Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Автоматики і комп'ютеризованих технологій
(повна назва)

Кафедра Комп'ютерно-інтегрованих технологій, автоматизації та робототехніки
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Розробка програмного модуля визначення траєкторії маніпулятора робота
(тема)

Виконав:
здобувач _2_ року навчання,
групи КІТПВм-24-1
Андрій ВОЛОВІК
(власне ім'я, прізвище)
Спеціальність 174 Автоматизація, комп'ютерно-інтегровані технології та робототехніка
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма: Комп'ютерно-інтегровані технологічні процеси і виробництва
(повна назва освітньої програми)
Керівник доц. Андрій ФРОЛОВ
(власне ім'я, прізвище)

Допускається до захисту

зав. кафедри КІТАР _____ Ігор НЕВЛЮДОВ
(підпис) (власне ім'я, прізвище)

2025 р.

Я, Воловік Андрій Володимирович, як здобувач вищої освіти ХНУРЕ, розумію і підтримую політику закладу із академічної доброчесності. Я не надавав і не одержував недозволену допомогу під час підготовки кваліфікаційної роботи. Я не використовував штучний інтелект для підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

15 грудня 2025 р.  Андрій ВОЛОВІК

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Факультет Автоматики і комп'ютеризованих технологій
Кафедра Комп'ютерно-інтегрованих технологій, автоматизації та робототехніки
Рівень вищої освіти другий (магістерський)
Спеціальність 174 Автоматизація, комп'ютерно-інтегровані технології та робототехніка
Тип програми Освітньо-професійна
Освітня програма Комп'ютерно-інтегровані технологічні процеси і виробництва

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
« __ » _____2025 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачу Воловіку Андрію Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка програмного модуля визначення траєкторії маніпулятора робота
Затверджена наказом по університету від 10.11.2025 р. No 1029 Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 24.12.2025 р.
3. Вихідні дані до роботи
3.1 Система технічного зору;
3.2 Середовище розробки Microsoft Visual Studio Professional 2017;
3.3 Бібліотека OpenCV;
3.4 Детектори контурів: Canny, Лапласа, Собеля;

4. Перелік питань, що потрібно опрацювати в роботі
4.1 Вступ;
4.2 Аналіз предметної області;
4.3 Аналіз зварювальних роботів з технічним зором;
4.4 Аналіз бібліотеки OpenCV;
4.5 Визначення контурів за допомогою детекторів;
4.6 Результати практичної частини;
4.6 Питання пов'язані з охороною праці;
4.7 Висновки.

5. Перелік графічного матеріалу із зазначенням креслень, схем, плакатів, комп'ютерних ілюстрацій Демонстраційний матеріал, представлений у форматі презентації PowerPoint (*.pptx). – с. формату А4.

6. Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз технічного завдання	01.09.25-14.09.25	Виконано
2	Опрацювання літератури за темою	15.09.25-3.10.25	Виконано
3	Аналіз бібліотеки OpenCV	5.10.25-17.10.25	Виконано
4	Пошук контурів електроду за допомогою детекторів	18.10.25-22.11.25	Виконано
5	Пошук піксельних координат	23.11.25-12.12.25	Виконано
6	Оформлення пояснювальної записки	15.12.25-16.12.25	Виконано
7	Подання роботи на нормоконтроль	17.12.25	
8	Подання роботи на перевірку Інтернет-сервісом StrikePlagiarism		
9	Подання роботи на рецензію		
10	Подання роботи на підпис зав. кафедри		
11	Подання атестаційної роботи в ЕК		

Дата видачі завдання 1.09.2025

Здобувач _____ Андрій ВОЛОВІК
(підпис) (прізвище, ініціали)

Керівник роботи _____ доц. Андрій ФРОЛОВ
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка містить: 72 с., 1 табл., 42 рис., 2 дод., 20 джерел.

ТЕХНІЧНИЙ ЗІР, ПРОМИСЛОВИЙ РОБОТ, ЕЛЕКТРОД, ПРОГРАМУВАННЯ, ВИЗНАЧЕННЯ КООРДИНАТ, КОНТУР, МАНІПУЛЯТОРИ.

Об’єкт дослідження – процес визначення координат маніпулятора промислового робота.

Предмет дослідження – програмна частина системи технічного зору за допомогою бібліотеки OpenCV.

Мета кваліфікаційної роботи – розробка програмного модуля визначення положення виконавчого елемента робота з використанням технічного зору.

У процесі дослідження розглянуто існуючі рішення в галузі технічного зору зварювальних роботів. Написано програмний код (мовою C++) роботи системи визначення координат та контурів електрода.

Розроблена система може використовуватися на виробництвах, де використовують зварювальні роботи. При зміні технічної частини, система також може використовуватися для різних цілей технічного та комп'ютерного зору.

ABSTRACT

The explanatory note contains: 72 pages, 1 tables, 42 figures, 2 application, 20 sources.

TECHNICAL VISION, INDUSTRIAL ROBOT, ELECTRODE, PROGRAMMING, COORDINATE DETERMINATION, CONTOUR, MANIPULATORS.

The object of research is the process of determining the coordinates of the manipulator of an industrial robot.

The subject of research is the software part of the vision system using the OpenCV library.

The purpose of the qualification work is to develop a software module for determining the position of the executive element of the robot using technical vision.

In the process of research, existing solutions in the field of technical vision of welding robots are considered. The program code is written in C++ of the system for determining the coordinates and contours of the electrode.

The developed system can be used in industries where welding work is used. When changing the technical part, the system can also be used for various purposes of technical and computer vision.

ЗМІСТ

СПИСОК СКОРОЧЕНЬ

РТК – роботизовані технологічні комплекси;

ЧПУ – числове програмне управління;

AVC – Automatic Voltage Control (автоматичне управління напругою);

CMOS – Complementary Metal-Oxide-Semiconductor (комплементарна структура метал-оксид-напівпровідник);

OpenCV – Open Source Computer Vision Library (бібліотека алгоритмів комп'ютерного зору);

USB – Universal Serial Bus (універсальна послідовна шина).

ВСТУП

В даний час з розвитком технологій в промисловості, виробництві і автоматизації продуктів для підвищення якості процесу використовуються системи технічного зору. Основними прикладами є обробка продукту, де потрібне візуальне детектування на основі зображення. Свою актуальність в даної сфері отримали промислові роботи. Роботизовані технологічні комплекси (РТК) вже успішно використовуються на великих підприємствах.

Усі дані роботи в основному закуповуються у великих компаній, що виробляють устаткування для промислової автоматизації. Внаслідок чого приватні компанії не мають повного доступу до програмної частини обладнання, що є недоліком при інтеграції з іншими видами програм.

Існують основні конкуруючі компанії у сфері виробництва промислових роботів: Fanuc Corporation та KUKA Roboter. Fanuc Corporation дуже популярна завдяки своїм контролерам R30-iA. Однією з особливостей даного контролера є інтегрована вбудована система iRVISION, яка дозволяє з допомогою системи візуального виявлення по технології plug&play розпізнавати та визначати місце розташування довільно розташованих виробів будь-якої форми та розміру. KUKA Roboter теж має ряд зварювальних маніпуляторів з системами технічного зору, але менш популярних ніж iRVISION.

Усі дані системи мають недолік – ускладнене інтегрування з іншими програмними засобами. У цій роботі з допомогою зображення електроду, отриманого з камери будемо визначати положення виконавчого елемента промислового робота.

Мета кваліфікаційної роботи – розробка програмного модуля визначення положення виконавчого елемента робота з використанням технічного зору.

Об’єкт дослідження – процес визначення координат маніпулятора

промислового робота.

Предмет дослідження – програмна частина системи технічного зору за допомогою бібліотеки OpenCV.

Для досягнення поставленої мети в роботі необхідно вирішити такі задачі:

– провести огляд існуючих рішень у галузі технічного зору зварювальних роботів;

– розробити технічну частину у вигляді конструкції експериментального макету;

– провести аналіз бібліотеки OpenCV, до якої входить як калібрування камери, так і підбір різних методів визначення контурів об'єкта;

– написати програмний код роботи системи визначення координат (мовою C++);

– провести експерименти для визначення ефективності роботи системи;

– оформити кваліфікаційну роботу згідно ДСТУ 3008:2015 [1], а також з методичними вказівками із підготовки та оформлення кваліфікаційних робіт здобувачів другого (магістерського) рівня спеціальності 174 «Комп’ютерно-інтегровані технологічні процеси і виробництва» [2].

Результати дослідження наведено у збірнику студентських наукових статей «Автоматизація та приладобудування» ADED-2025 [3].

Результати роботи відповідають Цілі сталого розвитку 9: «Промисловість, інновації та інфраструктура».

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд питань зварювання за допомогою маніпуляторів

Зварювання за допомогою маніпуляторів є повністю автоматизованою системою. Перевагами такого зварювання є висока якість готових виробів і висока продуктивність зварювального виробництва.

Основною перевагою зварювання за допомогою маніпулятора є висока точність позиціонування зварювального пальника. Однак бувають випадки, коли зварювальний робот не може знайти точку для зварювання як людина. Через це похибка правильного зварювання підвищується, що є неприйнятним. Для вирішення цієї проблеми використовують різні методи.

Якщо досягти заданої точності позиціонування неможливо, слід застосовувати методи корекції зварювальної траєкторії. Прикладом є використання лазерної системи стеження за стиком шва. Корекція траєкторій дозволить зберегти якість зварного виробу. При цьому її використання зменшить продуктивність аж до 30%.

Лідируючі позиції в розробці роботизованих зварювальних комплексів належать сьогодні компанії KUKA Roboter. Під цією маркою випускаються лідери в області електродуговий зварювання в середовищі захисного газу – роботи серії HW та роботи виду KUKA KR-16-2. Маючи вантажопідйомність до 16 кг і радіус дії до 2016 мм, ці маніпулятори виконують зварювання навіть важкодоступних з'єднань. Завдяки наявності шостий осі з можливістю нескінченного обертання, виключаються часові витрати на повернення в початкове положення, та скорочується час обробки деталі [4].

В якості прикладів розглянемо: роботизований осередок для зварювання KUKA KR-16-2, роботизований осередок для фрезерний обробки KUKA KR-120-R2700. KUKA KR-16-2 зображено рисунку 1.1. Роботизований осередок

для фрезерної обробки KUKA KR-120-R2700 зображений на рисунку 1.2 [4].

Рисунок 1.1 – Роботизований осередок для зварювання KUKA KR-16-2

Рисунок 1.2 – Роботизований осередок для фрезерної обробки KUKA KR-120-R2700

Основною проблемою для зварювального робота KUKA KR-16-2 є незручності роботи з електродом під час кожного запуску самого робота. У ході роботи потрібно вручну налаштовувати положення робота, оскільки

електрод зменшувався і тим самим потрібно було змінювати позиції робота. У ході вирішення проблеми було вирішено використати технічний зір, що дозволило б визначати піксельні координати зображення електроду. Після знаходження піксельних координат потрібно було знайти залежність між рештою координат. У результаті основний проблемою залишалася знайти самі координати зображення електрода, а також знайти залежність між світовими координатами, координатами камери та координатами зображення.

1.2 Використовувані програмні засоби

1.2.1 Visual Studio Professional 2017

Microsoft Visual Studio Professional 2017 – середовище розробки, призначене для розробки різних комп'ютерних і не лише програм. За допомогою цієї програми можна створювати також веб-сайти, мобільні програми та інше. Крім цього, можна створювати код і взаємодіяти його з іншими. Також може створювати як власний код, і керований код [5].

