат цих процесів визначитиме подальші операції над даними завдяки зв’язку з сховищем даних, структура якого повинна бути організована таким чином, щоб не лише підтягувати записи, а й зберігати результати роботи та пов’язаних з нею функцій.

Висновки до розділу 1

1. Здійснено огляд загальних теоретичних аспектів процесу кодування інформації та різноманітних способів кодування.

2. Розглянуто способи представлення даних у машиночитаному форматі та символіки, що забезпечують цей процес.

3. Проаналізовано специфікації символік, що забезпечують представлення даних через використання штрих-кодів.

4. Проведено огляд класифікації штрих-кодів, їх найбільш відомих та часто використовуваних сьогодні типів, визначено принципи їх побудови, схожі та відмінні риси, особливості процесу декодування для подальшої реалізації роботи з ними.

РОЗДІЛ 2
МАТЕМАТИЧНЕ ТА АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ РОЗПІЗНАВАННЯ ШТРИХ-КОДІВ

2.1. Основні проблеми задачі читання штрих-кодів

У загальному процес читання штрих-кодів можна розділити на два глобальних етапи:

1. Знайти розташування штрих-коду.
2. Розшифрувати штрих-код.

Перший етап також називають локалізацією, а другий – читанням або декодуванням [26].

Програмне забезпечення зчитування штрих-кодів сканує всю ширину зображення штрих-коду, щоб ідентифікувати власне сам штрих-код. Після сканування штрих-коду воно декодує його та представляє закодовану інформацію. Крім того, враховуючи широке використання мобільних пристроїв у наш час, камера також використовується для захоплення зображення штрих-коду для обробки та декодування його значення.

Після початку сканування сканер шукає будь-який штрих-код, щоб визначити його тип. Зчитувач штрих-кодів сканує всю ширину зображення, намагаючись визначити, чи є на ньому кандидати на те, щоб бути ідентифікованими у якості штрих-коду, а саме – чорно-білі візерунки. Після ідентифікації штрих-коду програмне забезпечення має області інтересу для виявлення. Потім починається декодування. На цьому кроці сканер штрих-коду намагається зрозуміти закодовану інформацію. Він підраховує та порівнює пікселі зображення на предмет відповідності початковим і кінцевим ідентифікаторам. Потім він інтерпретує візерунки між початковим і кінцевим ідентифікаторами на основі специфікацій цього типу коду, щоб розгадати зашифровані дані.

Проблема локалізації полягає в ідентифікації штрих-коду на захараченому зображенні, адже він може бути повернутий, перекошений, спотворений або на

зображенні може бути кілька штрих-кодів. Зазвичай це робиться шляхом ідентифікації обмежувальної рамки, яка містить штрих-код. Традиційно проблема локалізації вирішується одним із двох методів: на основі просторової області або на основі частотної області [27-31].

Отже, підсумовуючи іншими словами, ми повинні зробити два кроки, щоб відновити дані, закладені у візуальному коді. По-перше, ми знаходимо штрих-код, розпізнаємо його розмір, орієнтацію і, як правило, застосовуємо перетворення, такі як зменшення шуму, підвищення різкості, нормалізацію і виправлення спотворень. Після цього оброблений фрагмент зображення, що містить код, передається детектору, який шукає в ньому дійсні символьні дані. При правильній локалізації та попередній обробці цей етап є відносно простим, оскільки символи легко розпізнаються завдяки максимальним відстаням між символами в більшості випадків. Крім того, більшість патентів на штрих-коди надають додаткові, надлишкові дані для виправлення помилок. Етап локалізації має більше труднощів через різноманітність типів кодів, камер та сценаріїв.

Методи локалізації штрих-коду мають дві конкуруючі цілі: точність виявлення та швидкість. Для промислового середовища точність має вирішальне значення, оскільки невиявлені (пропущені) коди можуть призвести до втрати прибутку. Швидкість обробки є другорядною бажаною властивістю детекторів. На смартфонах точність не настільки критична, оскільки пристрій взаємодіє з користувачем і перезйомка виконується легко, проте бажаним є швидке (і досить точне) виявлення штрих-коду [32].

2.2. Алгоритми виявлення та обробки штрих-кодів

У цьому пункті представляється кілька алгоритмів виявлення штрих-коду, які використовують різні підходи для визначення розташування штрих-коду на зображенні.

Основна ідея локалізації штрих-коду полягає в методі скануючої лінії [33-34]. Вона імітує процес, коли лазерний сканер переміщується по зображенню, скануючи лінії і створюючи одновимірні профілі інтенсивності. Головна ідея

полягає в тому, щоб знайти пікові точки в розмитих моделях штрих-кодів, а потім адаптивно встановити порогове значення профілю інтенсивності для отримання двійкових значень. Частина декодування відбувається на цих опрацьованих профілях для розпізнавання послідовності символів за допомогою різних підходів.

Деякі роботи припускають, що штрих-коди вирівняні по осях, або вони вже локалізовані [35-37]. У цих роботах пропонуються точні алгоритми декодування, які можна поєднати з відповідними підходами до локалізації для вирішення завдання ефективного зчитування штрих-коду. Галло та інші [36] також використовують градієнти, які можна задіяти в тому числі й для локалізації [38].

Алгоритми з морфологією використовують комбінацію основних морфологічних операцій [39-42], таких як ерозія та розширення. Білі плями на оброблених зображеннях показують можливі місця розташування штрих-кодів. Для цих зображень потрібна подальша обробка, як-от сегментація та фільтрація маленьких плям. Їх можна використовувати як для одновимірних, так і для двовимірних штрих-кодів. Ця група вважається найповільнішим методом локалізації штрих-коду, оскільки морфологічні операції зазвичай вимагають згортки, яка є перешкодою при обробці зображень з високою роздільною здатністю.

Кілька робіт пропонують трансформацію Гафа [43-44] для етапу локалізації. У цих дослідженнях пропонується стандартне або імовірнісне перетворення Гафа для виділення ліній або сегментів ліній із зображення та прийняття подальших рішень за довжиною ліній, наближенням одна до одної та орієнтацією. Трансформація потребує карти крайових (контурних) точок як вхідних даних, яка зазвичай створюється за допомогою згортки з використанням ядер Кенні або Собеля [45]. Трансформація крайових точок у простір Гафа також є трудомістким завданням, завдяки чому цей метод можна порівняти за часом виконання з методами, заснованими на морфології.

Туінстра та інші використовують як морфологію, так і перетворення Гафа на етапі локалізації зі смуговим фільтром і пороговим значенням. Вони також

експериментували з градієнтною картою величини та трансформацією навмання, а також трасуванням долин, методом пошуку штрих-кодів у розмитих зображеннях із низькою роздільною здатністю, переважно на кадрах камери смартфона. Він складається з трьох кроків. Спочатку знаходяться початкові точки на малюнках, потім слідуємо за «долинами» і, нарешті, розпізнаємо кінці долин (смуги).

Поділ зображення на однорідні тайли [44;46], що використовуються для створення текстур, також є широко поширеною ідеєю розпізнавання образів, яка теж може бути використана як основа локалізації штрих-коду. Більшість штрих-кодів, як і звичайні текстури, можна легко ідентифікувати, спостерігаючи лише за невеликими їх частинами. Ці частини штрих-коду разом утворюють бажану область штрих-коду з відомою висотою та шириною. Перша частина підходу полягає в розділенні зображення на квадратні плитки та перегляді кожної плитки на наявність штрих-коду. Кожній плитці присвоюється значення, яке вказує на ступінь наявності цієї характеристики. Глобально з цих значень формується матриця. Текстурні частини мають подібну локальну статистику в своєму сусідстві, тому пошук у цій матриці компактних областей подібних значень визначає цікаві області, що представляють штрих-коди з високою ймовірністю. Цей підхід можна використати для швидкої локалізації, однак характеристики зображення мають бути правильно обрані, щоб підтримувати точність на прийнятному рівні.

Розглянемо детальніше, як проявляють себе на практиці алгоритми виявлення штрих-кодів, що відносяться до вище описаних категорій, звернувшись до деяких оригінальних першоджерел досліджень щодо них. Врахуємо також той факт, що у наш час якість цифрових зображень зазвичай дуже хороша, хоча можуть траплятися і матеріали низької якості. Причиною цього може бути пристрій захоплення зображення, а також проблеми з кондиціями навколишнього середовища. Тому часто виникає потреба у виправленні (покращенні) якості зображення перед початком конкретного процесу виявлення штрих-коду.

2.2.1. Метод, заснований на основних морфологічних операціях

В алгоритмі Туінстри [34] автор спирається на те, що в області штрих-коду різниця в інтенсивності між смугами висока, тому градієнт там виділяв би смуги. Ядра Собеля використовуються для оцінки градієнта у напрямках x та y. Потім градієнтне зображення обнулюється порогом, вибираються пікселі, що мають високе значення градієнта. На бінарному зображенні спочатку виконується перетворення навмання з елементом структурування лінії, який не вказано у вихідній статті. Потім виконується морфологічне розширення, щоб об'єднати прилеглі, але не обов'язково пов'язані об'єкти, щоб можна було скласти область. Структуруючий елемент має квадратну форму, але його точний розмір у статті також не був зафіксований. Насправді цей параметр визначається розміром зображення. Розглянемо на прикладі блоку 10x10 стандартного відхилення, який відповідає розміру, використаному на попередньому кроці. Потім слідує морфологічна ерозія, щоб виключити тонкі об'єкти із зображення та видалити небажані сегменти, які були злиті в результаті розширення. Тут структуруючий елемент більший, ніж використовуваний для розширення – блок розміром 20x20 пікселів. Остання операція – це перевірка міцності, яка порівнює кількість включених пікселів в області із опуклою оболонкою області. Після цього кроку можливі хибнопозитивні об'єкти видаляються і залишаються лише області штрих-коду.

2.2.2. Метод на основі сканування зображень

Метод Телкіна і Кафлана [6] був розроблений для людей із вадами зору або незрячих, щоб полегшити їхнє повсякденне життя. Він складається з двох основних етапів. По-перше, рівень шуму знижується за допомогою гаусового згладжування. Оскільки в оригінальній статті параметр σ не вказувався, було використано значення сігми 0,3. Потім значення піксельного градієнта обчислюються за допомогою оператора Собеля, таким чином створюючи своєрідне покращення країв зображення. Далі виконується бінаризація, щоб

пікселі, градієнт яких перевищує порогове значення, ставали білими, а інші – чорними. Тут було використане порогове значення, встановлене на 95% від максимального значення градієнта. На етапі виявлення спочатку сканується зображення в чотирьох напрямках (горизонтально, вертикально та в напрямку двох діагоналей). Спочатку проходить горизонтальне сканування, при якому виявляються крайові пікселі, орієнтовані вертикально. Цей метод шукає пікселі протилежної полярності поблизу кожного крайового пікселя. Область буде збережена, якщо в цій області є достатня кількість таких пікселів (як якщо б вони були частиною відрізка лінії). Далі слідує вертикальне сканування, коли залишаються ті сегменти, які мають майже однаковий початок і кінець, тому вони, швидше за все, належать до областей штрих-коду. На завершальному етапі виявлення штрих-коду для кожного пікселя в отриманому зображенні обчислюється значення ентропії, яке описує дисперсію інтенсивності в заданому околі навколо кожного пікселя.

2.2.3. Метод на основі фільтрації чорного циліндра

Алгоритм Джует та Кі заснований на фільтрації чорного циліндра (black top-hat) [41]. У попередній обробці метод коригує неідеальне зображення простим розтягуванням контрасту, щоб виділити відмінності між світлими та темними ділянками. Далі застосовується фільтрація, при якій розмір структурного елемента залежить від найширшої смуги в штрих-коді, який потрібно виявити. У оригінальній статті вказано блок 25x25 для зображень розміром 720×480 пікселів. Після бінаризації визначається контур. За ним слідує етап визначення орієнтації смужок, який виконується направленими розкриттями зображення з використанням відносно великого лінійного структурного елемента. Ці розкриття виконуються в 16 різних орієнтаціях з кроком 11,25 градусів. Спрямовані зображення розкриття підсумовуються і обчислюється зображення щільності з низькою роздільною здатністю, яке потім перетворюється назад у двійкове. Кожна область представляє потенційну

область штрих-коду. На останньому етапі відсіваються об'єкти, площа яких менше заданого порогу.

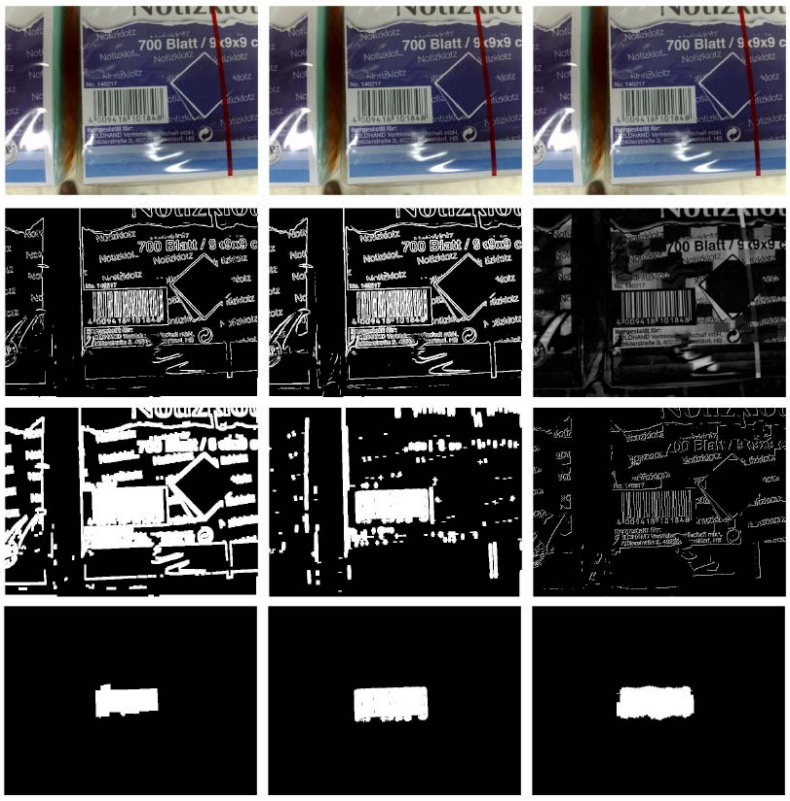

Рис.2.1. Виявлення штрих-коду методами з п.2.2.1, 2.2.2., 2.2.3

2.3. Узагальнена модель розшифрування штрих-коду на базі камери

На рис.2.2 показано захоплення зображення одного конкретного пікселя штрих-коду. Нехай $zone_{ij}$ позначає експоновану область для ij-го пікселя, тобто область, інтенсивність якої буде вимірюватися ij-м пікселем.

Рис.2.2. Середня інтенсивність у ij-му пікселі області $zone$ (зелене)

Інтенсивність моделюється просто як загальна кількість фотонів у піксельній області, де $intens(x, y)$ – рівень інтенсивності зображення на (x, y).

$$g_{ij} = \int_{zone_{ij}} intens(x, y)\, dx\, dy$$

Таким чином, для камери з $M \times N$ пікселів захоплене зображення є матрицею з елементами g_{ij}. На практиці ми отримуємо чорно-білі зображення, тому g_{ij} – це ціле число, яке приймає значення від 0 до 255. Насамкінець, варто зауважити, що коли штрих-код сканується за допомогою камери, знімається не лише зображення штрих-коду, але й навколишнє середовище (або шум). Тим не менш, запропонована модель також повинна підходити для захаращених зображень.

На даний момент ми обмежуємо аналіз випадком, коли штрих-код є одновимірним і горизонтальним відносно отвору камери. Тут x_0 – зсув або зміщення від лівого краю зображення до найпершої смуги ліворуч; d – розмір пікселя; x_i – праві краї смуг, які, як припускається, мають ширину h, як показано на рис.2.3. Один елемент коду відповідає одній цифрі і вимагає семи значень x_i. Ми можемо зменшити кількість параметрів у нашій задачі до двох, нормалізувавши розмір пікселя d до одиниці. Таким чином, єдині параметри, які нам потрібно мати на увазі, це горизонтальне зміщення або зсув штрих-коду x_0, і ширина найвужчої смуги h [28-30].