Програма включає вихідний код, а також відладчик вихідного коду. Тут не потрібно додатково встановлювати компілятор, т.к. він вбудований у саму програму. Visual Studio дуже зручний в плані кодингу, тому що містить для цього все рівні. Містить рядок для виводу помилок, а також коректор коду. Крім цього містить різні редактори, як для графічного інтерфейсу, так і для веб-додатків.

Visual Studio підтримує 36 різних мов програмування і дозволяє редактору коду та відладчику підтримувати (в тій чи іншій мірі) практично будь-яку мову програмування, якщо існує специфічна для мови служба. Вбудовані мови включають в себе C, C ++, C ++ / CLI, Visual Basic .NET, C#, F#, JavaScript, TypeScript, XML, XSLT, HTML та CSS. Також підтримує інші мови, такі як Python, Ruby, Node.js і Java [5].

Основною перевагою даної програми є її доступність та можливість інтегрування з бібліотекою OpenCV. Крім того, підтримується мова

програмування C++, яка теж інтегрований з OpenCV. Завдяки бібліотекам OpenCV ми отримаємо доступ до технічного зору, який відіграє основну роль нашому проекті.

За технічною частиною та сумісністю з персональним комп'ютером програма займає невелику частину пам'яті, що є доступним в сучасний час. Зручність даної програми також полягає в тому, що всі мови програмування знаходяться разом і можна використовувати їх у будь-який момент. Адже в інших випадках для кожної мови треба було б встановлювати все окремо.

1.2.2 Бібліотека Open Source Computer Vision

OpenCV (Open Source Computer Vision Library) є бібліотекою для технічного зору з відкритим вихідним кодом, а також бібліотекутеку для машинного навчання. OpenCV було створено як додаток для технічного зору, а також для зростання інтересу суспільства до технічного зору. Ця бібліотека є повністю безкоштовною та доступною в мережі, а також спрощує процес ознайомлення з технічним зором [6].

Бібліотека містить близько 2500 алгоритмів, призначених для сучасного комп'ютерного зору та машинного навчання. Ці алгоритми можна використовувати як для роботи звичайних камер, так і для більш професійних. Алгоритми містять роботу з камерами для ідентифікації об'єктів, а також для ряду інших робіт з технічним зором. Декільком прикладом їх використання: вилучення 3D-моделей об'єктів, створення 3D-хмар точок зі стереокамер, зшивання зображень разом для отримання високої роздільної здатності зображення всієї сцени, знаходження зображення з бази даних, згладжування і покращення зображень, можливість стеження за рухами очей, розпізнавання декорації та створення маркерів, щоб накладати їх на доповнену реальність тощо. Бібліотека широко використовується в компаніях, дослідницьких групах, навчальних закладах та урядових органах [6].

Відомі компанії на весь світ типу Google, Microsoft, Intel та інші

використовують цю бібліотеку. Існує багато стартапів, таких як Applied Minds, VideoSurf та Zeitera, які широко використовують бібліотеку OpenCV.

Бібліотека має інтерфейси C++, Python, Java, MATLAB і підтримує Windows, Linux, Android та Mac OS. OpenCV написаний мовою C++, що дуже зручно у нашому випадку. Офіційний сайт, де можна завантажити бібліотеку, представлено на рисунку 1.3.

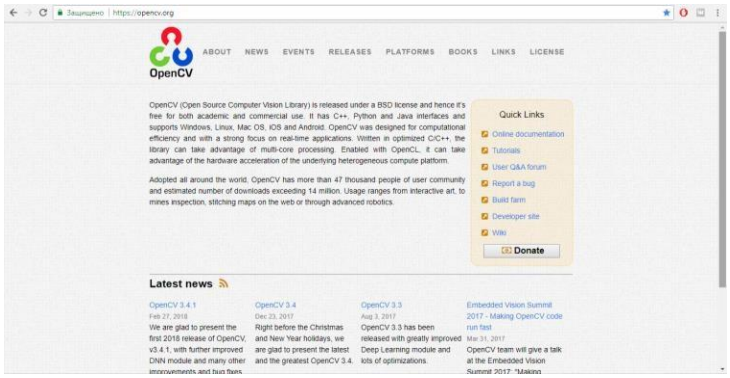

Рисунок 1.3 – Офіційний сайт OpenCV

2 ЗВАРЮВАЛЬНІ РОБОТИ З ТЕХНІЧНИМ ЗІРОМ

2.1 Приклади зварювальних роботів з технічним зором

Нині популярним серед промислових роботів є роботизоване зварювання. Зварювальні роботи дозволяють оптимізувати весь процес зварювання. Як говорилося вище, це дуже зручно і вигідно, але існують і свої недоліки. Основними є робота саме при зварюванні будь-якої деталі. У звичайного маніпулятора точність позиціонування падає при зміщенні шва, де людина в свою чергу могла б здогадатися, але не робот. У великих компаніях існують різноманітні рішення подібних ситуацій. Надалі будуть наведені приклади зварювальних роботів із технічним зором.

2.1.1 Зварювальний робот компанії Fanuc Corporation

Fanuc Corporation один з лідерів виробників обладнання для автоматизації. Компанія дуже популярна і має багато представництв по всьому світу. Основними напрямками є: ЧПУ, промислової роботи та лазерне обладнання.

Дуже популярна компанія завдяки серії своїх зварювальних роботів із технічним зором iRVision. Маніпулятор з технічним зором iRVision компанії Fanuc Corporation представлений рисунку 2.1 і рисунку 2.2 [7].

Розглянемо деякі функціональні можливості системи iRVision, що використовуються при роботизованому зварюванні [8].

1. VISION SHIFT застосовується в контактному та дуговому зварюванні для компенсації похибки між позиціонуванням деталі-шаблону, по якої проводилося програмування робота, та позиціонуванням наступних деталей, на яких повинна виконуватися програма.

Рисунок 2.1 – Маніпулятор Fanuc Corporation з вертикальною камерою

Рисунок 2.2 – Маніпулятор Fanuc Corporation зі вбудованою камерою

Можливості VISION SHIFT:

– функція використовує можливості технічного зору для автоматичного 3D калібрування програми робота;

– можна автоматично калібрувати робочу точку інструменту, що раніше могло бути зроблено лише вручну;

– на високоточне неконтактне калібрування програм замість кількох годин витрачається всього до 30 хвилин. Таким чином, ця функція значно скорочує час запуску;

– підтримує встановлення центральної точки інструменту та кінчика електрода для керування роботом з великою точністю;

– можливо застосовувати для калібрування груп спільно працюючих роботів (Dual-, Triple-, і Quadarm);

– дозволяє робити юстування роботів FANUC після зміни двигунів, енкодерів і т.д.

2. FANUC iRVISION Weld Tip служить для візуального контролю та перевірки стану робочого кінця електрода для точкового зварювання на предмет пошкоджень. Це спеціалізоване програмне забезпечення може виконувати самі різноманітні перевірки, включаючи вимір зносу знімного кінця електрода та визначення, чи здійснювалася його заміна або заправка.

3. FANIC iRVISION Torch Mate оснащено можливостями програмування на виробництві та промисловою камерою. Воно запобігає виникненню проблем з вирівнюванням за рахунок утримання центральної точки інструменту на траєкторії руху інструменту. Система iRVISION забезпечує стабільну якість зварювання, а також збільшує час безвідмовний роботи за рахунок усунення необхідності у випереджувальних діях.

4. 2D Multiview Vision – технічний зір з використанням кількох камер для визначення точного положення деталі. Дозволяє точно визначити положення деталей великих габаритів.

Контролер R30-iA у складі програмного забезпечення має програму Arctool, розроблену для спрощення і стандартизації установки та управління зварювальними додатками. Розглянемо деякі функції додатки:

– weaving – зварювання з коливаннями. Забезпечує коливальний рух пальника щодо лінії зварювання, що збільшує необхідну товщину шва;

– weld Resume – при зупинці робота під час руху по зварювальній доріжці і переміщенні назад до доріжки встановлюється необхідна відстань і зварювання триває;

– захист електрода – компенсує значення положення кінчика електрода дротом після довгого часу зварювання або заміни кінчика електрода. У цих випадках зварювальні операції не можуть виконуватися коректно. Захист електрода допомагає запобігти збою центральної точки інструменту, інакше її потрібно перезаписати;

– automatic adjustment of the tool center point – функція автоматичного відновлення пальника забезпечує автоматичне регулювання центральної точки інструменту. Автоматичне відновлення електрода автоматично компенсує вигин зварювального електрода та знос наконечника, зменшуючи кількість дефектів та збільшуючи продуктивність;

– torch guard collision detection – параметри для зап'ястя дуже чутливі до зіткнень електрода. Функція зупиняє робот до того моменту, як електрод, зіткнувшись із перешкодою, зламається;

– arc sensor TAST – (у момент руху по шву) метод контролю шва, який використовується для контролю положення поверхні деталі, що дуже важливо для роботизованого дугового зварювання. Цей метод використовується в якості датчика зварювального струму у разі зміни довжини дуги. Робот компенсуватиме зміну положення зварювального пальника;

– контроль положення пальники TAST дозволяє контролювати положення електрода за рахунок зміни зварювального струму при амплітудному русі. Програма визначає центр шва, порівнюючи ліву і праву половини по струму. Також можна контролювати висоту положення електрода. Положення електрода задано (відповідно до значення струму) і зберігається по середині амплітуди;

– touch sensing складається з: а) руху центральної точки інструменту до об'єкту, використовуючи контроль руху робота, швидкість і напрямок; б) використання вхідних сигналів для визначення початку контакту робота з об'єктом; в) збереження положень об'єкта у позиційні регістри; г) використання інформації про збережені положення роботом. Touch Sensing

визначає положення відхилення зварювальних швів в момент зварювання з допомогою Arc Sensor TAST. Touch sensing дозволяє роботу змінювати напрямок доріжки автоматично при русі в випадку, якщо поверхня деталі неточна, або деталь займає некоректне становище;

– опція Automatic Voltage Control (AVC) компенсує зміну форми деталі та відхилення під час зварювального процесу TIG (зварювання вольфрамовим електродом в інертному газі) або інших процесів при постійному струмі, наприклад, при виконанні плазмового різання. AVC компенсує різницю між оптимальними та реальними зварювальними параметрами під час руху по зварювальної доріжці. Функція AVC демонструє напругу при дуговому зварюванні TIG процесу. Вона дозволяє змінювати висоту пальника над деталлю для збереження постійної зварювальної напруги [7].

2.1.2 Зварювальний робот компанії ABB Robotics

ABB Robotics – це компанія, виробляє промислове обладнання. У тому числі роботизовані комплекси. Хоч і компанія була започаткована в 1988 році, але вже є одним з лідерів у світі. У компанії багато різних маніпуляторів: для складання та розбирання, палеттування, точкового та дугового зварювання, лакування і фарбування, а також інших. Одним з роботів є маніпулятор для електродугового зварювання IRB 1520ID, високопродуктивний робот із вбудованим зварювальним обладнанням, розроблений спеціально для електродугового зварювання. Всі кабелі та канали, що подають, прокладені всередині маніпулятора, що спрощує програмування і забезпечує оптимальний захист для всіх елементів. IRB 1520ID представлений рисунку 2.3 [9].

Рисунок 2.3 – Маніпулятор для електродугового зварювання компанії ABB Robotics

У компанії ведуться дослідження технічного зору. Одним із прикладів є робот Roberta. Роботи на ім'я Roberta були спеціально розроблені для маленьких і середніх підприємств, які хочуть досягти ефективної промислової автоматизації. Метою розробників було створення гнучких, легких та ергономічних роботів, які можна легко переміщати по цеху [9].

Основні Характеристики Roberta:

– легка вага;

– 6 ступенів свободи без точки сингулярності;

– велика вантажопідйомність.

У роботах Roberta представлені кілька інтегрованих концепції безпеки. Roberta може працювати з певним захватом, який є абсолютно безпечним для людини. Спеціальні датчики, ще один аспект безпеки, який знижує ризик будь-яких пошкоджень. Робот також оснащений технічним зором. Тобто він може бачити об'єкт, з яким працює. Серія Roberta доступна в 3 різних моделях з вантажопідйомністю 4,0 кг, 8,0 кг та 12,0 кг. Відмінності між моделями – у максимальній досяжності та вантажопідйомності.

2.1.3 Зварювальний робот компанії KUKA Roboter

Промислові роботи Kuka – одні з популярних роботів зварювання у світі.

Вони з успіхом замінюють ручну працю на багатьох підприємствах. Для зварювальних робіт у компанії є моделі високої вантажопідйомності, а також зварювальні роботи малої вантажопідйомності. Перерахуємо деякі моделі зварювальних роботів Kuka [10]:

– модель KR 5 arc рекомендують, коли потрібні відносно прості роботи для зварювання (дугового), паяльних робіт, фарбування та інших маніпуляцій. Її корисне навантаження дорівнює 5 кг, додаткове – 12 кг, а максимальний радіус дії вимірюється 1412 мм. Модель зображено рисунку 2.4;

Рисунок 2.4 – Модель KR 5 arc

– роботизований комплекс для зварювання в середовищі захисних газів дозволяє працювати навіть з дуже складними деталями великих розмірів, також на великий глибини деталей. Для подачі газу використовують різні комплекти шлангів. Це модель KR 5-2 arc HW з тими ж параметрами ваги і робочого радіуса, що і у попередньої. Такі роботи можуть кріпитися і на підлозі, і на стелі, ефективно виконуючи пайку та зварювання, в том числі і електродугову. Сюди ж відноситься KR 16 L8 arc HW [10];

– KR 16 L8 arc HW входить до серії L разом з KR 16 L6-2. Подовжений маніпулятор дозволяє збільшити робочу зону до 2016 мм і 1911 мм відповідно, але вантажопідйомність знижується (8 кг та 6 кг);

– цілий комплекс робіт може виконувати багатофункціональний промисловий робот, представлений моделями KR 6-2 та KR 16-2 (у каталозі Kuka позначаються як KR 6-3 та KR 16-3), які можуть налаштовуватися під десятки різних операцій. Перша модель має корисну навантаженням 6 кг та додатковою – 10 кг, з максимальним радіусом у 1611 мм. Друга модель має такі самі межі додаткового навантаження, але корисне складає 16 кг, а робоча зона – 1610 мм. Модель KR 16-2 представлений рисунку 1.1;

– для швидкісних робіт, де потрібна посилена потужність, створений робот KR 16-3 S, у якого час такту скорочено на 18%. Він має вантажопідйомність 16 кг і досяжність 1611 мм.

Також є інші моделі з різними можливостями. Для машин малої вантажопідйомності маса вантажу може бути від 5 кг до 16 кг, а максимальне завантаження в межах 36-48 кг. Всі ці роботи можуть бути оснащені камерами для виконання робіт, пов'язаних із технічним зором. Компанія KUKA Roboter має низку рішень у цій галузі, але ці рішення не знаходяться у відкритому доступі.

2.2 Розробка експериментального макету

У ході вирішення задачі були вивчені можливі рішення у статтях, пов'язаних з технічним зором, а точніше з системами для визначення об'єктів та їх координат. Але всі такі рішення мали велику ціну.

Для визначення координат електрода потрібно місце для ноого і для маніпулятора. У зв'язку з цим була придумана експериментальна конструкція, яка б дозволила робити знімки електрода маніпулятора. Г-подібна конструкція представлена на рисунку 2.5.

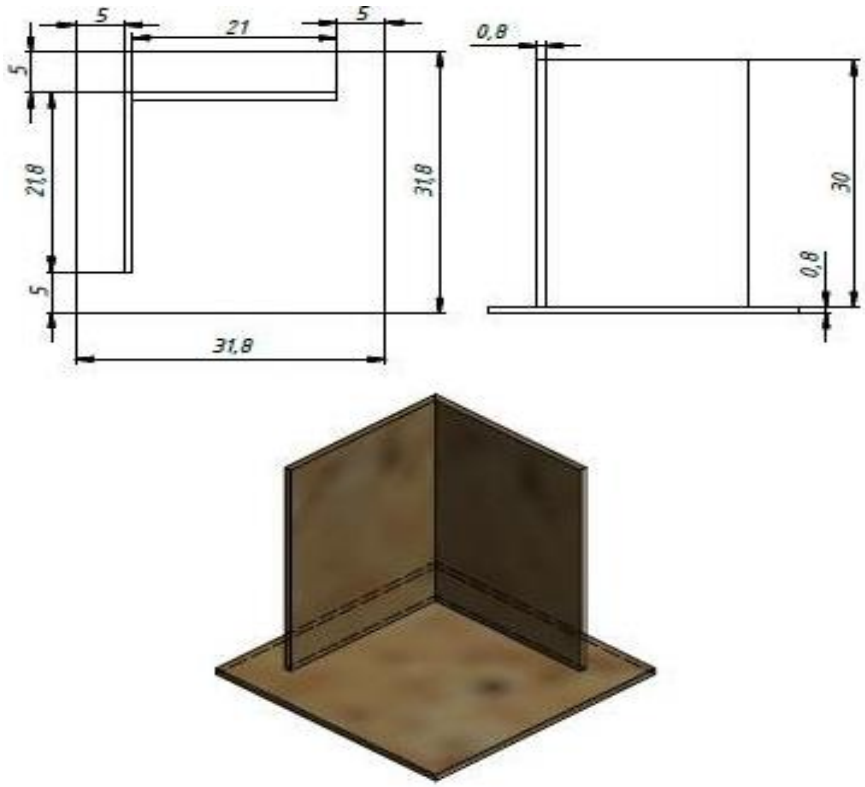

Рисунок 2.5 – Г-образна експериментальна конструкція, спроектована в Autodesk Inventor

Дана конструкція була змодельована в програмі Autodesk Inventor, нижня частина конструкції розміром 31,8 см на 31,8 см. Розміри для листів взяті під А4: 21,8 см і 31,8 відповідно. Для кожного листа взятий запас міцності: в 0,8 см для одного, 2,1 см для другого. Це пов'язано з тим, що при кріпленні листи перетинаються один з одним.

У технічну частину також входять 2 веб-камери Microsoft «LifeCam Cinema» 6CH-00002. Дані камери дуже чутливі до світла і мають згинальну частину для кріплення. Технічні характеристики веб-камери Microsoft «LifeCam Cinema» 6CH-00002 представлені в таблиці 2.1. Камеру зображено на рисунку 2.6 [11].

Таблиця 2.1 – Технічні характеристики камери Microsoft «LifeCam Cinema»

Підтримувана роздільна здатність відео	1280×720
Максимальна частота кадрів	30 Гц
Тип підключення	USB 2.0
Розміщення	кріплення на РК-монітор
Тип матриці	CMOS
Число мегапікселів матриці	0,9
Підтримувана роздільна здатність фотографії	1280×960
Фокусування	автоматична фокусування
Конектори і інтерфейси	USB тип A штекер (USB 2.0)
Матеріал корпусу	пластик
Розміри (ш×в×г), мм	46×40×56
Вага, г	95

Рисунок 2.6 – Веб-камера Microsoft «LifeCam Cinema» 6CH-00002

У ході роботи початкове положення електрода передбачається встановлювати вручну. Після повинна йти робота з налаштуванням камер та їх калібрування, камери підключені до ноутбука. У самому ноутбуці проводиться налагодження програмної частини мовою C++. Дані, отримані з камер зберігаються в кореневій папці. Після кожного знімка старе зображення

видаляється і замінюється новим знімком електрода. Паралельно можна змінювати положення камер для зручності знімків. Якщо не вдається настроїти за розміром зображення, отримані з камери, їх можна обрізати за допомогою вбудованих команд бібліотеки OpenCV.

Робота програмної частини повинна починатися після всіх налаштувань камер, а також після встановлення камер у нерухоме положення. У разі невиконання налаштувань програмна частина буде мати похибки при знаходженні контурів та координат електрода.

3 АНАЛІЗ БІБЛІОТЕКИ OPENCV

3.1 Знайомство з бібліотекою OpenCV

OpenCV – бібліотека комп'ютерного зору з відкритим вихідним кодом. Бібліотека написана на C, C++ і працює на комп'ютерах під керуванням Linux, Windows, Mac OS X. Також активно розвиваються інтерфейси бібліотеки для Python, Ruby, Matlab та інших мов програмування.

Бібліотека OpenCV була розроблена з метою підвищення обчислювальної ефективності та з ухилом на додатки реального часу. OpenCV написана за допомогою оптимізованого C і може використовувати багатоядерні процесори. Надалі, якщо буде потрібна автоматична оптимізація на апаратних платформах Intel, існує можливість купівлі бібліотеки IPP (Integrated Perfomance Primitives), яка складається з процедур із низькорівневою оптимізацією для різних алгоритмічних областей. OpenCV автоматично використовуватиме бібліотеку IPP під час виконання програми [6].

Однією з основних цілей OpenCV є надання простого у використанні інтерфейсу, який дозволить людям досить швидко будувати складні програми, що використовують комп'ютерне зору. Бібліотека OpenCV містить понад 500 функцій, які охоплюють багато сфер комп'ютерного зору, такі як: інспекція фабричної продукції, медицина, безпека, інтерфейс користувача, калібрування камери, стереозір і робототехніка. І все це завдяки тому, що комп'ютерний зір і машинне навчання часто йдуть "пліч-о-пліч", до того ж OpenCV повністю включає бібліотеку загального призначення MLL (Machine Learning Library). MLL бібліотека орієнтована на розпізнавання статичних образів та кластеризацію. MLL дуже корисна для завдань комп'ютерного зору, які становлять основу OpenCV, але вона досить узагальнена, щоб вирішувати

конкретні проблеми машинного навчання. Архітектура самої OpenCV зображена рисунку 3.1.

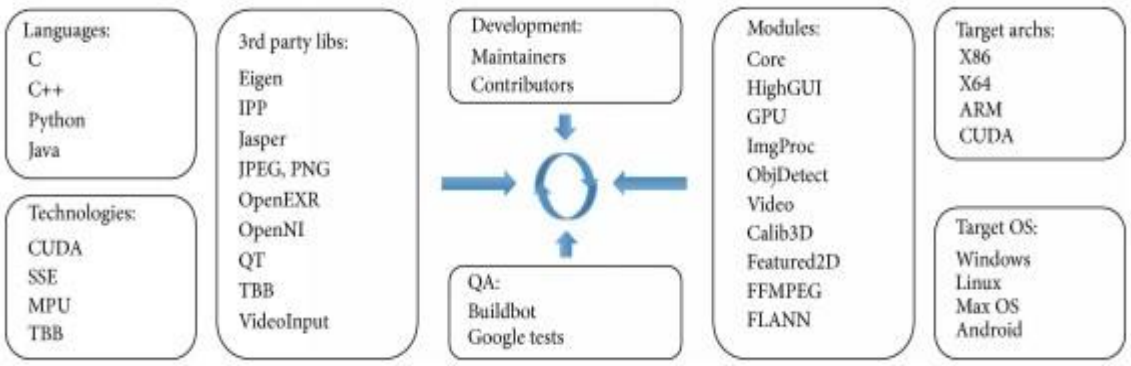

Рисунок 3.1 – Архітектура OpenCV

Основні модулі бібліотеки можна віднести до 4 груп (розділів):

– модулі Core, HighGUI, що реалізують базову функціональність (базові структури, математичні функції, генератори випадкових чисел, лінійна алгебра, швидке перетворення Фур'є, введення/виведення зображень і відео, введення/виведення у форматах XML, YAML та ін.);

– модулі ImgProc, Features2D для обробки зображень (фільтрація, геометричні перетворення, перетворення колірних просторів, сегментація, виявлення особливих точок та ребер, контурний аналіз та ін.);

– модулі Video, ObjDetect, Calib3D (калібрування камери, аналіз руху та відстеження об'єктів, обчислення положення у просторі, побудова карти глибини, детектування об'єктів, оптичний потік);

– модуль ML, що реалізує алгоритми машинного навчання (метод найближчих сусідів, наївний класифікатор байесу, дерева рішень, бустинг, градієнтний бустинг дерев рішень, випадковий ліс, машина опорних векторів, нейронні мережі та ін.). На рисунку 3.2 представлена загальна схема типового додатка, призначеного на вирішення тієї чи іншої завдання комп'ютерного зору. Надалі розберемо деякі можливості бібліотеки [7].