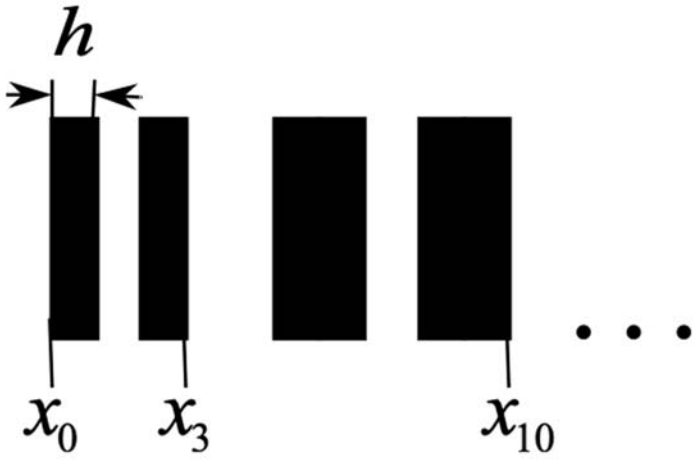

Рис.2.3. Передня частина штрих-коду, що ілюструє параметри камери

Представимо штрих-код у вигляді двійкової функції $intens(x) \in \{0,1\}$. Щоб представити цю двійкову функцію, ми введемо $\chi_j(x)$ – характеристичну функцію, визначену як:

$$\chi_j(x) = \begin{cases} 1, x_{j-1} < x < x_j \text{ (смуга } j) \\ \\ 0, \text{в інших випадках} \end{cases}$$

Ця функція визначає, чи бачить певну смужку піксель k. Знову ж таки, c є двійковим вектором, що представляє штрих-код, де, наприклад, для штрих-кодів категорії UPC $c \in \mathbb{R}^{95}$ і $c_j \in \{0,1\}$. За допомогою цієї конструкції штрих-код, представлений у вигляді двійкової функції, приймає форму:

$$intens(x) = \sum_{j=1}^{95} c_j \chi_j(x)$$

Нехай N буде кількістю пікселів, а матрицю камери будемо позначати $D \in \mathbb{R}^{N \times 95}$. Тоді kj-й елемент D є:

$$D_{kj} = \int_{y_{k-1}}^{y_k} \chi_j(x)\, dx$$

Без втрати загальності можемо встановити $y_k = k$, де $k = 0,1,2, \ldots, N$ і $x_j = x_0 + jk$. Також спостерігаємо, що $D_{kj} = 0$ для $(i)\ y_{k-1} > x_j$ та для $(ii)\ y_k < x_{j-1}$. Це видно з рис. 2.4.

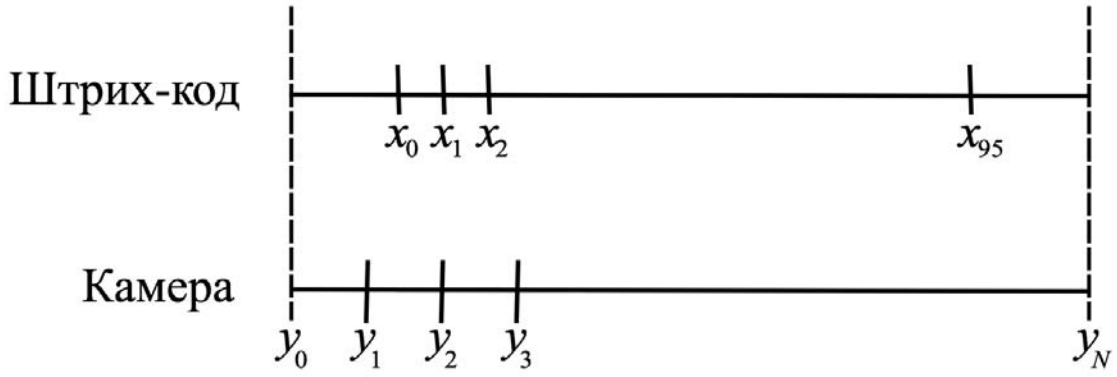

Рис.2.4. Схематичне зображення штрих-коду, отриманого з N пікселів

Припускається, що пікселі, які не перекриваються з $[x_0, x_{95}]$, мають нульову інтенсивність, як, наприклад, перший піксель на рис.2.4. Нехай $\lfloor x \rfloor$

позначає цілу частину x. $g_k = 0$ для $k < k_-$, де $k_- = \lfloor x_0 \rfloor$ і $g_k = 0$ для $k > k_+$, де $k_+ = \lfloor x_{95} \rfloor + 1$. Обов'язково, $D_{kj} = 0$ для $k \leq k_-$ і для $k \geq k_+$.

З таким розумінням спостережуване зображення є $g = \alpha Dc = \alpha DAz$, де $g \in \mathbb{R}^N$ – значення пікселів, що розглядаються як вектор, α – коефіцієнт масштабування для перепризначення значень пікселів від 0 до 255, z – двійковий вектор, що вибирає відповідні символи для кожної частини штрих-коду, A – матриця розміром 95 на 123, яка складається лише з нулів і одиниць і представляє словник цього виду штрих-коду.

Слід зазначити, що на практиці нульова інтенсивність відноситься до чорного кольору, а інтенсивність 255 – до білого. Зміна шкали відбулася тут без втрати загальності.

2.4. Опис та реалізація алгоритму зчитування штрих-кодів

Для реалізації процесу зчитування та розшифрування штрих-кодів різних типів на базі камери смартфону у контексті майбутнього практичного застосування були також детально вивчені такі три алгоритми, як Галло та Мандуччі [36], Л.Фана та Ксі [47], С.Фана [48].

Перший з названих вище методів для декодування не використовує бінаризацію, метод працює на шаблонах відносно даних щодо градацій сірого у всіх пікселях зображення, які можуть бути і розмитими, і слабкої роздільної здатності.

Другий алгоритм полягає у виділенні підобласті штрих-коду на області зображення, що його містить, у цьому випадку відмінністю від звичайного аналізу є використання інтегрального зображення. З його допомогою можна опрацьовувати розмиті зображення або такі, що містять нерівно розташовані зони локалізації штрих-коду, а також по-різному освітлені.

Нарешті, третій алгоритм забезпечує надійне розпізнавання штрих-коду з захаращених шумом зображень. Він використовує лінії сканування для

отримання області штрих-коду, потім позбавляється шуму в ній та за допомогою фільтру середньої смуги розшифровує штрих-код.

Підсумовуючи всі ці розглянуті принципи, методи та алгоритми обробки штрих-кодів, слід відзначити, що при використанні камери процес розшифрування стартує з процесу захоплення зображення, на якому знаходиться штрих-код. Він локалізується, відбувається виявлення його меж. Після цього визначаються особливості, що характерні для символіки різних типів штрих-кодів, вони впливають на визначення параметрів камери. Для відомої матриці камери використовується жадібний алгоритм, який відновлює бінарний код з локалізованого штрих-коду. Також зображення може бути спрощене до смуги з N пікселів, що зчитує послідовність показників інтенсивності в пікселях.

У даній системі запропонований алгоритм виявляє область штрих-коду за допомогою методу градієнта, а потім цю область обрізає. Модуль попередньої обробки зображення перетворює обрізане зображення штрих-коду в попередньо оброблене зображення у градаціях сірого, а потім зменшує шум і покращує контраст між смугами та пробілами в цьому попередньо обробленому зображенні. Потім це попередньо оброблене зображення передається в алгоритм декодування для вилучення номера зі штрих-коду. Порівняння цього числа з відповідним полем бази даних дозволяє знайти та отримати запис з потрібною інформацією.

Алгоритм локалізації Галло та Мандуччі забезпечує ефективне виконання задачі в тому числі з точки зору простоти та швидкості. Він припускає, що зображення штрих-коду фіксується орієнтованою на камеру системою таким чином, що його вертикальна вісь приблизно паралельна смугам, а далі це зображення перетворюється у відтінки сірого, як продемонстровано на рис. 2.5. Таким чином, щодо штрих-коду слід очікувати розширену область, представлену сильними горизонтальними градієнтами та слабкими вертикальними градієнтами.

Рис. 2.5. Оригінальне зображення та зображення у відтінках сірого

Відповідно, ми обчислюємо горизонтальні та вертикальні похідні $Ix(n)$ та $Iy(n)$ для кожного пікселя n. Потім вони об'єднуються нелінійним способом, заданим у вигляді $Ie(n) = |Ix(n)| - |Iy(n)|$. Доцільно припустити, що багато точок у штрих-коді повинні мати велике значення $Ie(n)$. Тоді запускається блоковий фільтр над $Ie(n)$ та отримується згладжена карту $Is(n)$.

У разі декодування на основі попередньо виявленої кінцевої точки рядка сканування просторове розташування обчислюється для кожного сегмента цифри в штрих-коді.

Розмір фільтра було обрано на основі діапазону розмірів вхідних зображень штрих-коду та мінімального розміру штрих-коду, який можна зчитувати нашим методом. Фільтрація блоків може бути реалізована настільки ефективно, щоб вимагалося лише кілька операцій на піксель. Зрештою, відбувається бінаризація $Is(n)$ з єдиним порогом, який обираємо за допомогою методу, запропонованого Оцу [49], і вихідне зображення виглядає таким чином, як це показано на рис.2.6.

Рис. 2.6. Зображення з застосування фільтру Гауса

Бінаризація використовується лише для локалізації штрих-коду, тоді як декодування виконується на зображенні в градаціях сірого. Через порогове значення карта $Is(n)$ може містити більше одного двійкового блоба. Замість обчислення пов'язаних компонентів порогової карти просто вибирається піксель N_0, який максимізує $Is(n)$, з припущенням, що правильний блоб (той, що відповідає штрих-кодові) містить такий піксель. Потім розгортаємо вертикальну та горизонтальну лінії від N_0 та формуємо прямокутник зі сторонами, паралельними осям зображення штрих-коду та що містить точки перетину цих ліній, із краєм бінарного набору даних. Слід зауважити, що тиха зона, тобто біла область навколо штрих-коду, яка полегшує локалізацію, межує з крайньою лівою та правою смугами штрих-коду. Тиха зона та великий розмір блок-фільтра гарантують, що вертикальні сторони прямокутника виходять за межі області штрих-коду принаймні на кілька пікселів, як показано на рис. 2.7.

Рис. 2.7. Зображення зони штрих-коду

Коли зображення береться для сканування, разом із штрих-кодом також додається інша непотрібна інформація, наприклад літери та слова, що оточують штрих-код. Таким чином, дуже важливо видалити шум і отримати лише область штрих-коду для належного сканування та декодування. Після локалізації штрих-коду видно лише область штрих-коду, а навколишня інформація видаляється шляхом встановлення всіх інших пікселів рівними 0. Потім ми виконуємо кадрування, спостерігаючи за інтенсивністю кожного пікселя та вилучаючи рядки штрих-коду, тобто пікселі з інтенсивністю більше 0.

У зображенні штрих-коду в реальному часі контраст між білими та чорними смугами низький через розмитість. Отже, необхідно покращити

контрастність зображення, щоб розрізняти смуги. Це виконується, роблячи чорні смуги на один відтінок темнішими в градаціях сірого порівняно з білими смугами, як показано на рис.2.8.

Рис.2.8. Відкадроване та покращене зображення штрих-коду

Зображення з посиленим контрастом спочатку піддається бінаризації. Але через отримання в реальному часі воно спотворюється. Щоб перетворити його на ідеальне, кожен стовпець сканується та перевіряється на максимальну кількість пікселів інтенсивності 0 або 1. Якщо певний стовпець містить більше пікселів з інтенсивністю 1, ніж 0, тоді весь стовпець перетворюється до пікселів з інтенсивністю 1. На рис.2.9. – ідеальне зображення штрих-коду.

Рис.2.9. Ідеальне зображення штрих-коду

Межа є пікселем, у якому відбувається раптова зміна інтенсивності. Створюється масив таких пікселів, що формують границі смужок зображення штрих-коду. Ширина смуги обчислюється відніманням послідовних елементів масиву вершин, що відповідають за межі. Ці ширини смуг надаються у якості вхідних даних для алгоритму декодування.

Даний алгоритм локалізації та розпізнавання може бути застосований для будь-якого типу штрих-коду на вході. Подальші процеси бінаризації та декодування залежать від конкретного типу штрих-коду. Для прикладу розглянемо роботу алгоритму для штрих-коду Code 128. Він використовується для багатьох документів посвідчення особи, зокрема і для студентських квитків,

інформація з яких має визначну роль у розв’язуваній за допомогою розпізнавання штрих-кодів практичній задачі.

Номер штрих-коду розшифровується [50] за допомогою масиву значень ширини штрихів. Таким чином, кодування штрих-коду включає чорні та білі лінії, що чергуються, змінної ширини від 1 до 4. Код починається з Start-символу і закінчується End-символом, між якими розташовуються зашифровані символи, кожен представлений 6 смужками (3 чорними та 3 білими). Розшифровка вимагає виконання зчитування смуги відповідно до стандарту EAN-128. Наприклад, штрих-код містить необроблене повідомлення виду 211214122411112214221114221114134111131222233111 2. Перевіряємо перші 6 цифр, що відповідають першим 3 чорним смугам і 3 білим смугам. У нашому прикладі 211214 означають Start-символ для Code 128B, отже розшифрування буде відбуватися відповідно до цього стовпця таблиці символів Code 128. Розділяємо повідомлення на блоки по 6 символів і розшифровуємо їх. Послідовність 122411 за таблицею відповідатиме символу «h», 112214 – символу «e» і т.д. Повідомленням, що міститься в штрих-коді, є слово «hello».

Також цей масив може бути перетворений до послідовності формату 11010010000100110000101011001000011001010000110010100001000111101010 00100110011000111010 11. Процес декодування полягатиме в тому, що масив розбивається на 11-бітні блоки і повідомлення отримується посимвольно, оскільки у штрих-коді Code 128 відповідно до кодування – типу A, B або C – набором з 11 бітів можуть бути літери або числа від 00 до 99. У представленій послідовності теж зашифровано слово «hello» [51].

Висновки до розділу 2

1. Визначено основні проблеми, що складають задачу зчитування штрих-кодів та намічені можливі шляхи до їх вирішення.

2. Здійснено огляд алгоритмів обробки штрих-кодів за категоріями, визначено характерні риси та особливості для подальшого вибору найбільш оптимального шляху реалізації у контексті простоти та ефективності.

3. Представлено модель процесу розшифрування штрих-коду за допомогою камери.

4. На базі розглянутих методів обробки штрих-кодів представлено адаптований алгоритм локалізації, у основі якого лежить алгоритм Галло та Мандучі, та декодування штрих-кодів на зображеннях з камери, що дозволить вирішити задачі дослідження та досягти його кінцевої мети.

РОЗДІЛ 3
ОПИС ПРОГРАМНОЇ СИСТЕМИ ТА ЇЇ РЕАЛІЗАЦІЇ

3.1. Архітектура розробленої системи

Для реалізації системи ідентифікації здобувачів за штрих-кодами студентських квитків була обрана форма мобільного додатку на базі Android. Процес сканування буде відбуватись за допомогою камери смартфону, яка захоплюватиме зображення штрих-коду та передаватиме його до додатку для локалізації для декодування зашифрованих у ньому даних. Також для функціонування системи необхідне сховище даних – сервер [52], до якого буде звертатись додаток для отримання повної інформації по зчитаному штрих-коду та зберігання пов’язаних з роботою даних. Крім того, у базі даних повинні міститись дані про викладачів, які власне і проводитимуть сканування та записуватимуть у неї результати обліку ідентифікованих здобувачів.

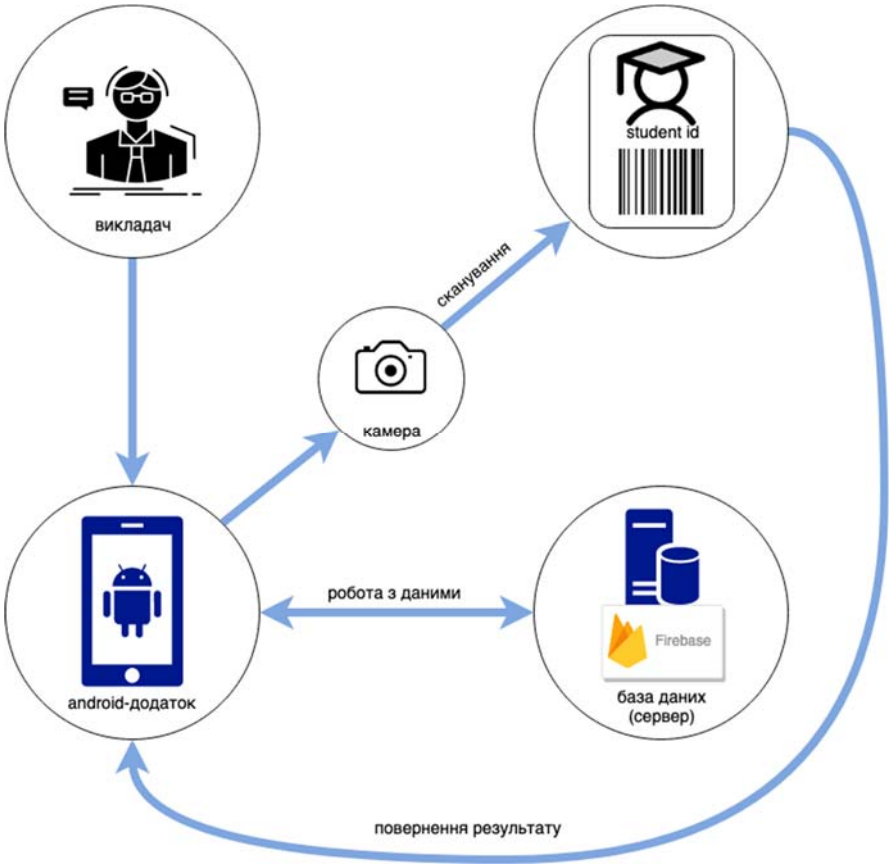

Рис.3.1. Структурна схема системи

Для написання додатку системи ідентифікації студентів було обрано середовище розробки Android Studio [53] та мова програмування Kotlin [54].

Android Studio є офіційним інтегрованим середовищем розробки для операційної системи Google Android, створеним на базі програмного забезпечення JetBrains' IntelliJ IDEA і спроєктованим спеціально для Android-розробки. Воно доступне для Windows, macOS та Linux. Android Studio підтримує мови програмування Java, JavaScript, Kotlin та C++. У 2019 році Kotlin стала офіційною та переважною мовою для розробки Android-додатків.

Kotlin – статично типізована кросплатформена мова програмування, що працює на базі JVM та розробляється компанією JetBrains. Вона працює на віртуальній машині Java, тому повністю сумісна з Java і не призводить до перешкод або збільшення розміру файлів. Kotlin є більш простою мовою для читання, оскільки потребує менше так званого шаблонного коду, крім того, ліквідує деякі помилки, наприклад, з нульовим вказівником [55].

Для забезпечення процесу зчитування штрих-кодів зі студентських квитків мобільний додаток співпрацює з камерою смартфону. Для організації роботи з камерою було використано деякі програмі рішення бібліотеки Zxing [56], зокрема, це дозволить визначати штрих-коди, незалежно від того, якого конкретного типу вони будуть, та проводити сканування також у альбомному і портретному режимах. Локалізація та декодування відбуваються за описаним у розділі 2 алгоритмом.

Робота системи потребує багатоцільового сховища даних. У ньому повинні зберігатися дані про здобувачів освіти, що стосуються процесу сканування та їх ідентифікації. Крім того, сховище повинно бути організовано таким чином, щоб забезпечити завдяки скануванню студентських квитків можливість для викладачів автоматизовано відмічати присутніх на заняттях, які вони проводять, а адміністрації – можливість моніторити діяльності навчального процесу та перевіряти його облік. У якості сховища даних було обрано сервіс хмарної СУБД Firebase.

Firebase – хмарна СУБД класу NoSQL, що дозволяє розробникам додатків зберігати і синхронізувати дані між кількома клієнтами. Її можна інтегрувати з додатками на Android. Для зберігання результатів роботи є сервіс Firebase Firestore, що забезпечує процес обміну даними між застосунками незалежно від якості мережі [57].

Для створюваної системи ідентифікації студентів Firebase також виступає у якості сервера – робота у системі організовується завдяки клієнт-серверній архітектури між додатком та базою даних [58].

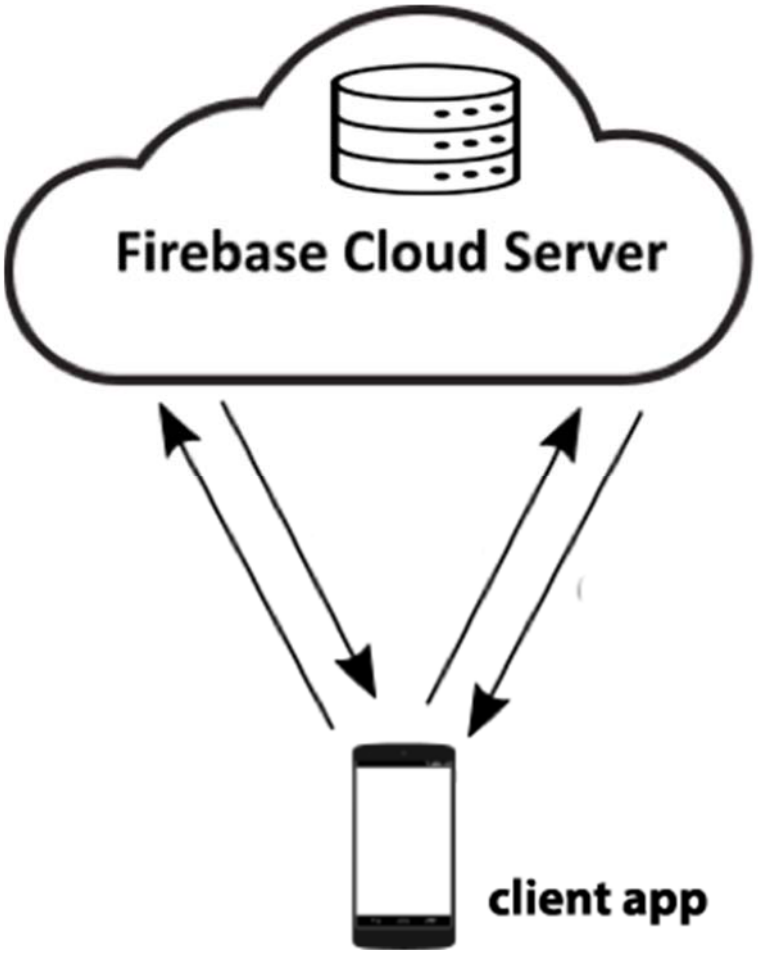

Рис.3.2. Клієнт-серверна архітектура

3.2. Опис структури та організації бази даних

Робота системи ідентифікації здобувачів та фіксації присутності на заняттях організована таким чином, що викладачі мають доступ до системи через комбінацію логін/пароль, які отримують від системного адміністратора. Дані для

авторизації зберігаються у Firebase Auth [59]. Після входу викладач має доступ до створення запису про нове заняття та до перегляду списку проведених занять. При створенні запису про нове заняття викладач обирає дисципліну та групу зі списку асоційованих з ним, інформація про що підтягується з колекцій сховища даних. Крім цього, викладач бачить список групи і в цей момент може починати сканувати штрих-коди квитків. У разі успішного сканування навпроти імені здобувача з'явиться відмітка. Після завершення цього процесу можна зберегти дані про заняття та відправити їх на сервер.

У системі також передбачена додаткова роль для дирекції, яка може переглядати інформацію стосовно будь-яких навчальних занять, редагувати дані та формувати вибірки для звітності.

Для забезпечення роботи основних функцій у нереляційній базі даних організовано 4 колекції:

1) disciplines (дисципліни) – містить назви дисциплін, групи, у яких вони викладаються та яким викладачем (teacher), поле teacher зберігає id цього викладача для зв'язку з іншими колекціями;

2) students (студенти) – містить усю інформацію про студента з квитка, тут же зберігається зашифроване в штрих-коді значення, завдяки якому після сканування отримується доступ до запису про студента;

3) teachers (викладачі) – містить інформацію про повне ім'я викладача, id та роль (звичайний викладач або дирекція);

4) visits (відвідування занять) – містить інформацію про всі проведені заняття та зберігає список присутніх та відсутніх здобувачів на кожному з них у мапі scannedData.

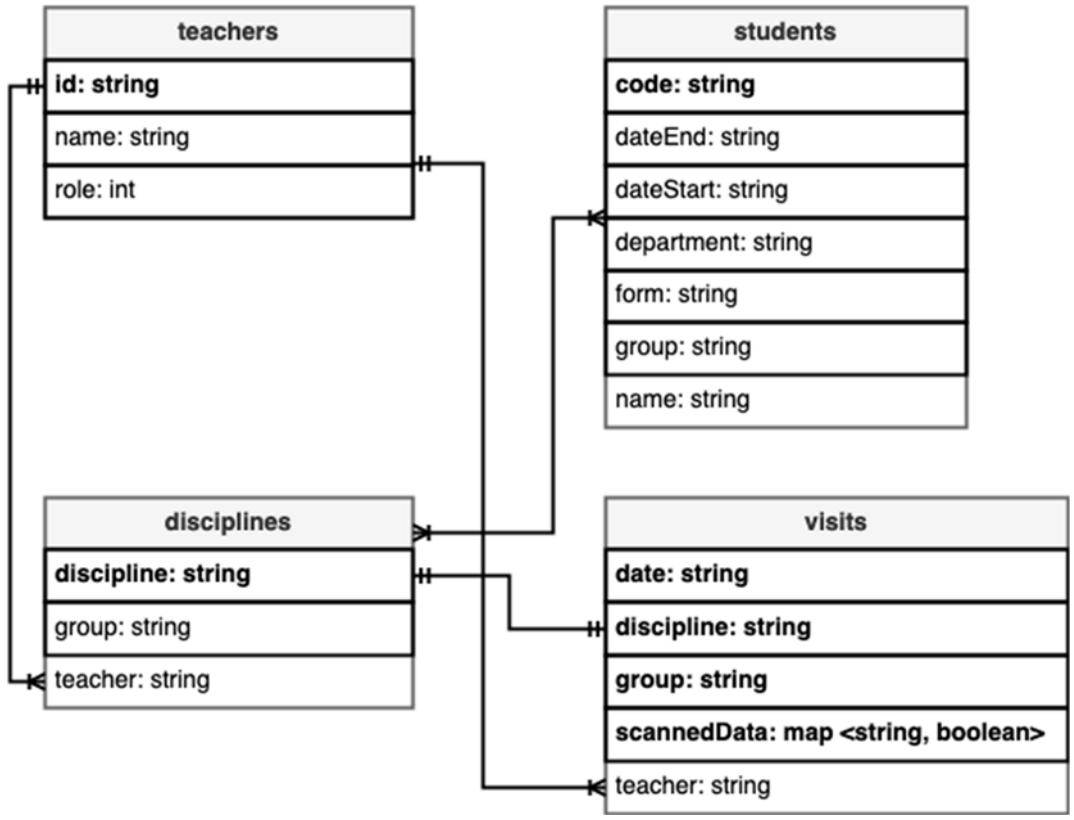

Рис.3.3. Модель даних

Вміст більшості полів у колекціях зашифровано за допомогою Advanced Encryption Standard (AES) [60] – симетричного алгоритму блочного шифрування, прийнятого урядом США у якості стандарту шифрування, що застосовується для захисту електронних даних. Основу алгоритму становлять заміни, підстановки та лінійні перетворення, кожне з яких виконується блоками по 128 біт, що є основою структури вхідних та вихідних даних. Повторення операцій відбувається неодноразово і в процесі кожної ітерації обчислюється унікальний ключ на основі ключа шифрування та вбудовується у подальші обчислення.

Наприклад, запис про відвідування заняття у базі зберігається у вигляді, показаному на рис.3.4. Зліва доступний для перегляду список колекцій у Firebase, посередині – список документів (записів), а у правій частині показані поля, що складають типовий запис документу відповідної колекції. У даному випадку це дата проведення, дисципліна, група, у якій проводилось заняття з дисципліни, список відміток відсканованих та невідсканових квитків групи, викладач, який проводив заняття.

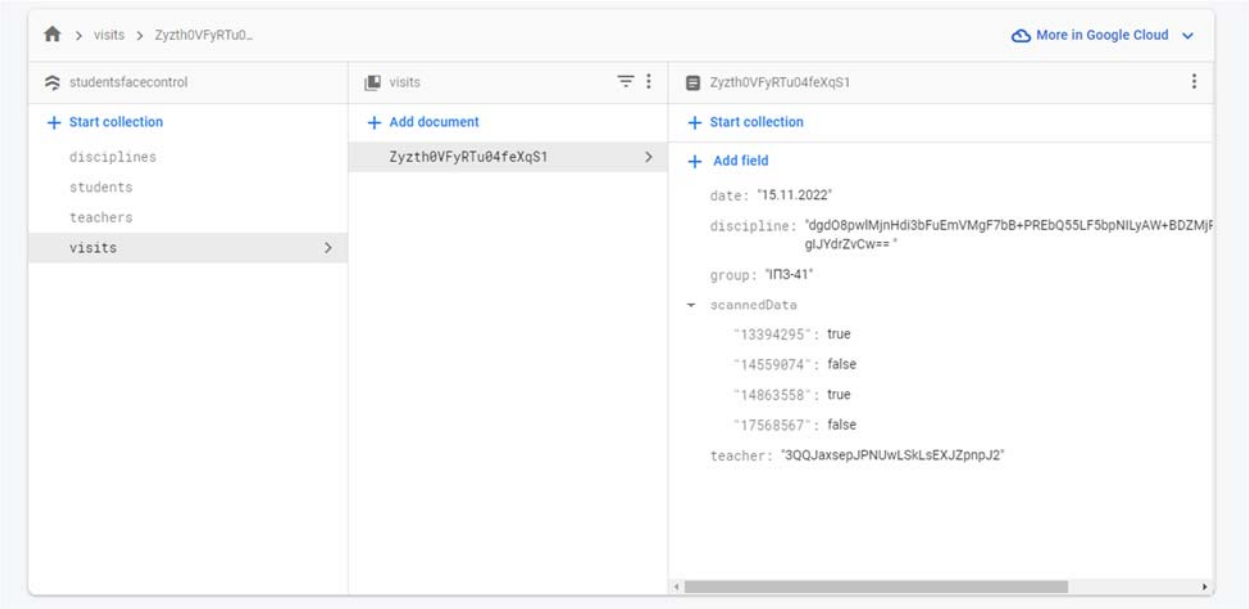

Рис.3.4. Організація даних у Firebase

3.3. Опис розробленого програмного забезпечення

Функціональність програмної системи відображена на діаграмі прецедентів [61]. Актори – викладач, дирекція, адміністратор.