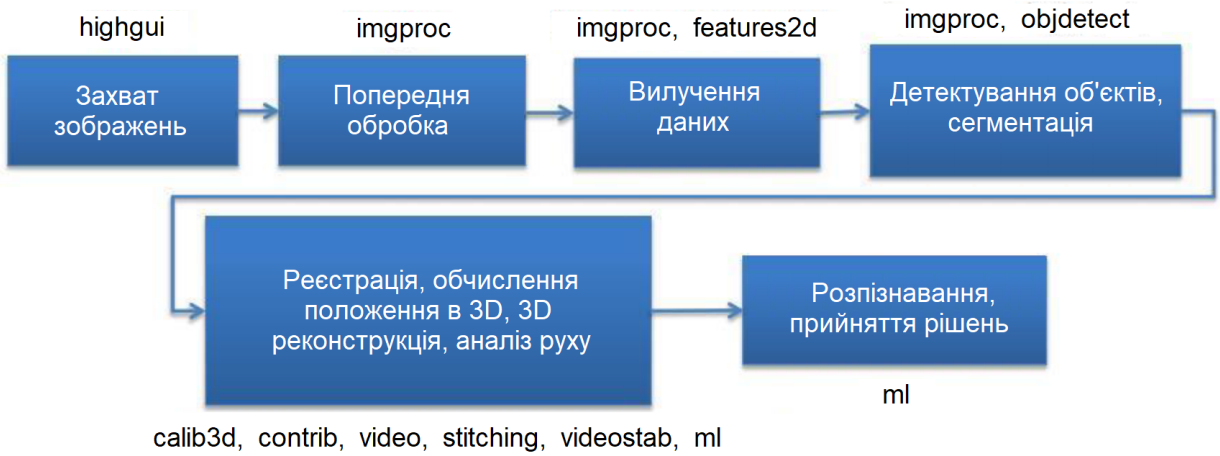

Рисунок 3.2 – Архітектура OpenCV

OpenCV дозволяє читати різні типи зображень, а також відео з камери. Всі ці можливі завдяки компоненту HighGUI, який включений до OpenCV. Надалі частина цієї програми буде використана для відкриття зображення [12]. Код для завантаження зображення та відображення його на екрані (рисунок 3.3).

```
#include "opencv\highgui.h"
int main ( int argc, char ** argv )
{
IplImage* img = cvLoadImage( argv [ 1 ]); // Ім'я зображення
cvNamedWindow( "electrode" , CV_WINDOW_AUTOSIZE); // Створюємо вікно зі
стандартним розміром
cvShowImage( " electrode" , img); // Зображення показується
cvWaitKey( 0 ); // Очікування
cvReleaseImage(& img ); // Звільнення пам'яті
     cvDestroyWindow( "electrode" ); // Видалення вікна
     }
```

Рисунок 3.3 – Зображення електрода

Ця програма просто завантажує зображення. Тепер спробуємо переглянути роботу програми з камерою. Код увімкнення камери та знімка електрода (рисунок 3.4).

```
//завантаження бібліотек
#include <opencv2\opencv.hpp>
#include <locale.h >
using namespace cv;
using namespace std;
int main( int , char **)
{
      // знімок з камери
      vector<int> wr_params;
      wr_params.push_back( cv:: IMWRITE_JPEG_QUALITY);
      wr_params.push_back(50);
      VideoCapture cap(2);
      if (!cap.isOpened())
            return -1;
      Mat edges;
      Mat frame;
      cap >> frame;
      //збереження знімка
     imwrite( "electrode_left.jpg" , frame, wr_params);
      //Показ зображення
      imshow( "electrode_left" , frame);
```

Рисунок 3.4 – Знімок електрода камерою

У ході вивчення роботи з бібліотекою було реалізовано різні варіанти перетворення зображення. Одним із варіантів було зрізання частини зображення. Якщо зображення мало зайве тло, його можна було обрізати. У результаті залишалося лише робоча зона. Код обрізки та зображення представлений нижче є продовженням попереднього.

```
Mat img0 = imread( "electrode_left.jpg", 1);
int x = 325,
y = 25,
width = 200,
height = 455;
img0 = img0 (Rect (x, y, width, height));
imshow("electrode_left" , frame);
```

Цей код дозволяє обрізати зображення вздовж координати, яку вибираєш. Обрізка відбувається за висотою та шириною зображення. Далі отримане зображення перезберігається, а старе замінюється обрізаним. Обрізане зображення електрода рисунка 3.4 показано рисунку 3.5 [7].

Рисунок 3.5 – Обрізаний знімок електрода

3.2 Калібрування камери

Перед початком використання, камера повинна бути відкалібрована. Калібрування дозволяє врахувати спотворення, що вносяться оптичною системою.

Калібрування камери зводиться до отримання внутрішніх та зовнішніх параметрів камери за наявними фотографіями або відео, отриманими з її допомогою. Калібрування камери зазвичай провадиться на початковому етапі вирішення багатьох задач комп'ютерного зору. Крім того, ця процедура дозволяє виправити дисторсію на фотографіях та відео.

Як правило, для представлення 2D-координат точки на площині використовується вектор-стовпчик виду $[u\ v\ 1]^T$, а для завдання положення 3D-точки в світових координатах – $[x_w\ y_w\ z_w\ 1]^T$. Слід зазначити, що ці вирази записані в розширеній нотації однорідних координат. Зокрема, в моделі камери-обскури матриця камери використовується для проектування точок тривимірного простору на площину зображення [13]

$$z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = A[R \quad T] \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

де Z_c – довільний масштабний коефіцієнт.

Параметри внутрішнього калібрування [13]

$$A = \begin{bmatrix} \alpha_x & \gamma & u_0 \\ 0 & \alpha_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

Матриця внутрішнього калібрування A містить 5 значущих параметрів. Ці параметри відповідають фокусній відстані, куту нахилу пікселів та принциповій точці. Зокрема, α_x та α_y відповідають фокусній відстані, виміряній у ширині та висоті пікселя, u_0 і v_0 – координатам принципової точки, а, $\gamma = \alpha_y \cdot tan\,\varphi$, де φ – кут нахилу пікселя. Нелінійні параметри внутрішнього калібрування, такі як коефіцієнти дисторсії, також мають важливе значення, хоча і не можуть бути включені в лінійну модель, що описується матрицею внутрішнього калібрування. Більшість сучасних алгоритмів калібрування камери визначає їх разом із параметрами лінійної частини моделі. Параметри внутрішнього калібрування відносяться лише до камери, але не до сцени, тому вони змінюються лише тоді, коли змінюються відповідні параметри камери.

Параметри зовнішнього калібрування R, T (де R – вектор 3×1 або матриця 3×3 повороту, T – вектор 3×1 переносу) визначають перетворення координат точок сцени зі світової в систему координат камери. Параметри зовнішнього калібрування пов'язані безпосередньо зі сценою, що фотографується, тому (на відміну від параметрів внутрішнього калібрування) кожній фотографії відповідає свій набір цих параметрів. У даній роботі буде реалізовано лише калібрування внутрішніх параметрів камери.

Під час зйомки світло зі сцени фокусується та захоплюється камерою. Цей процес зменшує кількість вимірювань даних, одержуваних камерою, з трьох до двох (світло з тривимірної сцени перетворюється на двовимірне зображення). Тому кожен піксель на отриманому зображенні відповідає променю світла вихідної сцени і відповідно під час калібрування камери відбувається пошук відповідності між тривимірними точками сцени і пікселями зображення.

У випадку ідеальної камери-обскури для такої відповідності достатньо однієї матриці проекції. Однак у разі складніших камер, спотворення, що вносяться лінзами, можуть сильно вплинути на результат. Таким чином, функція проектування набуває більш складного вигляду і часто записується як послідовність перетворень, наприклад:

$$x = I \cdot Dist(E \cdot X),$$

де $X = [x_w\ y_w\ z_w\ 1]^T$ – координати вихідної точки сцени;

$x = [u\ v\ 1]$ – координати пікселя на зображенні;

$E = \begin{bmatrix} R & T \\ 0_3^T & 1 \end{bmatrix}$ – матриця зовнішнього калібрування, де R – матриця повороту 3×3, T – вектор перенесення 3×1;

Dist – функція застосування дісторсії;

$I = \begin{bmatrix} \alpha_x & \gamma & u_0 \\ 0 & \alpha_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}$ – матриця внутрішнього калібрування.

Існує кілька різних підходів до вирішення задачі калібрування.

1. Класичний підхід – алгоритм Roger Y. Tsai [14], що складається з двох етапів, на першому визначаються параметри зовнішнього калібрування, на другому – внутрішнього калібрування та дисторсії.

2. «Гнучке калібрування», запропоноване в роботі [15] і засноване на використанні плоского калібрувального об'єкта у вигляді шахівниці (рисунок 3.6).

3. Автоматичне калібрування – отримання калібрувальних даних безпосередньо за зображеннями, без використання спеціальних калібрувальних об'єктів [16].

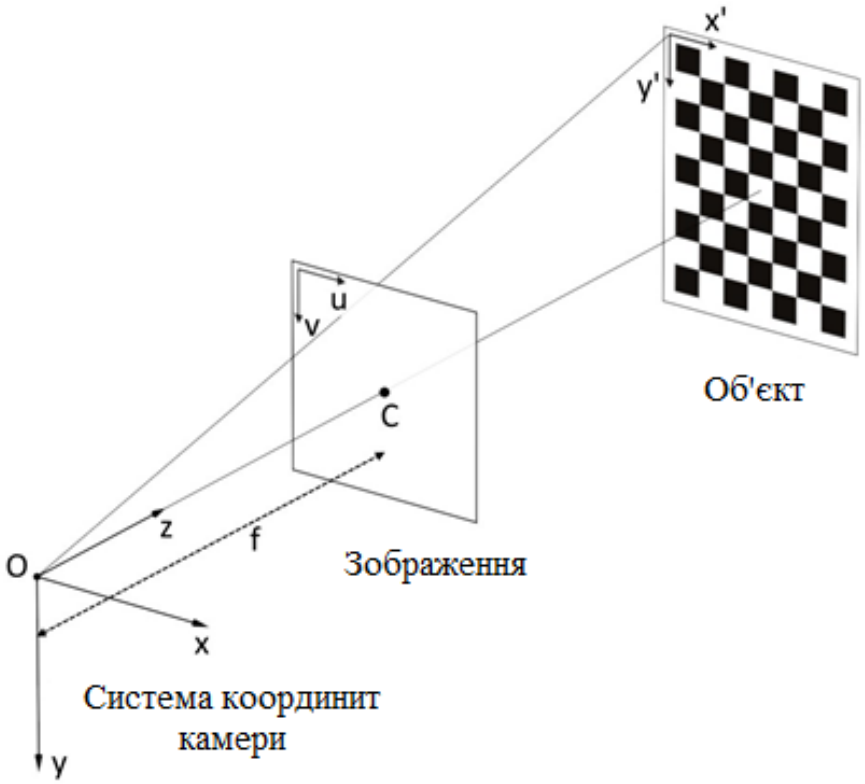

Рисунок 3.6 – Калібрування на основі шахівниці

У даній роботі використовується «гнучке калібрування», в силу простоти реалізації. Використання шахової дошки можливе завдяки тому, що відомо, які саме в неї точки можна подивитися, в які точки вони перейшли на зображенні.

Якщо багато разів показати цей шаблон камері, можна побачити, куди проектуються точки, отже, можна встановити ці параметри.

В результаті виникає задача оптимізації. Відомі тривимірні координати кутів (можна виміряти розміри квадрата і записати тривимірні координати шахівниці), можна продетектувати ці куточки на зображенні – це стандартна задача. Таким чином, буде відповідність між тривимірною точкою та двовимірною точкою. І потрібно буде знайти такі параметри, щоб тривимірні

точки переходили саме у ці двовимірні точки. Тобто потрібно оптимізувати параметри камери так, щоб виходила правильна проекція. Для калібрування потрібно показувати дошку під різними кутами (рисунок 3.7), повертаючи, її потрібно показати у всіх частинах камери для того, щоб надійно встановити її параметри. Тому що якщо показувати шаблон в одній і тій же позиції в одному кутку, то добре буде відомо, як камера проектує цей кут, але у всій іншій частині зображення все може бути дуже погано.

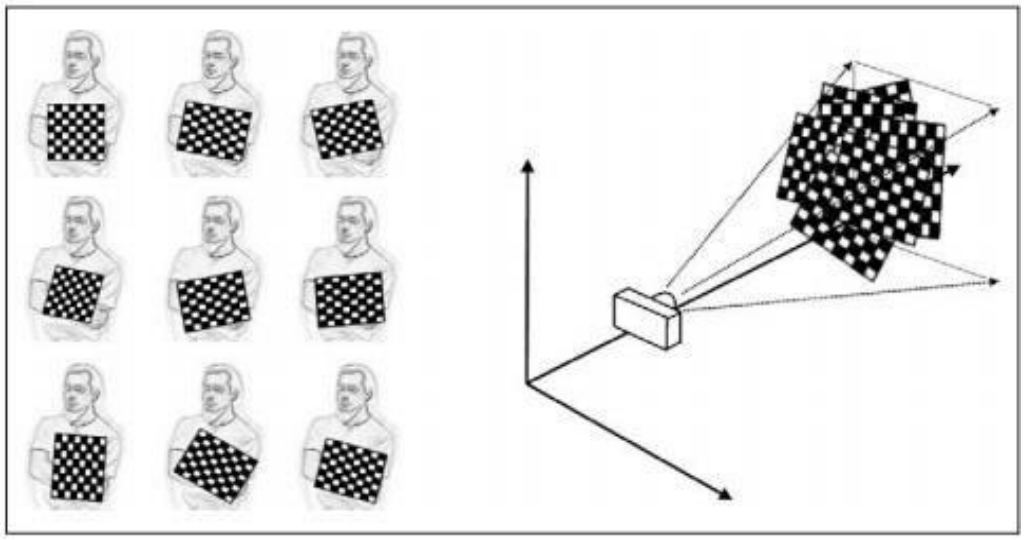

Рисунок 3.7 – Процес калібрування камери

Крім шахової дошки можна використовувати і інші шаблони – наприклад, шаблон із намальованих кіл, він показує більш точні результати, тому що центр кола можна знайти з більшою точністю, ніж кут шахової дошки. Відповідно, калібрація виходить точнішою. Можна використовувати складніші шаблони, наприклад, тривимірні.

У звичайних умовах достатньо одного калібрування для однієї камери (виробники камер надають технічні дані необхідні для калібрування). Проблема в тому, що для однієї моделі ці параметри трошки відрізнятимуться. Можна надати якісь параметри за замовчуванням і такі камери існують, для яких вони пораховані – для моделі. Але для конкретної камери цієї моделі

вони можуть трохи відрізнятися. Тому що, якби вони не відрізнялися, це означало б, що матриця розташована абсолютно на тому самому місці, абсолютно такі ж лінзи. Насправді таке неможливо. Завжди будуть невеликі відхилення, отже параметри калібрації теж будуть трохи іншими. Тому для більш точних параметрів потрібно робити калібрування самому. Можна намагатися робити автокалібровку – знімати камерою звичайне зображення – без шаблонів шахівниці, і при цьому намагатися зрозуміти, які у неї внутрішні параметри. Автоматично, без шаблонів, намагатись зрозуміти, як вона спотворює. Але на практиці виходить, що це іноді працює, іноді це не працює. В алгоритмі автокалібровки є кілька параметрів – ці параметри потрібно підбирати для того, щоб запустити алгоритм, тому що за одних параметрів алгоритм спрацює добре, за інших спрацює погано. У ситуації ж із використанням шаблонів із значно більшою надійністю можна отримати хорошу калібрацію камери.

Як було сказано раніше, камери-обскури призводять до великих спотворень зображень. Двома основними спотвореннями є тангенціальні спотворення (а) та радіальні спотворення (в) (рисунок 3.8).

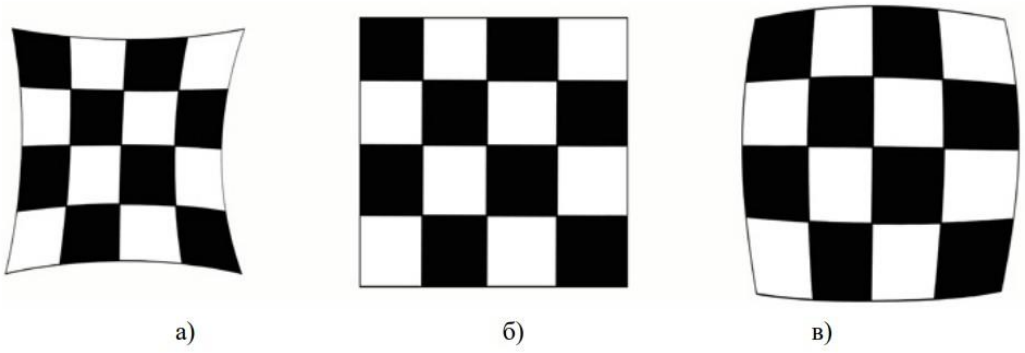

Рисунок 3.8 – Зображення шахівниці: а) танценціальне спотворення; б) неспотворене; в) радіальне спотворення

Через радіальне спотворення прямі лінії будуть вигнутими. Цей ефект збільшуватиметься з віддаленням від центру зображення. Наприклад, на

рисунку 3.9, де два краї шахівниці відзначені червоними лініями. Але можна побачити, що межа не є прямою і не відповідає червоному рядку. Усі очікувані прямі лінії вигнуті [17].

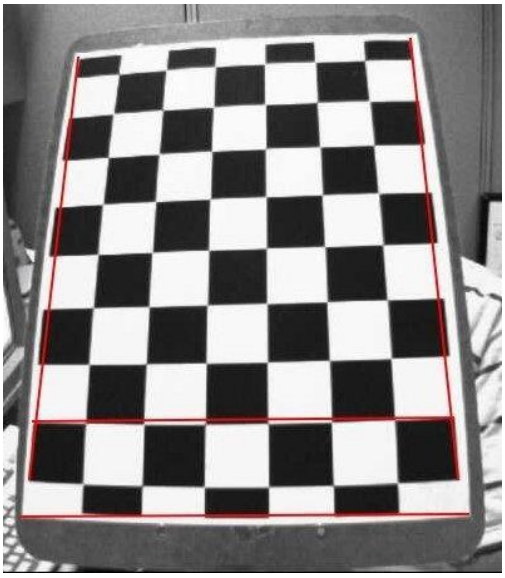

Рисунок 3.9 – Радіальне спотворення на шахівниці

Це спотворення вирішується так:

$$X_{corrected} = x\,(1 + k_1 \cdot r^2 + k_2 \cdot r^4 + k_3 \cdot r^6),$$
$$Y_{corrected} = y\,(1 + k_1 \cdot r^2 + k_2 \cdot r^4 + k_3 \cdot r^6)\,.$$

Іншим спотворенням є тангенціальне спотворення, яке відбувається, тому що зйомка зображення не вирівнюється ідеально паралельно площині зображення. Тому деякі області зображення можуть виглядати ближчими, ніж очікувалося. Воно вирішується так:

$$X_{corrected} = x + [2p_1 \cdot x \cdot y + p_2 \cdot (r^2 + 2x^2)]\,,$$
$$Y_{corrected} = yx + [2p_2 \cdot x \cdot y + p_1 \cdot (r^2 + 2y^2)]\,.$$

У результаті потрібно знайти п'ять параметрів, відомих як коефіцієнти спотворення, що визначаються:

$$Distortion\ coefficients = (k_1 k_2, p_1 p_2, k_3)\ .$$

На додаток до цього нам потрібно знайти ще деяку інформацію, наприклад, внутрішні та зовнішні параметри камери. Внутрішні характеристики специфічні для камери. Вони включають таку інформацію, як фокусна відстань $f_x\ f_y$, оптичні центри $c_x\ c_y$ і т. д. Їх також називають матрицею камери. Це залежить лише від камери, тому після обчислення вона може зберігатися для майбутніх цілей. Вона виражається як матриця 3×3:

$$I = \begin{bmatrix} \alpha_x & \gamma & u_0 \\ 0 & \alpha_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Зовнішні параметри відповідають векторам обертання та трансляції, що переводять координати тривимірної точки в систему координат.

Для того щоб коректно вимірювати ці спотворення необхідно спочатку скоригувати. Щоб знайти всі ці параметри, потрібно надати кілька зразкових зображень чітко визначеного шаблону (наприклад, шахівниці).

Як згадано вище, нам потрібно щонайменше кілька тестових шаблонів для калібрування камери. Для досягнення найкращих результатів, як радить сайт розробників бібліотеки OpenCV, необхідно зробити 10 тестових шаблонів. У результаті для камери №1 було зроблено п'ятнадцять шаблонів (рисунок 3.10), а для камери №2 десять (рисунок 3.11).