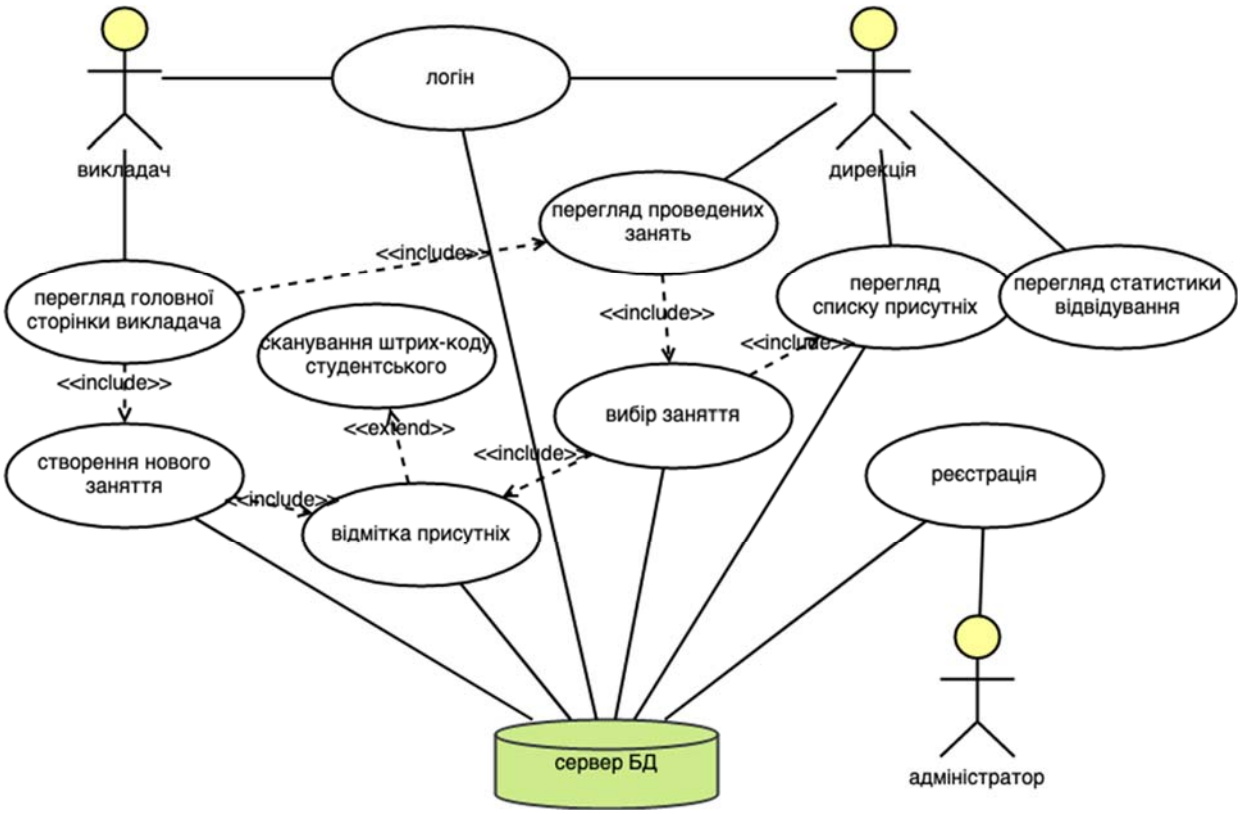

Рис.3.5. Діаграма прецедентів

Процес роботи з розробленим мобільним додатком демонструє діаграма діяльності (активності) [62] на рис.3.6. Діаграма діяльності візуально представляє ряд операцій або потік керування в системі, подібний до блок-схеми або діаграми потоків даних. Діаграми діяльності часто використовуються при моделюванні бізнес-процесів. Вони також можуть описувати можливі кроки акторів на діаграмі прецедентів. Змодельовані дії можуть бути послідовними та одночасними. В обох випадках діаграма діяльності матиме початок (початковий стан) і кінець (кінцевий стан).

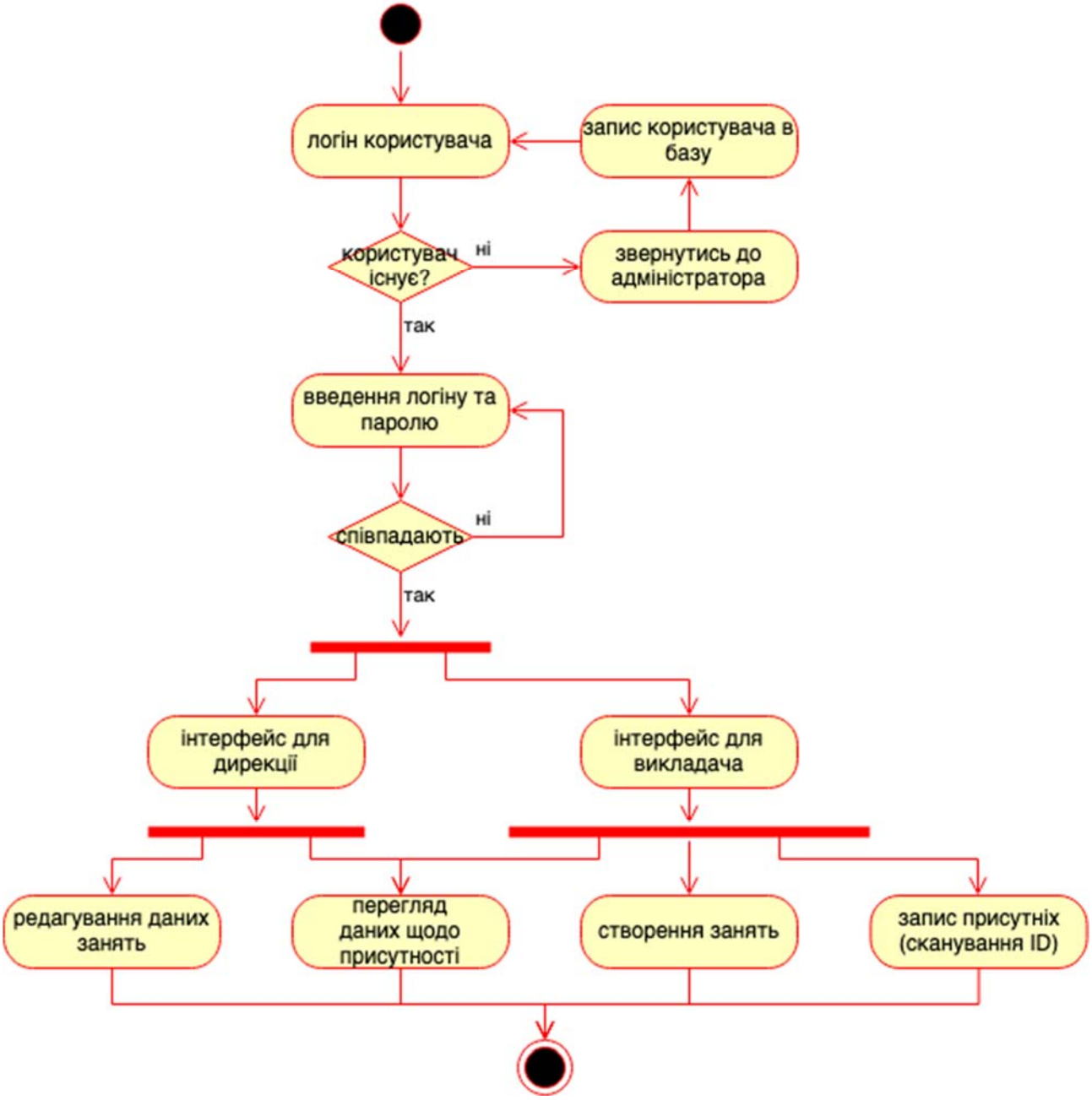

Рис.3.6. Діаграма діяльності

Робота починається з екрану логіну.

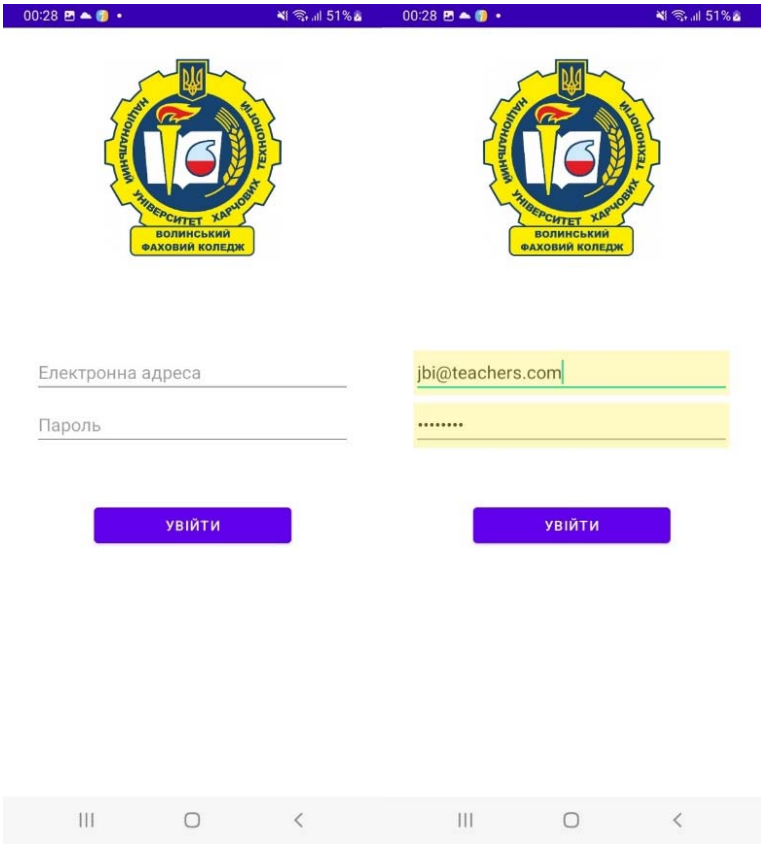

Рис.3.7. Екран логіну

У разі успішної авторизації користувач з роллю викладача потрапляє на головний екран додатку. Туди підвантажуються останні за датою проведені викладачем заняття, є можливість відкрити повний список проведених занять, перейшовши на відповідний екран, а також передивитись їх деталі, у яких будуть доступні списки студентів групи з відмітками про їх присутність. Крім того, викладач може внести правки у заняття та досканувати штрих-коди квитків здобувачів, які спізнились або ті, які випадково забули просканувати під час відмічання присутності. Натискання кнопки «Розпочати урок» переводить на екран заняття, де здійснюється вибір виду навчальної діяльності та дати проведення заняття, після чого зі сховища даних підтягується список групи та виконування подальших операцій.

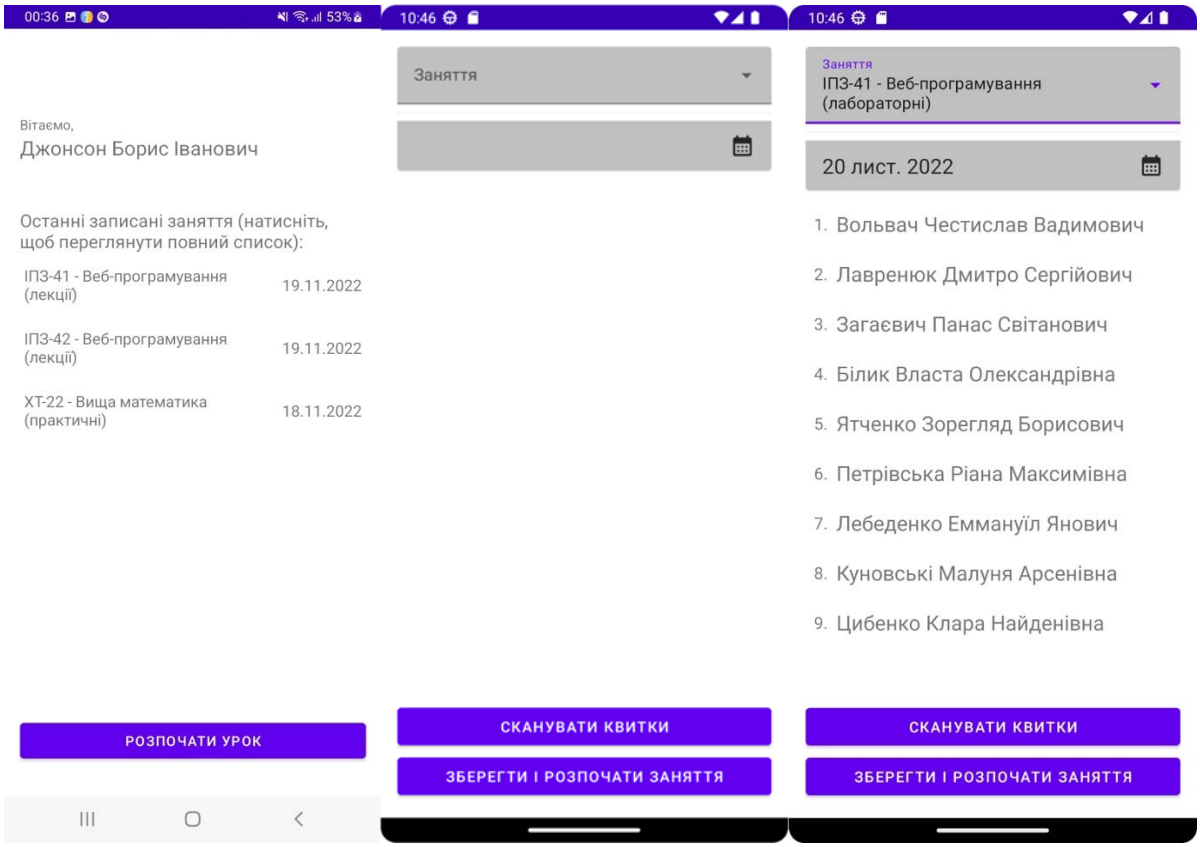

Рис.3.8. Головний екран та вибір заняття

Після того, як заняття обрано, список групи з’являється на тому ж екрані, а тоді кнопкою «Сканувати квитки» відбувається виклик камери, запит на дозволи проведення фото- та відеозйомки. Далі викладач може починати сканувати студентські квитки по одному. Після кожного успішного розпізнавання штрих-коду та звірки id студентського квитка з базою даних навпроти імені здобувача, штрих-код квитка якого відсканували, з’являється відмітка про присутність і таким чином список на екрані додатку оновлюється. Після завершення сканування усіх квитків натискається кнопка «Зберегти та розпочати заняття», дані про заняття відправляються у сховище на сервер, куди записуються і де зберігаються для переглядання відомостей у майбутньому викладачами та дирекцією, що може послужити зручним інструментом для підбиття підсумків навчального семестру, додаткової перевірки кількості проведених занять, що регламентовані навчальним планом.

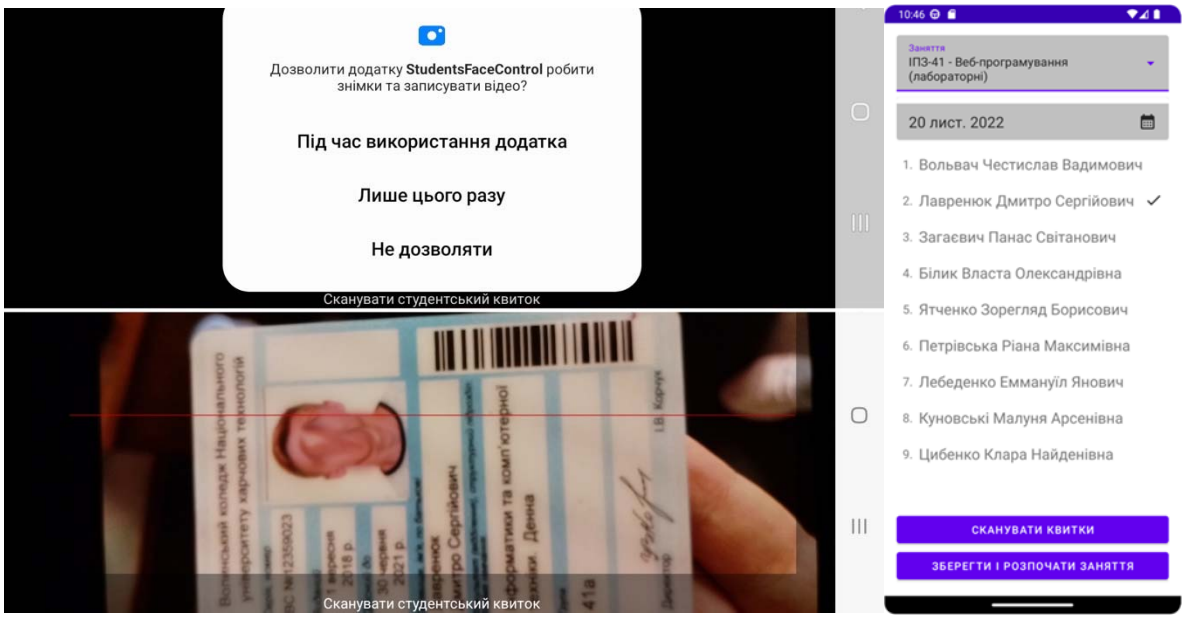

Рис.3.9. Сканування студентського квитка та запис присутності

Якщо у користувача призначена роль дирекції, то після входу їм також буде доступний екран перегляду вибірки про проведені заняття за різними параметрами, а також можливість вносити зміни та виправляти помилки.