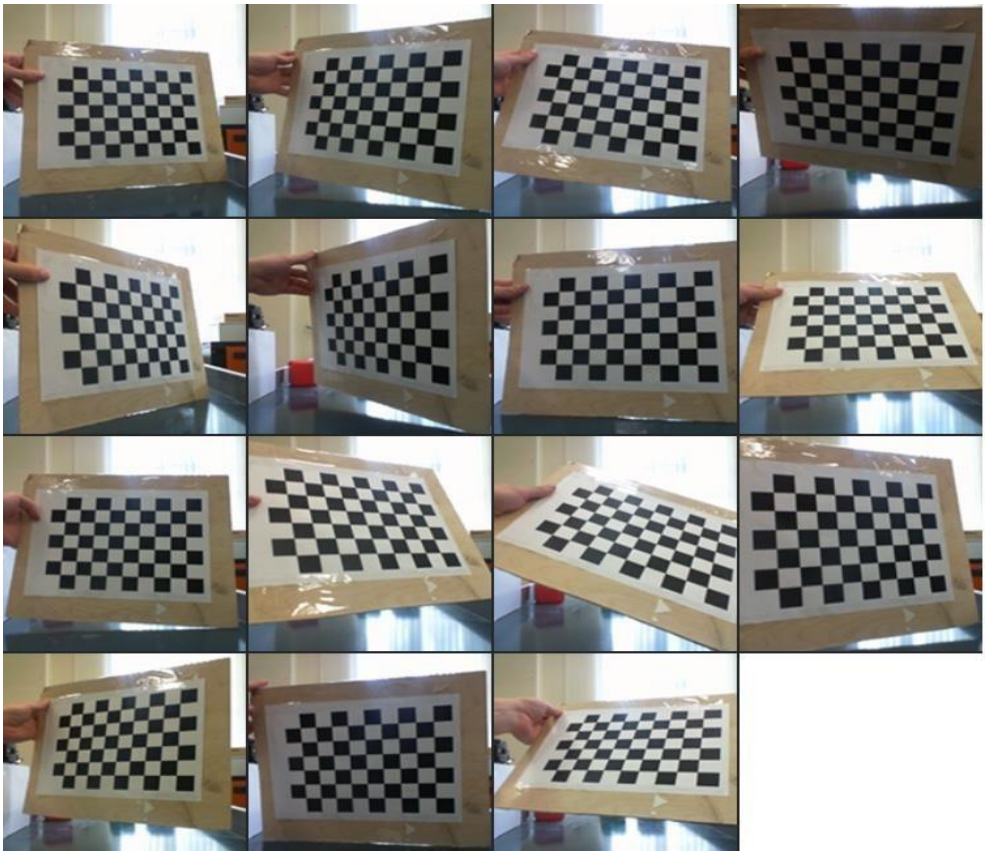

Рисунок 3.10 – Тестові шаблони для камери №1

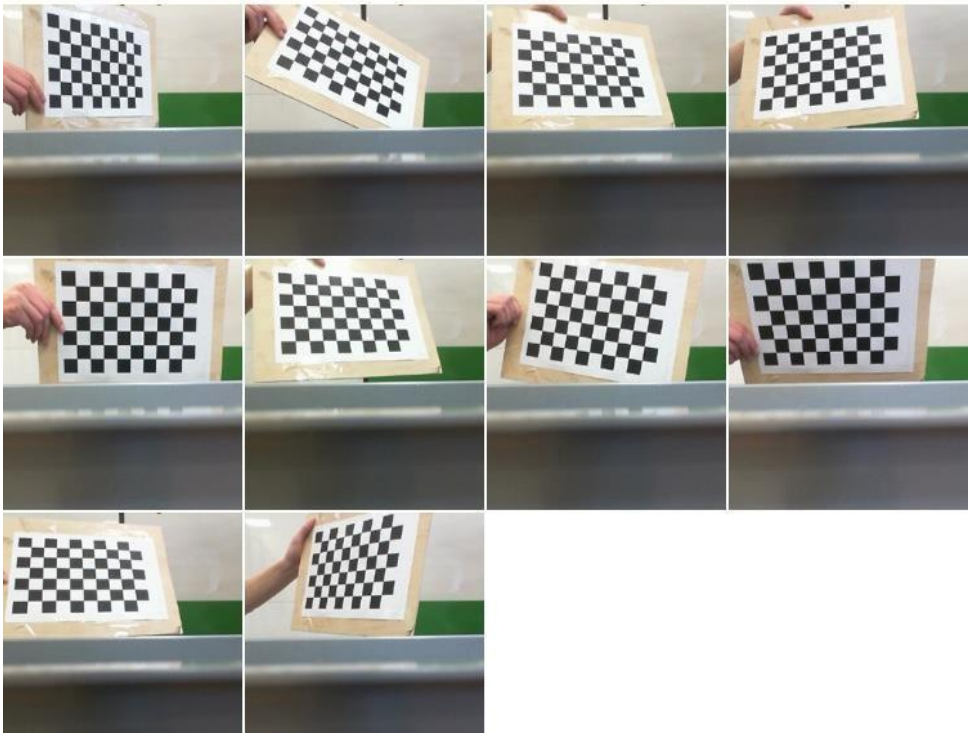

Рисунок 3.11 – Тестові шаблони для камери №2

Важливими вхідними даними, необхідними для калібрування камери, є набір тривимірних точок реального світу і відповідні їм точки 2D-зображення. 3D-точки називаються об'єктними точками, а точки 2D-зображення називаються точками зображення.

Отже, щоб знайти шаблон на шахівниці, ми використовуємо функцію cv2.findChessboardCorners (). Нам також потрібно встановити, який тип шахової сітки використовується. В даній роботі використовується сітка розмірністю 6×9 (зазвичай шахова дошка має 8×8 квадратів та 7×7 внутрішніх кутів). Ця функція повертає кутові точки, якщо шаблон був успішно знайдений. Ці кути будуть розміщені у порядку (зліва направо, зверху донизу). Ця функція може виявитися не в змозі знайти потрібний шаблон у всіх зображеннях. Тому потрібно по черзі перевіряти кожен шаблон. Як тільки кути знайдені, можна підвищити точність їхнього відображення, використовуючи cv2.cornerSubPix(). Після цього можемо нарисувати їх на шаблоні, використовуючи cv2.drawChessboardCorners(). Всі ці кроки включені в код нижче.

```
#include <iostream>
#include <cstdlib>
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
cv::VideoCapture m_camera;
cv::Mat m_currentFrame;
cv::Size m_cameraResolution;
std::vector<std::vector<cv::Point2f> > m_detectedPoints;
cv::Mat m_cameraMatrix( 3, 3, CV_64FC1 );
cv::Mat m_distortionCoefficients( 5,1,CV_64FC1 );
cv::Mat m_R;
cv::Mat m_T;
cv::Mat m_grayImage;
std::vector<std::vector<cv::Point3f> > m_realPoints;
size_t m_framesGrabed( 0 );
const cv::Size patternSize( 7,7 );
m_camera.open( CV_CAP_ANY );
// зберігаємо роздільну здатність камери
```

```
m_cameraResolution = cv::Size( ( int ) m_camera.get(
CV_CAP_PROP_FRAME_WIDTH ), ( int ) m_camera.get(
CV_CAP_PROP_FRAME_HEIGHT ) );
// отримуємо поточну картинку з камери
m_camera >> m_currentFrame;
// шукаємо шахівницю на зображенні
if( cv::findChessboardCorners( m_currentFrame, patternSize, m_detect-
edPoints[m_framesGrabed] ) ) {
  cv::cvtColor( m_currentFrame, m_grayImage, CV_BGR2GRAY );
// перевіряємо координати кутів дошки
cv::cornerSubPix( m_grayImage, m_detectedPoints[m_framesGrabed],
                     cv::Size( 11,11 ), cv::Size( -1, -1 ),
cv::TermCrite-ria(cv::TermCriteria::EPS | cv::TermCriteria::MAX_ITER,
30, 0.1));
     cv::drawChessboardCorners( m_currentFrame, patternSize, m_detect-
edPoints[m_framesGrabed], true );
     ++ m_framesGrabed;
}
std::vector<cv::Mat> rvecs(m_realPoints.size());
std::vector<cv::Mat> tvecs(m_realPoints.size());
m_distortionCoefficients = cv::Scalar( 0 );
m_cameraMatrix=cv::Scalar( 0 );
int flags = cv::CALIB_FIX_K3;
// початкові параметри камери
m_cameraMatrix.at( 0, 0 ) = m_initialFocusValue;
m_cameraMatrix.at( 0, 2 ) = m_cameraResolution.width >> 1;
m_cameraMatrix.at( 1, 1 ) = m_initialFocusValue;
m_cameraMatrix.at( 1, 2 ) = m_cameraResolution.height >> 1;
m_cameraMatrix.at( 2, 2 ) = 1.0;
flags |= cv::CALIB_USE_INTRINSIC_GUESS;
// виконуємо калібрування
cv::calibrateCamera( m_realPoints, m_detectedPoints,
m_cameraResolution, m_cameraMa-trix, m_distortionCoefficients, rvecs,
tvecs, flags );
m_R = rvecs[0];
  m_T = tvecs[0];
```

Один із використовуваних шаблонів з успішно знайденими кутами показаний на рисунку 3.12.

Рисунок 3.12 – Калібрувальний шаблон із знайденими кутами

Отже, тепер у нас є точки об'єкта та точки зображення, які були необхідні для калібрування камери. У бібліотеці OpenCV для калібрування використовується функція cv2.calibrateCamera(). Вона повертає матрицю камери, коефіцієнти спотворення, вектори обертання та трансляції тощо.

```
cv :: calibrateCamera ( m_realPoints, m_detectedPoints,
m_cameraResolution, m_cameraMatrix, m_distortionCoefficients, rvecs,
tvecs, flags ) ;
```

Після отримання параметрів калібрування можна отримати неспотворене зображення. Але до цього спочатку потрібно уточнити матрицю камери на основі вільного параметра масштабування, використовуючи cv2.getOptimalNewCameraMatrix ().

Функція обчислює та повертає оптимальну нову матрицю камери на основі параметра вільного масштабування (alpha). Змінюючи цей параметр, можна отримати лише чутливі пікселі alpha = 0, зберегти всі вихідні пікселі зображення, якщо є цінна інформація у кутах alpha = 1 або отримати щось середнє між ними. Коли α > 0, неспотворений результат, ймовірно, матиме

деякі чорні пікселі, що відповідають «віртуальним» пікселям за межами захопленого спотвореного зображення.

Одне з отриманих неспотворених зображень представлене рисунку 3.13.

Рисунок 3.13 – Неспотворене зображення шахівниці

У результаті можна побачити, що це грані прямі.

Точність вимірювання параметрів камери (коефіцієнти дісторсії, матриця камери) визначається середньою величиною помилки перепроецування (ReEr, Reprojection Error). Вона зображена на рисунку 3.14 і являє собою відстань (у пікселях) між проекцією P' на площину зображення точки P на поверхні об'єкта, і проекцією P" цієї точки P, побудованої після усунення дісторсії з використанням параметрів камери.

Ця помилка має бути якомога ближче до нуля. Враховуючи матриці внутрішнього спотворення, обертання та трансляції, спочатку потрібно перетворити точку об'єкта на точку зображення, використовуючи cv2.projectPoints (). Потім ми обчислюємо абсолютну норму між тим, що ми отримали з нашим перетворенням та алгоритмом пошуку кутів. Щоб знайти середню помилку, ми обчислюємо середнє арифметичне помилок для всіх калібрувальних зображень.

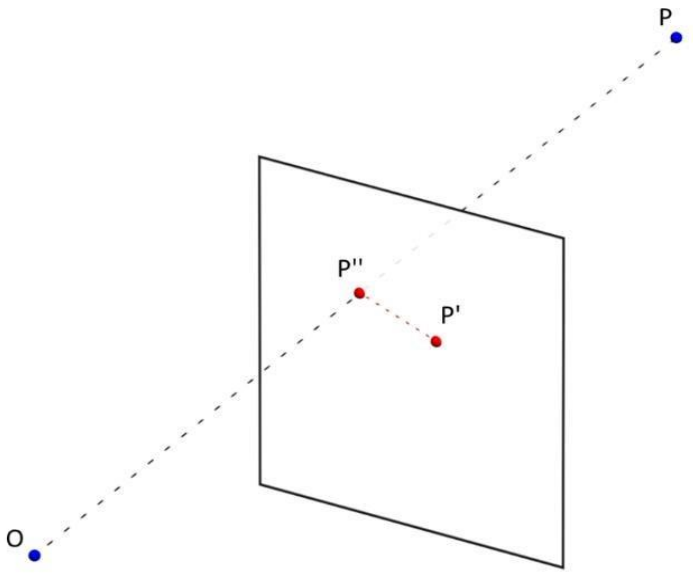

Рисунок 3.14 – Подання параметрів камери у вигляді площин

Тепер можна зберігати матрицю камери та коефіцієнти спотворення, використовуючи функції запису в Numpy (np.savez, np.savetxt і т.д.) для використання їх надалі. Це означає, що внутрішні параметри камери не потрібно щоразу перераховувати, для певної моделі вони постійні.