Граф навігації по додатку для викладачів та дирекції виглядатиме наступним чином:

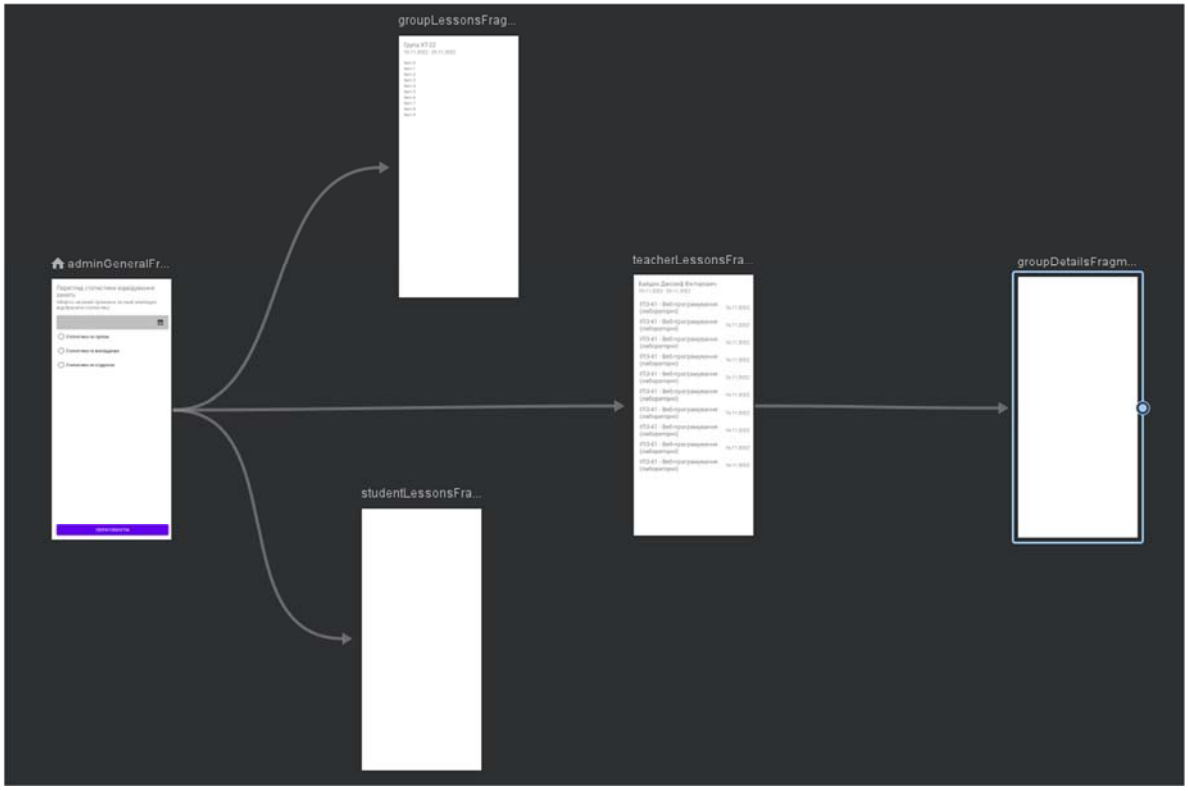

Рис.3.10. Граф навігації по додатку для викладачів

Характерним для представників дирекції буде наявність екрану перегляду статистики по відвідуванню, де можна буде формувати вибірки з бази даних по групам, викладачам, студентам.

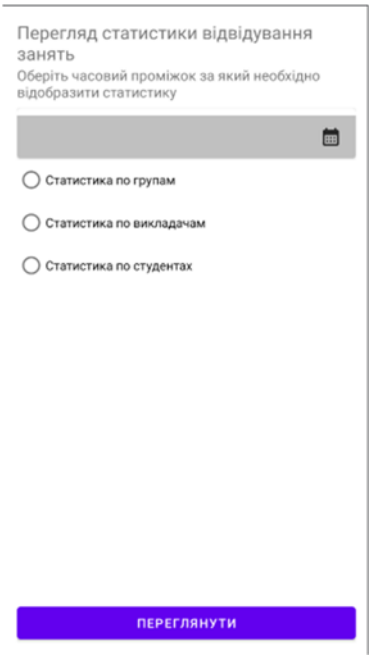

Рис.3.11. Екран перегляду статистику по відвідуванню

Для роботи з створеним додатком StudentsFaceControl потрібна версія ОС Android 8.0 та вище. Інсталяція здійснюється за допомогою apk-файлу.

Висновки до розділу 3

1. Здійснено опис структурних частин, що забезпечують роботу розроблюваної системи ідентифікації здобувачів за штрих-кодами студентських квитків для обліку відвідування занять.
2. Проведено огляд інструментів, методів та технологій, за допомогою яких створюється програмне забезпечення системи та реалізується його робота.
3. Охарактеризовано структуру сховища даних системи, принципи збереження інформації в ній, описано роботу з мобільним додатком, що забезпечує зв’язок з серверною базою даних та операції з її вмістом.

ВИСНОВКИ

В ході виконання випускної кваліфікаційної роботи було розроблено математичне та програмне забезпечення розпізнавання та декодування штрих-кодів за допомогою камери смартфону, яке у кінцевому результаті було інтегровано у систему ідентифікації здобувачів закладу освіти за студентськими квитками для обліку відвідування занять.

У процесі роботи над дослідженням було вирішено такі завдання:

1. Виконано огляд процесу кодування інформації за допомогою штрих-кодів, розглянуто їх класифікацію та визначено принципи побудови.

2. Розглянуто існуючі методи, алгоритми розпізнавання, зчитування та декодування штрих-кодів за допомогою камери смартфону.

3. На основі особливостей існуючих підходів до опрацювання штрих-кодів за допомогою камери реалізовано власне програмне рішення для зчитування штрих-кодів студентських квитків. Фундаментом для цього слугувала бібліотека zxing.

4. Розроблено архітектуру програмної системи та організацію системи збереження даних, що використовуються для ідентифікації та обліку здобувачів освіти за їх квитками, засобами Firebase.

5. Реалізовано клієнт-серверну систему, що забезпечує роботу процесу ідентифікації здобувачів освіти за студентськими квитками та зв’язок зі сховищем даних, що виступає у якості сервера, для фіксації перебування у навчальному закладі та формування звітних вибірок щодо відвідування.

Розроблене програмне забезпечення є достатньо ефективним у контексті коректного виконання заявленого функціоналу, що підтверджується впровадженням закладом освіти. Обслуговування системи не потребує додаткових значних матеріальних затрат, оскільки безкоштовного ресурсу Firebase у даному випадку достатньо для збереження даних за навчальний рік, а вартість збільшення об’єму сховища є дуже низькою.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Mobile Barcode Based Examination Attendance System / R. Ullah Khan et al. International Journal of Engineering & Technology. 2018. Vol. 7, no. 3.22. P. 49. URL: https://doi.org/10.14419/ijet.v7i3.22.17124

2. Barcode based Student Attendance System / K. LakshmiSudha et al. International Journal of Computer Applications. 2015. Vol. 119, no. 2. P. 1–5. URL: https://doi.org/10.5120/21036-3147

3. Using Barcode to Track Student Attendance and Assets in Higher Education Institutions / S. Elaskari et al. Procedia Computer Science. 2021. Vol. 184. P. 226–233. URL: https://doi.org/10.1016/j.procs.2021.04.005

4. Pros & Cons of Using Bar Code and QR Codes for Recording Attendance - Jackrabbit Class. Jackrabbit Class. URL: https://www.jackrabbitclass.com/blog/pros-cons-of-using-bar-code-and-qr-codes-for-recording-attendance

5. Morphological Operations - MATLAB & Simulink. MathWorks - Makers of MATLAB and Simulink - MATLAB & Simulink. URL: https://www.mathworks.com/help/images/morphological-filtering.html

6. Tekin E., Coughlan J. M. An algorithm enabling blind users to find and read barcodes. 2009 Workshop on Applications of Computer Vision (WACV), Snowbird, UT, USA, 7–8 December 2009. 2009. URL: https://doi.org/10.1109/wacv.2009.5403098

7. Top-hat filtering and bottom-hat filtering. Online books from Yougui Liao. URL: https://www.globalsino.com/EM/page1004.html

8. Image binarization (1): Introduction. The Craft of Coding. URL: https://craftofcoding.wordpress.com/2017/02/13/image-binarization-1-introduction/

9. Coding theory - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Coding_theory

10. Code - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Code

11. Digital data - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Digital_data

12. Kelbert M., Suhov Y. Information Theory and Coding by Example. Cambridge University Press, 2013. 526 p.

13. Types of Encoding Techniques - Javatpoint. www.javatpoint.com. URL: https://www.javatpoint.com/types-of-encoding-techniques

14. What Every Programmer Absolutely, Positively Needs to Know About Encodings and Character Sets to Work With Text. kunststube. URL: https://kunststube.net/encoding/

15. Data Encoding Techniques. Online Tutorials Library. URL: https://www.tutorialspoint.com/digital_communication/digital_communication_data_encoding_techniques.htm

16. Barcodes: A Brief History. Tracking Software for Returnable Containers | TrackAbout. URL: https://corp.trackabout.com/blog/barcodes-brief-history

17. Bashkar Raj A. Barcodes: Technology and Implementation. india : McGraw-Hill Education, 2001. 346 p.

18. History of the Barcode Scanner: Who Invented the Barcode Scanner?. Refurbished IT Solutions | Barcode Scanners, Printers, & More | DBK. URL: https://www.dbk.com/resources/barcode-scanner-history.html

19. Behind Barcodes: How They Work. Oracle NetSuite. URL: http://www.netsuite.com/portal/resource/articles/inventory-management/barcode.html

20. Admin. Chapter 2. Character set encoded in barcodes. Dynamsoft Blog. URL: https://www.dynamsoft.com/blog/insights/code39-code128-barcode-reading-101/

21. Guide to Barcode Types and Standards: 1D, 2D Barcode Symbologies, Requirements, and Standards-Issuing Entities. Camcode. URL: https://www.camcode.com/blog/guide-to-barcode-types-standards/ (дата звернення: 11.07.2022).

22. Types of Barcodes: Choosing the Right Barcode. Scandit. URL: https://www.scandit.com/blog/types-barcodes-choosing-right-barcode/

23. Barcoding Systems | Barcode Inventory System & Solutions. URL: https://www.barcoding.co.uk/wp-content/uploads/2016/02/E-Book_Total-Guide-to-Barcoding.pdf

24. Barcode - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Barcode

25. Comparison of Barcode Types | Dynamsoft. Dynamsoft - Experts in Document Capture and Barcode Reading SDKs. URL: https://www.dynamsoft.com/barcode-reader/barcode-types/compare-barcode/

26. Chai D., Hock F. Locating and Decoding EAN-13 Barcodes from Images Captured by Digital Cameras. 2005 5th International Conference on Information Communications & Signal Processing, Bangkok, Thailand. URL: https://doi.org/10.1109/icics.2005.1689328

27. Ohbuchi E., Hanaizumi H., Lim Ah Hock. Barcode Readers using the Camera Device in Mobile Phones. 2004 International Conference on Cyberworlds, Tokyo, Japan. URL: https://doi.org/10.1109/cw.2004.23 (date of access: 19.11.2022).

28. Kutiyanawala A., Qi X., Tian J. A simple and efficient approach to barcode localization. Signal Processing (ICICS), Macau, China, 8–10 December 2009. 2009. URL: https://doi.org/10.1109/icics.2009.5397472

29. Parikh D., Jancke G. Localization and Segmentation of A 2D High Capacity Color Barcode. 2008 IEEE Workshop on Applications of Computer Vision (WACV), Copper Mountain, CO, USA, 7–9 January 2008. 2008. URL: https://doi.org/10.1109/wacv.2008.4544033

30. Jain A. K., Chen Y. Bar code localization using texture analysis. 2nd International Conference on Document Analysis and Recognition (ICDAR '93), Tsukuba Science City, Japan. URL: https://doi.org/10.1109/icdar.1993.395786

31. Automatic Real-Time Barcode Localization in Complex Scenes / C. Zhang et al. 2006 International Conference on Image Processing, Atlanta, GA, 8–11 October 2006. 2006. URL: https://doi.org/10.1109/icip.2006.312435

32. Kaur S., Maini R. Comprehensive Analysis of Barcode Localization Methods. International Journal of Engineering, Research & Technology. 2014. Vol. 3, issue. 4. P. 958–962. URL: https://doi.org/10.17577/IJERTV3IS041138

33. Adelmann R. Toolkit for bar code recognition and resolving on camera. Gesellschaft für Informatik e.V. : Text/Conference Paper, Bonn. 2006. P. 366–373.

34. Tuinstra T.R. Reading Barcodes from Digital Imagery. PhD thesis, Cedarville University. 2005. URL: https://www.semanticscholar.org/paper/Reading-Barcodes-from-Digital-Imagery-Tuinstra-Cedarville/

35. Tekin E., Coughlan J. A Bayesian Algorithm for Reading 1D Barcodes. 2009 Canadian Conference on Computer and Robot Vision (CRV), Kelowna, British Columbia, Canada, 25–27 May 2009. 2009. URL: https://doi.org/10.1109/crv.2009.31

36. Gallo O., Manduchi R. Reading 1D Barcodes with Mobile Phones Using Deformable Templates. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2011. Vol. 33, no. 9. P. 1834–1843. URL: https://doi.org/10.1109/tpami.2010.229

37. Kongqiao Wang, Yanming Zou, Hao Wang. Bar code reading from images captured by camera phones. IEE Mobility Conference 2005. The Second International Conference on Mobile Technology, Applications and Systems, Guangzhou. 2005. URL: https://doi.org/10.1109/mtas.2005.207160

38. Mazakov T. Z., Abilkaiyr Z. S. BAR CODE RECOGNITION IN STRONG DISTORTED AND LOW-RESOLUTION IMAGES. PHYSICO-MATHEMATICAL SERIES. 2021. Vol. 2, no. 336. P. 109–114. URL: https://doi.org/10.32014/2021.2518-1726.28

39. Bodnar P., Nyul L. G. Improving Barcode Detection with Combination of Simple Detectors. 2012 Eighth International Conference on Signal-Image Technology & Internet-Based Systems (SITIS 2012), Naples, 25–29 November 2012. 2012. URL: https://doi.org/10.1109/sitis.2012.52

40. Lin D.-T., Lin M.-C., Huang K.-Y. Real-time automatic recognition of omnidirectional multiple barcodes and DSP implementation. Machine Vision and

Applications. 2010. Vol. 22, no. 2. P. 409–419. URL: https://doi.org/10.1007/s00138-010-0299-3

41. Juett J., Qi X. Barcode localization using bottom-hat filter. NSF Research Experience for Undergraduates. 2005. URL: https://docplayer.net/50346182-Barcode-localization-using-a-bottom-hat-filter.html

42. Katona M., Nyul L. G. A Novel Method for Accurate and Efficient Barcode Detection with Morphological Operations. 2012 Eighth International Conference on Signal-Image Technology & Internet-Based Systems (SITIS 2012), Naples, 25–29 November 2012. 2012. URL: https://doi.org/10.1109/sitis.2012.53

43. Youssef S. M., Salem R. M. Automated barcode recognition for smart identification and inspection automation. Expert Systems with Applications. 2007. Vol. 33, no. 4. P. 968–977. URL: https://doi.org/10.1016/j.eswa.2006.07.013

44. Bodnár P., Nyúl L. G. Barcode Detection with Morphological Operations and Clustering. Signal Processing, Pattern Recognition and Applications, Crete, Greece. Calgary, AB, Canada, 2012. URL: https://doi.org/10.2316/p.2012.778-014

45. Saisha. FINDING THE EDGE: CANNY AND SOBEL DETECTORS (Part 1). Medium. URL: https://medium.com/srm-mic/finding-the-edge-canny-and-sobel-detectors-part-1-65a59b7ef62a

46. Šimurda P. Barcode Localization in Image. Information Sciences and Technologies Bulletin of the ACM Slovakia 3. 2012. P. 55–56. URL: http://acmbulletin.fiit.stuba.sk/abstracts/simurdaSPY.pdf

47. Fang L., Xie C. 1-D Barcode Localization in Complex Background. 2010 International Conference on Computational Intelligence and Software Engineering (CiSE), Wuhan, China, 10–12 December 2010. 2010. URL: https://doi.org/10.1109/cise.2010.5677072

48. Automatic Recognition of Noisy Code-39 Barcode / X. Fang et al. 16th International Conference on Artificial Reality and Telexistence-Workshops (ICAT'06), Hangzhou, Zhejiang, China, 29 November – 1 December 2006. 2006. URL: https://doi.org/10.1109/icat.2006.45

49. Otsu N. A Threshold Selection Method from Gray-Level Histograms. IEEE Transactions on Systems, Man, and Cybernetics. 1979. Vol. 9, no. 1. P. 62–66. URL: https://doi.org/10.1109/tsmc.1979.4310076

50. Barcode 128 Cipher - 128a 128b 128c - Online Decoder, Encoder. dCode - Solveurs, Crypto, Maths, Codes, Calculs en Ligne. URL: https://www.dcode.fr/barcode-128

51. DmitrySpb79. How does a barcode work?. Хабр. URL: https://habr.com/ru/post/439768/

52. What Is a Database Server & What Is It Used For. Knowledge Base by phoenixNAP. URL: https://phoenixnap.com/kb/what-is-a-database-server

53. Android Studio - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Android_Studio

54. Основы Kotlin. Введение – Fandroid.info. Fandroid.info – Уроки по разработке андроид-приложений. URL: https://www.fandroid.info/osnovy-kotlin-vvedenie/

55. Kotlin (programming language) - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Kotlin_(programming_language)

56. GitHub - zxing/zxing: ZXing ("Zebra Crossing") barcode scanning library for Java, Android. GitHub. URL: https://github.com/zxing/zxing/

57. Add Firebase to your Android project | Firebase for Android. Firebase. URL: https://firebase.google.com/docs/android/setup

58. Основные понятия и особенности клиент-серверной архитектуры - TestMatick. TestMatick. URL: https://testmatick.com/ru/osnovnye-ponyatiya-i-osobennosti-klient-servernoj-arhitektury/

59. Using Firebase Authentication | FlutterFire. FlutterFire | FlutterFire. URL: https://firebase.flutter.dev/docs/auth/usage/

60. Щенев В. AES шифрование и Android клиент. Хабр. URL: https://habr.com/ru/company/rambler_and_co/blog/279835/

61. What is Use Case Diagram?. Ideal Modeling & Diagramming Tool for Agile Team Collaboration. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/.

62. Activity Diagram - Activity Diagram Symbols, Examples, and More. SmartDraw - Create Flowcharts, Floor Plans, and Other Diagrams on Any Device. URL: https://www.smartdraw.com/activity-diagram/.

ДОДАТКИ

Додаток А

Копії публікацій

Науковий журнал "Комп'ютерно-інтегровані технології: освіта, наука, виробництво"
124 *Луцьк, 2022. Випуск № 48*

DOI: https://doi.org/10.36910/6775-2524-0560-2022-48-19
УДК 004.932
Юхта Олександр Андрійович[1], студент
https://orcid.org/0000-0002-4057-3052
Ройко Олександр Юрійович[2], к.т.н., викладач
https://orcid.org/0000-0001-8642-7707
[1]Західноукраїнський національний університет, м. Тернопіль, Україна
[2]Відокремлений структурний підрозділ «Волинський фаховий коледж Національного університету харчових технологій», м. Луцьк, Україна

АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ РОЗПІЗНАВАННЯ ШТРИХ-КОДІВ НА БАЗІ КАМЕРИ

Юхта О.А., Ройко О.Ю. Алгоритмічне забезпечення розпізнавання штрих-кодів на базі камери. Представлено огляд алгоритмів виявлення та декодування штрих-кодів на прикладі одновимірних штрих-кодів. Запропоновано алгоритм, що забезпечує розпізнавання за допомогою камери та розшифрування штрих-кодів для подальшої роботи з асоційованими з ними даними.
Ключові слова: штрих-код, камера, локалізація, декодування, розшифрування, Code 128.

Yukhta O., Roiko O. Algorithmic support for camera-based barcode reading. An overview of the algorithms for detecting and decoding barcodes on the example of one-dimensional barcodes is presented. An algorithm that provides camera-based recognition and decoding of barcodes for further work with the data associated with them is proposed.
Keywords: barcode, camera, localization, decoding, decryption, Code 128.

Постановка наукової проблеми. Незважаючи на те, що технологія штрих-кодів існує з 1960-х років, декодери на основі камер додають нових викликів в процес їх декодування. Наприклад, тепер ми повинні подбати про роздільну здатність зображення. Хоча сучасні камери є багатопіксельними, штрих-код на захопленому зображенні зазвичай насправді міститься в невеликій підобласті області зображення. Таким чином, роздільна здатність може бути обмежена, якщо ніхто навмисно не максимізує її, що зазвичай не є доцільним на практиці. Зчитування штрих-кодів на великих відстанях є ще однією проблемною ділянкою, де має значення роздільна здатність. Крім того, ще однією перешкодою для зчитування може бути шум від спотворення зображення штрих-коду. Такі викривлення та дефекти можуть бути результатом різноманітних умов навколишнього середовища, включаючи нерівномірне освітлення, кут камери під час зйомки зображення і фізичну поверхню, наприклад, вигнуті або відбиваючі. Під час пошуку штрих-коду на зображенні виникає багато ускладнень – ця проблема відома як локалізація. Тому додатково необхідно приділити увагу проблемі ідентифікації початкової та кінцевої послідовностей штрих-коду для того, щоб мати опорні точки для його декодування. [1]

Нарешті, для роботи слід мати та використовувати практичний алгоритм. Навіть для найкращих умов щодо якості та чіткості вихідного зображення, за яким буде відбуватися зчитування штрих-коду, існує великий попит на швидкі та ефективні алгоритми. Практичність і функціональність вимагають того, щоб штрих-коди можна було сканувати та декодувати протягом кількох мілісекунд.

У цій роботі вивчаються алгоритми виявлення, зчитування та декодування штрих-кодів для їх адаптації під вирішення практичних задач.

Аналіз досліджень. Зчитування штрих-кодів досліджується десятиліттями. Після постійного вдосконалення тепер воно представлене усталеними промисловими стандартом. Донедавна зчитування штрих-кодів виконувалося майже виключно за допомогою спеціального обладнання. Незважаючи на те, що використання зчитувачів на основі камери зростає, більшість проблем, пов'язаних із цим методом, ще належить вирішити. Комерційні сканери, які використовуються в супермаркетах, освітлюють штрих-код смугою імпульсного світла та записують інтенсивність його відображення. Оскільки сканер використовує активне освітлення, він робить штрих-коди практично нечутливими до змін навколишнього освітлення. Крім того, їх структура часто вимагає, щоб штрих-коди були відносно близько до сканера, щоб успішно сканувати їх. Загалом ці спеціальні пристрої створюють високоякісний сигнал, який дозволяє надійно зчитувати штрих-код. З іншого боку, читання штрих-кодів за допомогою камер створює нові проблеми. Основною проблемою є низька якість зображення через шум або низький контраст. [2,6]

Ораціо Галло і Роберто Мандучі запропонували новий підхід до декодування штрих-кодів, який обходить процес бінаризації. Їх метод заснований на шаблонах, що деформуються, при цьому використовуючи всю інформацію про градації сірого кожного пікселя. Вони використали символ штрих-коду UPC-A та представили новий алгоритм декодування штрих-коду (локалізація та зчитування), який може працювати з розмитими зображеннями з шумом і низькою роздільною здатністю. [4]

Люпін Фан і Хао Ксі представили алгоритм, який може виявляти підобласть штрих-коду навіть на складному фоні за допомогою аналізу зображення на основі області. На відміну від традиційного аналізу зображень на основі області, алгоритм обробляється з інтегральним зображенням. Він може працювати з зображеннями, які розмиті, містять похило розташовані області штрих-коду та зняті при різному освітленні. [5]

Сяньон Фан запропонував швидкий і надійний метод розпізнавання штрих-коду Code 39, що містить шум. Запропонований метод складається з двох етапів: пошуку та декодування. На першому кроці всі зірочки на зображенні виявляються рівномірно визначеними лініями сканування, а потім ці лінії з однаковими напрямками зіставляються разом, щоб отримати дійсну область штрих-коду. На другому кроці для усунення шуму в області штрих-коду застосовується локальний метод усунення шуму. Крім того, для декодування штрих-коду використовується фільтр середньої смуги. Цей метод також має певні недоліки. Для зображень, роздільна здатність яких значно відрізняється від 2400x3500, цей підхід не може успішно розпізнати штрих-код автоматично, оскільки багато граничних значень потрібно встановити знову, наприклад, локальний поріг шумозаглушення. [6]

Виклад основного матеріалу й обгрунтування отриманих результатів дослідження. У багатьох напрямках сфери обслуговування спостерігається сильний рух від традиційного лазерного сканування та декодування до декодування на основі камери, зокрема і з допомогою камери мобільних пристроїв. Замість проходження лазером по штрих-коду, штрих-код фіксується камерою, що дозволяє потенційно більше надійне декодування зашифрованої у ньому інформації.

Штрих-коди можуть бути одновимірними (лінії чорних і білих смуг) або двовимірними (концентричні кільця або квадрати модулів) і існувати в різних формах та розмірах з різною ємністю та можливістю виправлення помилок. Ми визначаємо символіку як представлення будь-якого символу ASCII послідовністю нулів та одиниць. Щоб перетворити це на зображення штрих-коду, «0» та «1» замінюються білими та чорними модулями відповідно певного фіксованого розміру. [1]

Оскільки деякі символіки підходять певним програмам краще, ніж інші, існує багато різних застосувань штрих-кодів. Штрих-коди використовуються для інвентаризації, програм охорони здоров'я, продажів, виробництва, поштових застосувань, складування, доставки, віртуальних квитків та багатьох інших програм. Незважаючи на те, що розробляються нові технології, які виконують подібні функції, штрих-коди залишаються широко використовуваними сьогодні завдяки своїй недорогій вартості, гнучкості та надійності.

Оскільки проблема, для якої планується застосувати спроєктований алгоритм, передбачає роботу з одновимірними штрих-кодами, то результати будуть описані саме для них.

Система зчитування штрих-кодів на основі камери передбачає два етапи автоматичного сканування всіх штрих-кодів:

1. Знайти положення штрих-коду.
2. Розшифрувати штрих-код.

Більш детально, процес декодування штрих-коду за допомогою камери починається із захоплення зображення, що містить штрих-код. Штрих-код локалізується та виявляються його межі. Особливості, характерні для символіки, визначаються та використовуються для визначення параметрів камери. Коли матриця камери відома, двійковий код відновлюється шляхом запуску жадібного алгоритму для локалізованого зображення. Вхідне зображення можна додатково спростити до простої смужки з N пікселів, яка зчитує послідовність значень, що представляють інтенсивність зображення в пікселях. [3, 9]

У цій системі запропонований алгоритм виявляє область штрих-коду за допомогою методу градієнта, а потім цю область обрізає. Модуль попередньої обробки зображення перетворює обрізане зображення штрих-коду в попередньо оброблене зображення у градаціях сірого, а потім зменшує шум і покращує контраст між смугами та пробілами в цьому попередньо обробленому зображенні. Потім це попередньо оброблене зображення передається в алгоритм декодування для

вилучення номера зі штрих-коду. Порівняння цього числа з відповідним полем бази даних дозволяє знайти та отримати запис з потрібною інформацією.

Було виявлено, що простий і швидкий алгоритм, представлений Ораціо Галло та Роберто Мандучі, забезпечує чудові результати навіть у складних ситуаціях. Цей алгоритм локалізації припускає, що зображення штрих-коду фіксується орієнтованою на камеру системою таким чином, що його вертикальна вісь приблизно паралельна смугам, як показано на рис. 1. Це зображення далі перетворюється на зображення у відтінках сірого, як показано на рис. 1. Таким чином, щодо штрих-коду слід очікувати розширену область, представлену сильними горизонтальними градієнтами та слабкими вертикальними градієнтами. [8]

Рис. 1. Оригінальне зображення та зображення у відтінках сірого

Відповідно, ми обчислюємо горизонтальні та вертикальні похідні $Ix(n)$ та $Iy(n)$ для кожного пікселя n. Потім ми об'єднуємо їх нелінійним способом, заданим як $Ie(n) = |Ix(n)| - |Iy(n)|$. Доцільно припустити, що багато точок у штрих-коді повинні мати велике значення $Ie(n)$. Ми запускаємо блоковий фільтр над $Ie(n)$, отримуючи згладжену карту $Is(n)$.

У разі декодування на основі попередньо виявленої кінцевої точки рядка сканування просторове розташування обчислюється для кожного сегмента цифри в штрих-коді.

Розмір фільтра було обрано на основі діапазону розмірів вхідних зображень штрих-коду та мінімального розміру штрих-коду, який можна зчитувати нашим методом. Зауважте, що фільтрація блоків може бути реалізована настільки ефективно, щоб вимагалося лише кілька операцій на піксель. Зрештою, ми бінаризуємо $Is(n)$ з єдиним порогом, який вибираємо за допомогою методу, запропонованого Оцу, і вихідне зображення виглядає як показано на рис. 2. [7]

Рис. 2. Зображення з застосування фільтру Гауса

Ця бінаризація використовується лише для локалізації штрих-коду, тоді як декодування виконується на зображенні в градаціях сірого. Через порогове значення карта $Is(n)$ може містити більше одного двійкового блоба. Замість обчислення пов'язаних компонентів порогової карти просто вибирається піксель No, який максимізує $Is(n)$, з припущенням, що правильний блоб (той, що відповідає штрих-кодові) містить такий піксель. Потім розгортаємо вертикальну та горизонтальну лінії від No та формуємо прямокутник зі сторонами, паралельними осям зображення штрих-коду та що містить точки перетину цих ліній, із краєм блоба. Слід зауважити, що тиха зона, тобто біла область навколо штрих-коду, яка полегшує локалізацію, межує з крайньою лівою та правою смугами штрих-коду. Тиха зона та великий розмір блок-фільтра гарантують, що вертикальні сторони прямокутника виходять за межі області штрих-коду принаймні на кілька пікселів, як показано на рис.3.

Рис. 3. Зображення зони штрих-коду

Коли зображення береться для сканування, разом із штрих-кодом також додається інша непотрібна інформація, наприклад літери та слова, що оточують штрих-код. Таким чином, дуже важливо видалити шум і отримати лише область штрих-коду для належного сканування та декодування. Після локалізації штрих-коду видно лише область штрих-коду, а навколишня інформація видаляється шляхом встановлення всіх інших пікселів рівними 0. Потім ми виконуємо кадрування, спостерігаючи за інтенсивністю кожного пікселя та вилучаючи рядки штрих-коду, тобто пікселі з інтенсивністю більше 0.