3.3 Визначення контурів за допомогою детектора меж Canny

Детектор меж Canny названий на честь його винахідника Джона Ф. Канні, який вигадав алгоритм у 1986 році. Алгоритм набагато більш затребуваний, ніж інші, і вважається одним із популярних методів. Основні принципи, якими керує сенсор Canny, залишаються постійними. Це означає, що ми будемо використовувати значення градієнта як індикатор того, належить піксель до області потенційного краю чи ні. Для розуміння того, як працює детектор меж Canny наведено наступні кроки [7]:

– неагресивне придушення: цей крок виконується після обчислення величини та напряму градієнта (G та Θ). Безперервне придушення переважно використовується для тонких контурів. Для кожного пікселя в градієнтному

зображенні ми порівнюємо його градієнтну величину з малою площею пікселів, які розташовані в тому напрямку, що і градієнт. Якщо градієнтна величина пікселя справді найвища серед усіх таких сусідів, значення зберігається; в іншому випадку він пригнічується. Цей крок необхідний, тому що краї, які були виведені лише із значень градієнта, досить розмиті (товсті) навколо крайових областей. Таким чином, не максимальне придушення покращує якість контурів, роблячи їх якомога тоншими та максимально наближеними до реального життя;

– подвійний поріг: після виконання максимального придушення ми використовуємо метод подвійного порога для зменшення ефекту хибних спрацьовувань. Хибні спрацьовування – це області, які були виявлені як контури, але насправді не є ними. Це відбувається головним чином через присутність шуму зображення. Ми визначаємо два порогові значення: низький і високий поріг. Після визначення значень ми будемо піддавати кожен піксель наступним критеріям класифікації: 1) якщо величина градієнта вище за високий поріг, вона класифікується як контурний піксель і називається сильним контурним пікселем; 2) якщо величина градієнта нижче за нижній поріг, вона відкидається; 3) якщо величина лежить між низьким і високим порогом, піксель класифікується як слабкий контурний піксель. Слабкі контурні пікселі підпорядковуються процесу, що називається гістерезисом, де вони зберігаються, тільки якщо один з їх восьми сусідів є пікселем з сильним краєм; інакше вони відкидаються [7].

Наступний фрагмент коду застосовує алгоритм Canny для одноканального зображення електрода (рисунок 3.15) у відтінках сірого.

```
// завантаження бібліотек
#include <iostream>
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
using namespace std;
```

```
using namespace cv;
int main( )
{
      // зчитування зображення електрода
      Mat img0 = imread( "electrode.jpg" );
      // Використання відтінків сірого кольору
      cvtColor (img0, img0, CV_BGR2GRAY );
      Mat edges;
      // Налаштування визначення контуру
      Canny( img0, edges, 100, 300, 3, false );
      // показ зображення контуру на екрані
      imshow( "Canny detector" , edges);
      waitKey(0);
      return 0;
      }
```

Результат роботи визначення контурів за допомогою детектора Canny представлений рисунку 3.16.

Рисунок 3.15 – Вихідне зображення для визначення

Як видно з програми, спочатку завантажуються бібліотеки для їх використання. Далі завантажується зображення електрода, який знаходиться у кореневій папці OpenCV. З використанням команди cvtColor досягаємо використання відтінків сірого кольору. Canny дозволяє змінювати чутливість визначення контуру для знаходження контуру електрода. Змінюючи

коефіцієнти, можна отримати досить прийнятний результат визначення контуру об'єкта. В кінці на екран виводиться результат роботи за допомогою команди imshow.

Рисунок 3.16 – Визначення контуру електрода з детектором Canny

Щоб переконатися, що з допомогою даного методу визначення контуру можна досягти максимального визначення, спробуємо знайти контури інших об'єктів. Наприклад, знайдені контури олівця представлені на рисунку 3.17.

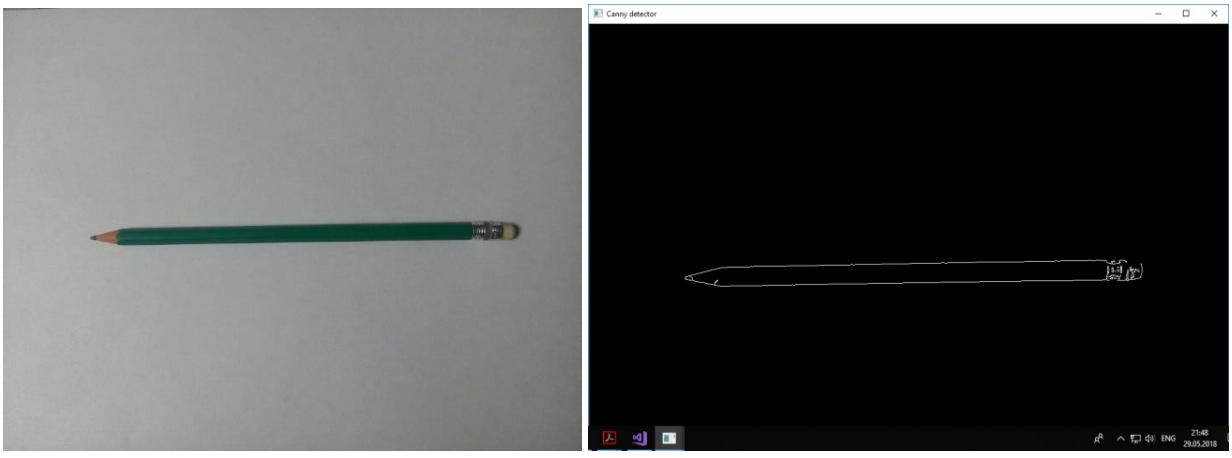

Рисунок 3.17 – Визначення контуру олівця з детектором Canny

Як видно за результатами, при визначенні контуру олівця та електрода помилок не помічено. З цього випливає, що при роботі з однотонним заднім фоном і довільним невеликим об'єктом цей метод практично не має помилок. У нашому випадку для визначення контуру електрода було створено всі умови для зменшення помилок знаходження контуру.

3.4 Визначення контурів за допомогою оператора Лапласа

Існує ще один метод, доступний для визначення контурів – оператор Лапласа. Оператор Лапласа – це друга похідна зображення. Математично це представляється так:

$$dst = \frac{d^2 src}{dx^2} + \frac{d^2 src}{dy^2}$$

Коли шукаються контури з використанням першої похідної, можна помітити, що області, які потенційно є контурними областями, мають досить велику величину похідної (градієнт). Як виявилося, у тих самих контурних областях друга похідна дорівнює нулю. Це явище використовується як критерій для виявлення контурів за допомогою оператора Лапласа [18].

Функція Laplacian( ) OpenCV реалізує оператор Лапласа. Фактично, один виклик Laplacian( ) оброблятиме як розміри x, так і y. Всередині він викликає функцію Sobel() для обчислення градієнтів. Фрагмент коду, що показує реалізацію Laplacian( ), виглядає так.

```
// завантаження бібліотек
#include <iostream>
#include <cstdlib>
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
```

```
using namespace std;
using namespace cv;
int main( )
{
Mat input_image = imread( "cat.jpg" ,IMREAD_GRAYSCALE );
// зчитування зображення кота
Mat output, scaled_output;
// Використання оператора Лапласа
Laplacian ( input_image, output, CV_16S , 3);
convertScaleAbs(output, scaled_output);
// показ зображення
imshow( "Laplacian" , scaled_output);
waitKey(0);
return 0;
}
```

Результат роботи знаходження контуру ручки, олівця та електрода представлений на рисунках 3.18, 3.19 та 3.20 відповідно.

Рисунок 3.18 – Визначення контуру ручки оператором Лапласа

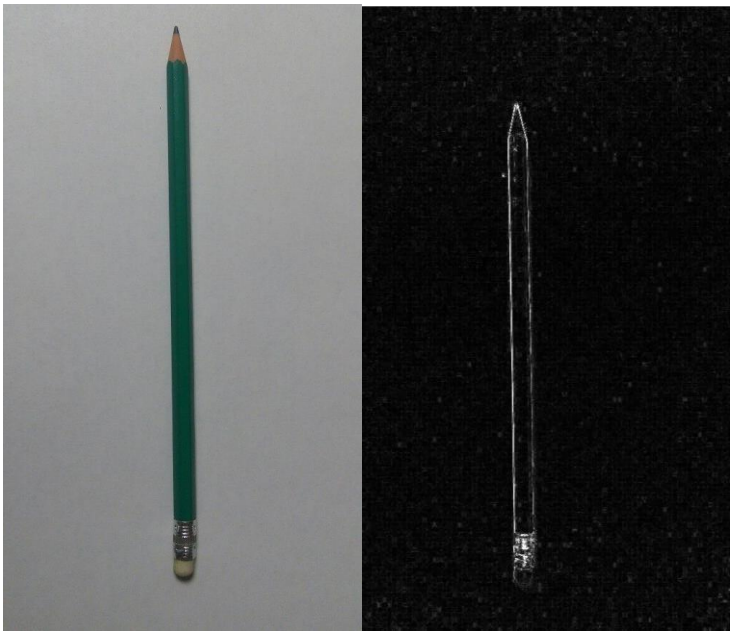

Рисунок 3.19 – Визначення контуру ручки оператором Лапласа

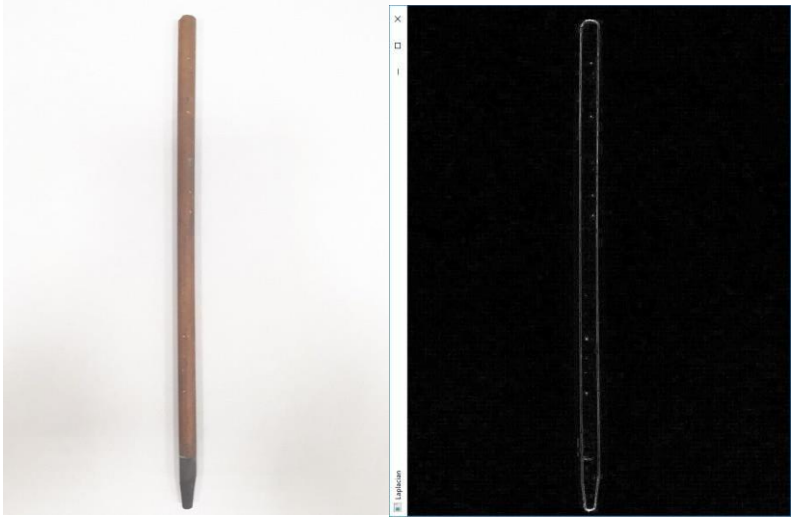

Рисунок 3.20 – Визначення контуру електрода оператором Лапласа

Основним мінусом цього методу є знаходження контуру шляхом зміни зображення і зображення виходять з додатковими відблисками або білими плямами. У ході роботи з координатами цей метод не дуже ефективний.