У зображенні штрих-коду в реальному часі контраст між білими та чорними смугами низький через розмитість. Отже, необхідно покращити контрастність зображення, щоб розрізняти смуги. Це виконується, роблячи чорні смуги на один відтінок темнішими в градаціях сірого порівняно з білими смугами, як показано на рис. 4.

Рис. 4. Відкадроване та покращене за контрастністю зображення штрих-коду

Зображення з посиленим контрастом спочатку піддається бінаризації. Але через отримання в реальному часі це зображення спотворюється. Щоб перетворити його на ідеальне зображення, кожен стовпець сканується та перевіряється на максимальну кількість пікселів інтенсивності 0 або 1. Якщо певний стовпець містить більше пікселів з інтенсивністю 1, ніж 0, тоді весь стовпець перетворюється до пікселів з інтенсивністю 1. На рис. 5 показано ідеальне зображення штрих-коду.

Рис. 5. Ідеальне зображення штрих-коду

Межа – це піксель, у якому відбувається раптова зміна інтенсивності. Створюється масив таких пікселів, який формує межі смужок зображення штрих-коду. Ширина смуги обчислюється шляхом віднімання послідовних елементів масиву вершин, що відповідають за межі. Ці ширини смуг надаються як вхідні дані для алгоритму декодування. [9]

Даний алгоритм локалізації та розпізнавання може бути застосований для будь-якого типу штрих-коду на вході. Подальші процеси бінаризації та декодування залежать від конкретного типу штрих-коду. Для прикладу розглянемо роботу алгоритму для штрих-коду Code 128, який використовується для багатьох документів посвідчення особи, зокрема і для студентських квитків.

Номер штрих-коду розшифровується за допомогою масиву значень ширини штрихів. Таким чином, кодування штрих-коду включає чорні та білі лінії, що чергуються, змінної ширини від 1 до 4. Код починається з Start-символу і закінчується End-символом, між якими розташовуються зашифровані символи, кожен представлений 6 смужками (3 чорними та 3 білими). Розшифровка вимагає виконання зчитування смуги відповідно до стандарту EAN-128. Наприклад, штрих-код містить необроблене повідомлення 211214122411112214221114221114134111131222233111 2. Перевіряємо перші 6 цифр, що відповідають першим 3 чорним смугам і 3 білим смугам. У нашому прикладі 211214 означають Start-символ для Code 128B, отже розшифрування буде відбуватися відповідно до цього стовпця таблиці символів Code 128. Розділяємо повідомлення на блоки по 6

символів і розшифровуємо їх. Послідовність 122411 за таблицею відповідатиме символу «h», 112214 – символу «e» і т.д. Повідомленням, що міститься в штрих-коді, є слово «hello».

Також цей масив може бути перетворений до послідовності формату

```
11010010000100110000101011001000011001010000110010100001000111101010001001100100011101011
```

Процес декодування полягатиме в тому, що масив розбивається на 11-бітні блоки і повідомлення отримується посимвольно, оскільки у штрих-коді Code 128 відповідно до кодування – типу A, B або C – набором з 11 бітів можуть бути літери або числа від 00 до 99. У представленій послідовності теж зашифровано слово «hello».

Висновки та перспективи подальшого дослідження. Використавши розглянуті адаптовані та модифіковані алгоритми можна реалізувати систему ідентифікації та обліку студентів навчального закладу за їх студентськими квитками, на яких міститься штрих-код, за допомогою камери. У штрих-коді зашифрований номер квитка, що складається лише з цифр. Після його зчитування та декодування за розшифрованим номером можна отримати інформацію про студента з існуючої бази даних та провести дії з запису певних даних, за якими будуть вестись статистика та формуватися звіти для керівництва.

Список бібліографічного опису

1. Roger C. Palmer. The Bar Code Book. Trafford Publishing, 2007.
2. Bhaskar Raj, Barcodes Technology and implementation, McGraw-Hill Education (India) Pvt Limited, 346 p.
3. Chai D, Hock F, "Locating and Decoding EAN-13 Barcodes from Images Captured by Digital Cameras," Information, Communications and Signal Processing, 2005 Fifth International Conference, pp.1595-1599.
4. Gallo, O., Manduchi, R., "Reading 1D Barcodes with Mobile Phones Using Deformable Templates," Pattern Analysis and Machine Intelligence, IEEE Transactions, vol.33, no.9, pp.1834,1843.
5. Luping Fang; Chao Xie, "1-D Barcode Localization in Complex Background," Computational Intelligence and Software Engineering (CiSE), 2010 International Conference, pp.1,3.
6. Xianyong Fang, FuLi Wu, Bin Luo, Haifeng Zhao and Peng Wang, "Automatic Recognition of Noisy Code-39 Barcode," Artificial Reality and Telexistence–Workshops, 2006. (ICAT '06), pp.79,82.
7. N. Otsu, "A Threesold Selection Method from Gray-Level Histograms" IEEE Trans. Systems, Man, and Cybernetics, vol.9, no.1, pp.62-66.
8. Thomas, V.M., "A Universal Code for Lifecycle Management of Products," Electronics & the Environment, Proceedings of the 2007 IEEE International Symposium, pp.180,183.
9. Sampada S. Upasani, Adarsh N. Khandate, Ankita U. Nikhare, Rupali A. Mange, R.V.Tornekar. Robust Algorithm for Developing Barcode Recognition System using Web-cam. International Journal of Scientific & Engineering Research, Volume 7, Issue 4, pp. 82-86.

References

1. Roger C. Palmer. The Bar Code Book. Trafford Publishing, 2007.
2. Bhaskar Raj, Barcodes Technology and implementation, McGraw-Hill Education (India) Pvt Limited, 346 p.
3. Chai D, Hock F, "Locating and Decoding EAN-13 Barcodes from Images Captured by Digital Cameras," Information, Communications and Signal Processing, 2005 Fifth International Conference, pp.1595-1599.
4. Gallo, O., Manduchi, R., "Reading 1D Barcodes with Mobile Phones Using Deformable Templates," Pattern Analysis and Machine Intelligence, IEEE Transactions, vol.33, no.9, pp.1834,1843.
5. Luping Fang; Chao Xie, "1-D Barcode Localization in Complex Background," Computational Intelligence and Software Engineering (CiSE), 2010 International Conference, pp.1,3.
6. Xianyong Fang, FuLi Wu, Bin Luo, Haifeng Zhao and Peng Wang, "Automatic Recognition of Noisy Code-39 Barcode," Artificial Reality and Telexistence–Workshops, 2006. (ICAT '06), pp.79,82.
7. N. Otsu, "A Threesold Selection Method from Gray-Level Histograms" IEEE Trans. Systems, Man, and Cybernetics, vol.9, no.1, pp.62-66.
8. Thomas, V.M., "A Universal Code for Lifecycle Management of Products," Electronics & the Environment, Proceedings of the 2007 IEEE International Symposium, pp.180,183.
9. Sampada S. Upasani, Adarsh N. Khandate, Ankita U. Nikhare, Rupali A. Mange, R.V.Tornekar. Robust Algorithm for Developing Barcode Recognition System using Web-cam. International Journal of Scientific & Engineering Research, Volume 7, Issue 4, pp. 82-86.

Додаток Б

Програмний код модуля OneDReader.java

```
package com.google.zxing.oned;

import com.google.zxing.BinaryBitmap;
import com.google.zxing.ChecksumException;
import com.google.zxing.DecodeHintType;
import com.google.zxing.FormatException;
import com.google.zxing.NotFoundException;
import com.google.zxing.Reader;
import com.google.zxing.ReaderException;
import com.google.zxing.Result;
import com.google.zxing.ResultMetadataType;
import com.google.zxing.ResultPoint;
import com.google.zxing.common.BitArray;

import java.util.Arrays;
import java.util.EnumMap;
import java.util.Map;


public abstract class OneDReader implements Reader {

  @Override
  public Result decode(BinaryBitmap image) throws
NotFoundException, FormatException {
    return decode(image, null);
  }

  // Note that we don't try rotation without the try harder flag,
even if rotation was supported.
  @Override
  public Result decode(BinaryBitmap image,
                       Map<DecodeHintType,?> hints) throws
NotFoundException, FormatException {
    try {
      return doDecode(image, hints);
    } catch (NotFoundException nfe) {
      boolean tryHarder = hints != null &&
hints.containsKey(DecodeHintType.TRY_HARDER);
      if (tryHarder && image.isRotateSupported()) {
        BinaryBitmap rotatedImage =
image.rotateCounterClockwise();
        Result result = doDecode(rotatedImage, hints);
        // Record that we found it rotated 90 degrees CCW / 270
degrees CW
        Map<ResultMetadataType,?> metadata =
result.getResultMetadata();
        int orientation = 270;
```

```
        if (metadata != null &&
metadata.containsKey(ResultMetadataType.ORIENTATION)) {
          // But if we found it reversed in doDecode(), add in
that result here:
          orientation = (orientation +
              (Integer)
metadata.get(ResultMetadataType.ORIENTATION)) % 360;
        }
        result.putMetadata(ResultMetadataType.ORIENTATION,
orientation);
        // Update result points
        ResultPoint[] points = result.getResultPoints();
        if (points != null) {
          int height = rotatedImage.getHeight();
          for (int i = 0; i < points.length; i++) {
            points[i] = new ResultPoint(height - points[i].getY()
- 1, points[i].getX());
          }
        }
        return result;
      } else {
        throw nfe;
      }
    }
  }

  @Override
  public void reset() {
    // do nothing
  }


  private Result doDecode(BinaryBitmap image,
                          Map<DecodeHintType,?> hints) throws
NotFoundException {
    int width = image.getWidth();
    int height = image.getHeight();
    BitArray row = new BitArray(width);

    boolean tryHarder = hints != null &&
hints.containsKey(DecodeHintType.TRY_HARDER);
    int rowStep = Math.max(1, height >> (tryHarder ? 8 : 5));
    int maxLines;
    if (tryHarder) {
      maxLines = height; // Look at the whole image, not just the
center
    } else {
      maxLines = 15; // 15 rows spaced 1/32 apart is roughly the
middle half of the image
    }

    int middle = height / 2;
    for (int x = 0; x < maxLines; x++) {
```

```
      // Scanning from the middle out. Determine which row we're
looking at next:
      int rowStepsAboveOrBelow = (x + 1) / 2;
      boolean isAbove = (x & 0x01) == 0; // i.e. is x even?
      int rowNumber = middle + rowStep * (isAbove ?
rowStepsAboveOrBelow : -rowStepsAboveOrBelow);
      if (rowNumber < 0 || rowNumber >= height) {
        // Oops, if we run off the top or bottom, stop
        break;
      }

      // Estimate black point for this row and load it:
      try {
        row = image.getBlackRow(rowNumber, row);
      } catch (NotFoundException ignored) {
        continue;
      }


      for (int attempt = 0; attempt < 2; attempt++) {
        if (attempt == 1) { // trying again?
          row.reverse(); // reverse the row and continue

          if (hints != null &&
hints.containsKey(DecodeHintType.NEED_RESULT_POINT_CALLBACK)) {
            Map<DecodeHintType,Object> newHints = new
EnumMap<>(DecodeHintType.class);
            newHints.putAll(hints);

newHints.remove(DecodeHintType.NEED_RESULT_POINT_CALLBACK);
            hints = newHints;
          }
        }
        try {
          // Look for a barcode
          Result result = decodeRow(rowNumber, row, hints);
          // We found our barcode
          if (attempt == 1) {
            // But it was upside down, so note that
            result.putMetadata(ResultMetadataType.ORIENTATION,
180);
            // And remember to flip the result points
horizontally.
            ResultPoint[] points = result.getResultPoints();
            if (points != null) {
              points[0] = new ResultPoint(width - points[0].getX()
- 1, points[0].getY());
              points[1] = new ResultPoint(width - points[1].getX()
- 1, points[1].getY());
            }
          }
          return result;
```

```
        } catch (ReaderException re) {
          // continue -- just couldn't decode this row
        }
      }
    }

    throw NotFoundException.getNotFoundInstance();
  }


  protected static void recordPattern(BitArray row,
                                      int start,
                                      int[] counters) throws
NotFoundException {
    int numCounters = counters.length;
    Arrays.fill(counters, 0, numCounters, 0);
    int end = row.getSize();
    if (start >= end) {
      throw NotFoundException.getNotFoundInstance();
    }
    boolean isWhite = !row.get(start);
    int counterPosition = 0;
    int i = start;
    while (i < end) {
      if (row.get(i) != isWhite) {
        counters[counterPosition]++;
      } else {
        if (++counterPosition == numCounters) {
          break;
        } else {
          counters[counterPosition] = 1;
          isWhite = !isWhite;
        }
      }
      i++;
    }

    if (!(counterPosition == numCounters || (counterPosition ==
numCounters - 1 && i == end))) {
      throw NotFoundException.getNotFoundInstance();
    }
  }

  protected static void recordPatternInReverse(BitArray row, int
start, int[] counters)
      throws NotFoundException {
    // This could be more efficient I guess
    int numTransitionsLeft = counters.length;
    boolean last = row.get(start);
    while (start > 0 && numTransitionsLeft >= 0) {
      if (row.get(--start) != last) {
        numTransitionsLeft--;
        last = !last;
```

```
      }
    }
    if (numTransitionsLeft >= 0) {
      throw NotFoundException.getNotFoundInstance();
    }
    recordPattern(row, start + 1, counters);
  }


  protected static float patternMatchVariance(int[] counters,
                                              int[] pattern,
                                              float
maxIndividualVariance) {
    int numCounters = counters.length;
    int total = 0;
    int patternLength = 0;
    for (int i = 0; i < numCounters; i++) {
      total += counters[i];
      patternLength += pattern[i];
    }
    if (total < patternLength) {
      // If we don't even have one pixel per unit of bar width,
assume this is too small
      // to reliably match, so fail:
      return Float.POSITIVE_INFINITY;
    }

    float unitBarWidth = (float) total / patternLength;
    maxIndividualVariance *= unitBarWidth;

    float totalVariance = 0.0f;
    for (int x = 0; x < numCounters; x++) {
      int counter = counters[x];
      float scaledPattern = pattern[x] * unitBarWidth;
      float variance = counter > scaledPattern ? counter -
scaledPattern : scaledPattern - counter;
      if (variance > maxIndividualVariance) {
        return Float.POSITIVE_INFINITY;
      }
      totalVariance += variance;
    }
    return totalVariance / total;
  }
  public abstract Result decodeRow(int rowNumber, BitArray row,
Map<DecodeHintType,?> hints)
      throws NotFoundException, ChecksumException,
FormatException;

}
```

Додаток В

Програмний код модуля Code128Reader.java

```
package com.google.zxing.oned;

import com.google.zxing.BarcodeFormat;
import com.google.zxing.ChecksumException;
import com.google.zxing.DecodeHintType;
import com.google.zxing.FormatException;
import com.google.zxing.NotFoundException;
import com.google.zxing.Result;
import com.google.zxing.ResultPoint;
import com.google.zxing.common.BitArray;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;

/**
 * <p>Decodes Code 128 barcodes.</p>
 *
 * @author Sean Owen
 */
public final class Code128Reader extends OneDReader {

  static final int[][] CODE_PATTERNS = {
      {2, 1, 2, 2, 2, 2}, // 0
      {2, 2, 2, 1, 2, 2},
      {2, 2, 2, 2, 2, 1},
      {1, 2, 1, 2, 2, 3},
      {1, 2, 1, 3, 2, 2},
      {1, 3, 1, 2, 2, 2}, // 5
      {1, 2, 2, 2, 1, 3},
      {1, 2, 2, 3, 1, 2},
      {1, 3, 2, 2, 1, 2},
      {2, 2, 1, 2, 1, 3},
      {2, 2, 1, 3, 1, 2}, // 10
      {2, 3, 1, 2, 1, 2},
      {1, 1, 2, 2, 3, 2},
      {1, 2, 2, 1, 3, 2},
      {1, 2, 2, 2, 3, 1},
      {1, 1, 3, 2, 2, 2}, // 15
      {1, 2, 3, 1, 2, 2},
      {1, 2, 3, 2, 2, 1},
      {2, 2, 3, 2, 1, 1},
      {2, 2, 1, 1, 3, 2},
      {2, 2, 1, 2, 3, 1}, // 20
      {2, 1, 3, 2, 1, 2},
      {2, 2, 3, 1, 1, 2},
      {3, 1, 2, 1, 3, 1},
```

```
{3, 1, 1, 2, 2, 2},
{3, 2, 1, 1, 2, 2}, // 25
{3, 2, 1, 2, 2, 1},
{3, 1, 2, 2, 1, 2},
{3, 2, 2, 1, 1, 2},
{3, 2, 2, 2, 1, 1},
{2, 1, 2, 1, 2, 3}, // 30
{2, 1, 2, 3, 2, 1},
{2, 3, 2, 1, 2, 1},
{1, 1, 1, 3, 2, 3},
{1, 3, 1, 1, 2, 3},
{1, 3, 1, 3, 2, 1}, // 35
{1, 1, 2, 3, 1, 3},
{1, 3, 2, 1, 1, 3},
{1, 3, 2, 3, 1, 1},
{2, 1, 1, 3, 1, 3},
{2, 3, 1, 1, 1, 3}, // 40
{2, 3, 1, 3, 1, 1},
{1, 1, 2, 1, 3, 3},
{1, 1, 2, 3, 3, 1},
{1, 3, 2, 1, 3, 1},
{1, 1, 3, 1, 2, 3}, // 45
{1, 1, 3, 3, 2, 1},
{1, 3, 3, 1, 2, 1},
{3, 1, 3, 1, 2, 1},
{2, 1, 1, 3, 3, 1},
{2, 3, 1, 1, 3, 1}, // 50
{2, 1, 3, 1, 1, 3},
{2, 1, 3, 3, 1, 1},
{2, 1, 3, 1, 3, 1},
{3, 1, 1, 1, 2, 3},
{3, 1, 1, 3, 2, 1}, // 55
{3, 3, 1, 1, 2, 1},
{3, 1, 2, 1, 1, 3},
{3, 1, 2, 3, 1, 1},
{3, 3, 2, 1, 1, 1},
{3, 1, 4, 1, 1, 1}, // 60
{2, 2, 1, 4, 1, 1},
{4, 3, 1, 1, 1, 1},
{1, 1, 1, 2, 2, 4},
{1, 1, 1, 4, 2, 2},
{1, 2, 1, 1, 2, 4}, // 65
{1, 2, 1, 4, 2, 1},
{1, 4, 1, 1, 2, 2},
{1, 4, 1, 2, 2, 1},
{1, 1, 2, 2, 1, 4},
{1, 1, 2, 4, 1, 2}, // 70
{1, 2, 2, 1, 1, 4},
{1, 2, 2, 4, 1, 1},
{1, 4, 2, 1, 1, 2},
{1, 4, 2, 2, 1, 1},
{2, 4, 1, 2, 1, 1}, // 75
{2, 2, 1, 1, 1, 4},
```

```
        {4, 1, 3, 1, 1, 1},
        {2, 4, 1, 1, 1, 2},
        {1, 3, 4, 1, 1, 1},
        {1, 1, 1, 2, 4, 2}, // 80
        {1, 2, 1, 1, 4, 2},
        {1, 2, 1, 2, 4, 1},
        {1, 1, 4, 2, 1, 2},
        {1, 2, 4, 1, 1, 2},
        {1, 2, 4, 2, 1, 1}, // 85
        {4, 1, 1, 2, 1, 2},
        {4, 2, 1, 1, 1, 2},
        {4, 2, 1, 2, 1, 1},
        {2, 1, 2, 1, 4, 1},
        {2, 1, 4, 1, 2, 1}, // 90
        {4, 1, 2, 1, 2, 1},
        {1, 1, 1, 1, 4, 3},
        {1, 1, 1, 3, 4, 1},
        {1, 3, 1, 1, 4, 1},
        {1, 1, 4, 1, 1, 3}, // 95
        {1, 1, 4, 3, 1, 1},
        {4, 1, 1, 1, 1, 3},
        {4, 1, 1, 3, 1, 1},
        {1, 1, 3, 1, 4, 1},
        {1, 1, 4, 1, 3, 1}, // 100
        {3, 1, 1, 1, 4, 1},
        {4, 1, 1, 1, 3, 1},
        {2, 1, 1, 4, 1, 2},
        {2, 1, 1, 2, 1, 4},
        {2, 1, 1, 2, 3, 2}, // 105
        {2, 3, 3, 1, 1, 1, 2}
    };

    private static final float MAX_AVG_VARIANCE = 0.25f;
    private static final float MAX_INDIVIDUAL_VARIANCE = 0.7f;

    private static final int CODE_SHIFT = 98;

    private static final int CODE_CODE_C = 99;
    private static final int CODE_CODE_B = 100;
    private static final int CODE_CODE_A = 101;

    private static final int CODE_FNC_1 = 102;
    private static final int CODE_FNC_2 = 97;
    private static final int CODE_FNC_3 = 96;
    private static final int CODE_FNC_4_A = 101;
    private static final int CODE_FNC_4_B = 100;

    private static final int CODE_START_A = 103;
    private static final int CODE_START_B = 104;
    private static final int CODE_START_C = 105;
    private static final int CODE_STOP = 106;
```

```
  private static int[] findStartPattern(BitArray row) throws
NotFoundException {
    int width = row.getSize();
    int rowOffset = row.getNextSet(0);

    int counterPosition = 0;
    int[] counters = new int[6];
    int patternStart = rowOffset;
    boolean isWhite = false;
    int patternLength = counters.length;

    for (int i = rowOffset; i < width; i++) {
      if (row.get(i) != isWhite) {
        counters[counterPosition]++;
      } else {
        if (counterPosition == patternLength - 1) {
          float bestVariance = MAX_AVG_VARIANCE;
          int bestMatch = -1;
          for (int startCode = CODE_START_A; startCode <=
CODE_START_C; startCode++) {
            float variance = patternMatchVariance(counters,
CODE_PATTERNS[startCode],
                MAX_INDIVIDUAL_VARIANCE);
            if (variance < bestVariance) {
              bestVariance = variance;
              bestMatch = startCode;
            }
          }
          // Look for whitespace before start pattern, >= 50% of
width of start pattern
          if (bestMatch >= 0 &&
              row.isRange(Math.max(0, patternStart - (i -
patternStart) / 2), patternStart, false)) {
            return new int[]{patternStart, i, bestMatch};
          }
          patternStart += counters[0] + counters[1];
          System.arraycopy(counters, 2, counters, 0,
counterPosition - 1);
          counters[counterPosition - 1] = 0;
          counters[counterPosition] = 0;
          counterPosition--;
        } else {
          counterPosition++;
        }
        counters[counterPosition] = 1;
        isWhite = !isWhite;
      }
    }
    throw NotFoundException.getNotFoundInstance();
  }

  private static int decodeCode(BitArray row, int[] counters, int
rowOffset)
```

```
      throws NotFoundException {
    recordPattern(row, rowOffset, counters);
    float bestVariance = MAX_AVG_VARIANCE; // worst variance we'll
accept
    int bestMatch = -1;
    for (int d = 0; d < CODE_PATTERNS.length; d++) {
      int[] pattern = CODE_PATTERNS[d];
      float variance = patternMatchVariance(counters, pattern,
MAX_INDIVIDUAL_VARIANCE);
      if (variance < bestVariance) {
        bestVariance = variance;
        bestMatch = d;
      }
    }
    // TODO We're overlooking the fact that the STOP pattern has 7
values, not 6.
    if (bestMatch >= 0) {
      return bestMatch;
    } else {
      throw NotFoundException.getNotFoundInstance();
    }
  }

  @Override
  public Result decodeRow(int rowNumber, BitArray row,
Map<DecodeHintType,?> hints)
      throws NotFoundException, FormatException, ChecksumException
{

    boolean convertFNC1 = hints != null &&
hints.containsKey(DecodeHintType.ASSUME_GS1);

    int[] startPatternInfo = findStartPattern(row);
    int startCode = startPatternInfo[2];

    List<Byte> rawCodes = new ArrayList<>(20);
    rawCodes.add((byte) startCode);

    int codeSet;
    switch (startCode) {
      case CODE_START_A:
        codeSet = CODE_CODE_A;
        break;
      case CODE_START_B:
        codeSet = CODE_CODE_B;
        break;
      case CODE_START_C:
        codeSet = CODE_CODE_C;
        break;
      default:
        throw FormatException.getFormatInstance();
    }
```

```
    boolean done = false;
    boolean isNextShifted = false;

    StringBuilder result = new StringBuilder(20);

    int lastStart = startPatternInfo[0];
    int nextStart = startPatternInfo[1];
    int[] counters = new int[6];

    int lastCode = 0;
    int code = 0;
    int checksumTotal = startCode;
    int multiplier = 0;
    boolean lastCharacterWasPrintable = true;
    boolean upperMode = false;
    boolean shiftUpperMode = false;

    while (!done) {

      boolean unshift = isNextShifted;
      isNextShifted = false;

      // Save off last code
      lastCode = code;

      // Decode another code from image
      code = decodeCode(row, counters, nextStart);

      rawCodes.add((byte) code);

      // Remember whether the last code was printable or not
(excluding CODE_STOP)
      if (code != CODE_STOP) {
        lastCharacterWasPrintable = true;
      }

      // Add to checksum computation (if not CODE_STOP of course)
      if (code != CODE_STOP) {
        multiplier++;
        checksumTotal += multiplier * code;
      }

      // Advance to where the next code will to start
      lastStart = nextStart;
      for (int counter : counters) {
        nextStart += counter;
      }

      // Take care of illegal start codes
      switch (code) {
        case CODE_START_A:
        case CODE_START_B:
        case CODE_START_C:
```

```
          throw FormatException.getFormatInstance();
        }

        switch (codeSet) {

          case CODE_CODE_A:
            if (code < 64) {
              if (shiftUpperMode == upperMode) {
                result.append((char) (' ' + code));
              } else {
                result.append((char) (' ' + code + 128));
              }
              shiftUpperMode = false;
            } else if (code < 96) {
              if (shiftUpperMode == upperMode) {
                result.append((char) (code - 64));
              } else {
                result.append((char) (code + 64));
              }
              shiftUpperMode = false;
            } else {
              // Don't let CODE_STOP, which always appears, affect
whether whether we think the last
              // code was printable or not.
              if (code != CODE_STOP) {
                lastCharacterWasPrintable = false;
              }
              switch (code) {
                case CODE_FNC_1:
                  if (convertFNC1) {
                    if (result.length() == 0) {
                      // GS1 specification 5.4.3.7. and 5.4.6.4. If
the first char after the start code
                      // is FNC1 then this is GS1-128. We add the
symbology identifier.
                      result.append("]C1");
                    } else {
                      // GS1 specification 5.4.7.5. Every subsequent
FNC1 is returned as ASCII 29 (GS)
                      result.append((char) 29);
                    }
                  }
                  break;
                case CODE_FNC_2:
                case CODE_FNC_3:
                  // do nothing?
                  break;
                case CODE_FNC_4_A:
                  if (!upperMode && shiftUpperMode) {
                    upperMode = true;
                    shiftUpperMode = false;
                  } else if (upperMode && shiftUpperMode) {
                    upperMode = false;
```

```
                shiftUpperMode = false;
              } else {
                shiftUpperMode = true;
              }
              break;
            case CODE_SHIFT:
              isNextShifted = true;
              codeSet = CODE_CODE_B;
              break;
            case CODE_CODE_B:
              codeSet = CODE_CODE_B;
              break;
            case CODE_CODE_C:
              codeSet = CODE_CODE_C;
              break;
            case CODE_STOP:
              done = true;
              break;
          }
        }
        break;
      case CODE_CODE_B:
        if (code < 96) {
          if (shiftUpperMode == upperMode) {
            result.append((char) (' ' + code));
          } else {
            result.append((char) (' ' + code + 128));
          }
          shiftUpperMode = false;
        } else {
          if (code != CODE_STOP) {
            lastCharacterWasPrintable = false;
          }
          switch (code) {
            case CODE_FNC_1:
              if (convertFNC1) {
                if (result.length() == 0) {
                  // GS1 specification 5.4.3.7. and 5.4.6.4. If
the first char after the start code
                  // is FNC1 then this is GS1-128. We add the
symbology identifier.
                  result.append("]C1");
                } else {
                  // GS1 specification 5.4.7.5. Every subsequent
FNC1 is returned as ASCII 29 (GS)
                  result.append((char) 29);
                }
              }
              break;
            case CODE_FNC_2:
            case CODE_FNC_3:
              // do nothing?
              break;
```

```
              case CODE_FNC_4_B:
                if (!upperMode && shiftUpperMode) {
                  upperMode = true;
                  shiftUpperMode = false;
                } else if (upperMode && shiftUpperMode) {
                  upperMode = false;
                  shiftUpperMode = false;
                } else {
                  shiftUpperMode = true;
                }
                break;
              case CODE_SHIFT:
                isNextShifted = true;
                codeSet = CODE_CODE_A;
                break;
              case CODE_CODE_A:
                codeSet = CODE_CODE_A;
                break;
              case CODE_CODE_C:
                codeSet = CODE_CODE_C;
                break;
              case CODE_STOP:
                done = true;
                break;
            }
          }
          break;
        case CODE_CODE_C:
          if (code < 100) {
            if (code < 10) {
              result.append('0');
            }
            result.append(code);
          } else {
            if (code != CODE_STOP) {
              lastCharacterWasPrintable = false;
            }
            switch (code) {
              case CODE_FNC_1:
                if (convertFNC1) {
                  if (result.length() == 0) {
                    // GS1 specification 5.4.3.7. and 5.4.6.4. If
the first char after the start code
                    // is FNC1 then this is GS1-128. We add the
symbology identifier.
                    result.append("]C1");
                  } else {
                    // GS1 specification 5.4.7.5. Every subsequent
FNC1 is returned as ASCII 29 (GS)
                    result.append((char) 29);
                  }
                }
                break;
```

```
              case CODE_CODE_A:
                codeSet = CODE_CODE_A;
                break;
              case CODE_CODE_B:
                codeSet = CODE_CODE_B;
                break;
              case CODE_STOP:
                done = true;
                break;
            }
          }
          break;
      }

      // Unshift back to another code set if we were shifted
      if (unshift) {
        codeSet = codeSet == CODE_CODE_A ? CODE_CODE_B :
CODE_CODE_A;
      }

    }

    int lastPatternSize = nextStart - lastStart;


    nextStart = row.getNextUnset(nextStart);
    if (!row.isRange(nextStart,
                     Math.min(row.getSize(), nextStart +
(nextStart - lastStart) / 2),
                     false)) {
      throw NotFoundException.getNotFoundInstance();
    }

    // Pull out from sum the value of the penultimate check code
    checksumTotal -= multiplier * lastCode;
    // lastCode is the checksum then:
    if (checksumTotal % 103 != lastCode) {
      throw ChecksumException.getChecksumInstance();
    }

    // Need to pull out the check digits from string
    int resultLength = result.length();
    if (resultLength == 0) {
      // false positive
      throw NotFoundException.getNotFoundInstance();
    }


    if (resultLength > 0 && lastCharacterWasPrintable) {
      if (codeSet == CODE_CODE_C) {
        result.delete(resultLength - 2, resultLength);
      } else {
        result.delete(resultLength - 1, resultLength);
```

```
      }
    }

    float left = (startPatternInfo[1] + startPatternInfo[0]) /
2.0f;
    float right = lastStart + lastPatternSize / 2.0f;

    int rawCodesSize = rawCodes.size();
    byte[] rawBytes = new byte[rawCodesSize];
    for (int i = 0; i < rawCodesSize; i++) {
      rawBytes[i] = rawCodes.get(i);
    }

    return new Result(
        result.toString(),
        rawBytes,
        new ResultPoint[]{
            new ResultPoint(left, rowNumber),
            new ResultPoint(right, rowNumber)},
        BarcodeFormat.CODE_128);

  }

}
```