3.5 Визначення контурів за допомогою детектора меж Собеля

Найперший крок для виявлення контуру, з якого все починається, – це обчислення похідних. Похідні вздовж напрямків *x* та *y* обчислюються окремо і зберігаються у двох різних матрицях. Таким чином, для кожного пікселя (*x*, *y*) у вхідному зображенні ми маємо по суті градієнт у напрямках *x* та *y*: G_x та G_y. З цих двох значень градієнтів ми обчислюємо так звану градієнтну величину в точках (*x*, *y*). Розмір градієнта визначається такою формулою:

$$G = \sqrt{G_x^2 + G_y^2}\,.$$

Це те саме, що і величина двовимірного вектора, який має компоненти G_x і G_y. Фактично, похідні (або градієнти) у кількох вимірах часто візуалізуються концептуально як двовимірні вектори. Крім того, на додаток до величини градієнт також має напрямок, заданий формулою [19]:

$$\theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

Однак ми не будемо використовувати напрям градієнта у наших обчисленнях на даний момент. Повертаючись до нашої структури виявлення контуру, для кожного розташування пікселя тепер ми маємо величину градієнта в цій точці: *G*. Оскільки *G* враховує градієнти обох напрямків, це хороша кількісна оцінка кількості варіацій значення інтенсивності пікселів навколо кожної точки (*x*, *y*).

У визначенні контурів встановлено, що контур – це ті області, де зміна інтенсивностей пікселів велика, і ми маємо кількісну міру величини цієї зміни для кожного пікселя в зображенні. Все, що ще потрібно зробити, це з'ясувати, які пікселі мають значення *G*, які ми вважаємо досить високими, щоб їх класифікували як контури. І спосіб, яким ми вирішуємо це, – це зробити

функцію threshold, як останній крок нашої рамки виявлення меж. На рисунку 3.21 представлено блок-схему алгоритму роботи детектора Собеля [20].

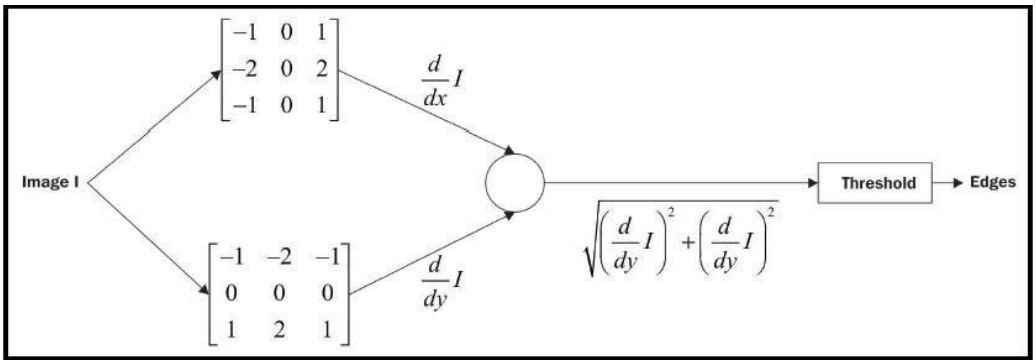

Рисунок 3.21 – Блок-схема алгоритму роботи детектора контурів Собеля

Ця блок-схема насправді зображує те, що ми називаємо детектором контурів Собеля. Це пов'язано з тим, що ядра, які були використані для обчислення похідних, є ядрами Собеля. Фрагмент коду, що реалізує алгоритм роботи детектора контурів Собеля, представлений нижче.

```
#include <iostream>
#include <cstdlib>
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
using namespace std;
using namespace cv;
// функція градієнта getGradientMagnitude
Mat getGradientMagnitude( Mat xGrad , Mat yGrad ) {
        CV_ Assert ( ( xGrad .rows == yGrad .rows) && ( xGrad .cols == yGrad .cols));
        Mat gradient_ magnitude( xGrad .rows, xGrad .cols, CV_16S );
        for ( int i = 0; i < xGrad .rows; ++i)
               for ( int j = 0; j < xGrad .cols; ++j)
              gradient_magnitude.at < short > (i, j) =
              abs( xGrad .at< short >( i, j)) + abs( yGrad .at< short >(i,
                  j));}
```

```
        return gradient_ magnitude; }
       // функція threshold
Mat thresholdGradientMagnitude( Mat gradient_magnitude , int threshold ) {
        Mat edges = Mat :: zeros( gradient_magnitude .rows,
              gradient_magnitude .cols, CV_8U );
        for ( int i = 0; i < gradient _magnitude .rows; ++i) {
              for ( int j = 0; j < gradient_magnitude .cols; ++j) {
                    if ( gradient_magnitude .at < short > (i, j) > = threshold )
                          edges.at < uchar > (i, j) = 255; } }
       return edges; }
int main( )
       //основна частина програми з командою Sobel
{
        Mat input_image = imread( "cat.jpg" , IMREAD_GRAYSCALE );
        Mat x_gradient, y_gradient;
        Sobel( input_image, x_gradient, CV_16S , 1, 0);
        Sobel( input_image, y_gradient, CV_16S , 0, 1);
        Mat gradient_magnitude = getGradientMagnitude( x_gradient, y_gradient);
        Mat edges_output = thresholdGradientMagnitude( gradient_magnitude, 200);
        imshow( "Edges" , edges_output);
        waitKey(0 );
        return 0; }
```

Функція getGradientMagnitude( ) приймає два об'єкти Mat, які є градієнтами *x* і *y* вхідного зображення як параметри, і повертає інший Mat, який містить величину градієнта для кожної точки пікселя. Розміри всіх трьох об'єктів Mat (два аргументи і той, що повертається) однакові та дорівнюють розмірам вихідного вхідного зображення.

Функція CV_ Assert( ) приймає вираз як аргумент і видає виняток, якщо він оцінюється як хибний під час виконання. Використовуючи CV_Assert(), ми спробували забезпечити той факт, що два вхідні аргументи мають однаковий розмір (який у наступному рядку також стає розміром Mat, який має бути повернутий). CV_ Assert( ) дуже корисний, якщо ви хочете забезпечити дотримання таких умов у своєму коді. Фактично, ця функція широко використовується у більшості внутрішніх реалізацій бібліотеки OpenCV. У

частині функції getGradientMagnitude( ) також описано обчислення G, яка спрощена від коренів до суми модуля кожного G_x і G_y . Робиться це для простоти, тому що праця з корнями дає величезні значення, а із сумою G_x і G_y видає добрі результати. Вибрана формула виглядає так [20]:

$$G = | G_x | + | G_y | .$$

Повна реалізація оператора Собеля описана вище. Є ще один аспект, який можна змінити у коді. Цей код можна запустити з різними значеннями порогових значень. Результати роботи детектора Собеля представлено на рисунках 3.22 та 3.23.

Під час роботи з різними методами визначення контуру було обрано детектор Canny. Основною перевагою є знаходження контуру за будь-якого типу освітлення. Також код дуже зручний для роботи з іншими функціями.

Основними недоліками інших методів є чутливість до освітлення, а для детектора Собеля написання зайвого коду для обчислення градієнтів.

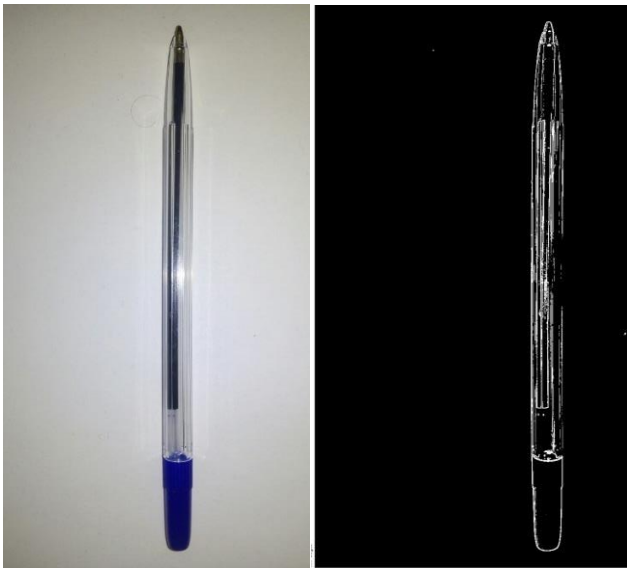

Рисунок 3.22 – Визначення контуру ручки з детектором Собеля

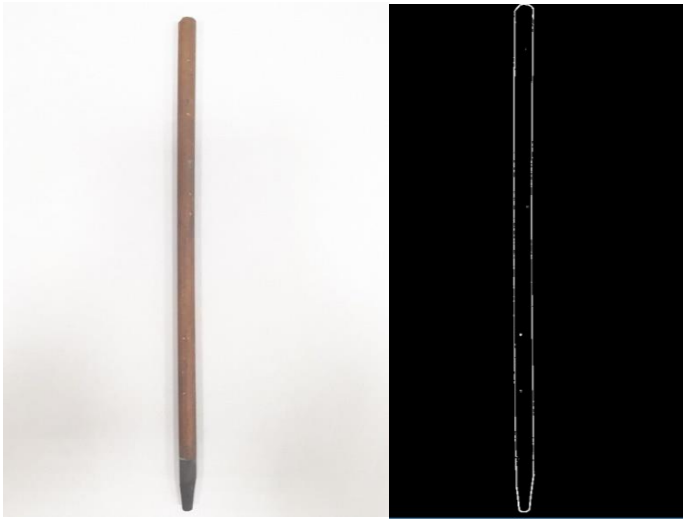

Рисунок 3.23 – Визначення контуру електрода з детектором Собеля

4 РЕЗУЛЬТАТИ ПРАКТИЧНОЇ ЧАСТИНИ

4.1 Реалізований пошук контуру електрода

Для визначення контурів було вибрано детектор контурів Canny. Даний метод дуже добре підходить для роботи в приміщеннях з освітленням, що змінюється. Як було сказано раніше, для поліпшення визначення контуру електрода були додані на заднє тло листи А4. Під час налаштування камер у разі отримання зображення іншого фону, крім білого, небіле тло вирізувалося програмним шляхом.

Програмний код визначення контуру представлений нижче.

```
#include <opencv2/opencv.hpp>
using namespace cv;
using namespace std;
int main( ) {
       // читаємо електрод (можна змінити шлях у зображенні)
       Mat img0 = imread( "electrode.jpg", 1);
       Mat img1;
       cvtColor( img0, img1, CV_RGB2GRAY);
       // Налаштування фільтра
       Canny( img1, img1, 100, 200);
       // Знаходження контурів
       vector contours;
       findContours( img1, contours, CV_RETR_EXTERNAL,
CV_CHAIN_APPROX_NONE);
       // використовуємо mask виділення електрода
       Mat mask = Mat: zeros (img1.rows, img1.cols, CV_8UC1);
      // CV_FILLED fills connected components found drawContours( mask,
contours, -1, Scalar(255), CV_FILLED);
      // Створення нового зображення
      Mat crop( img0.rows, img0.cols, CV_8UC3);
      // Завдання кольору заднього фону
      crop.setTo (Scalar(0, 0, 0));
      img0.copyTo(crop, mask);
      normalize(mask.clone(), mask, 0.0, 255.0, CV_MINMAX, CV_8UC1);
       // показ зображень на екрані
       imshow( "original", img0);
```

```
        imshow("mask", mask);
        imshow("canny", img1);
        waitKey();
        return 0;
}
```

Результати роботи програмної частини визначення контурів електрода представлені на рисунках 4.1 і 4.2. У ході роботи також використовувалася функція crop.setTo, що дозволяє встановити колір заднього фону.

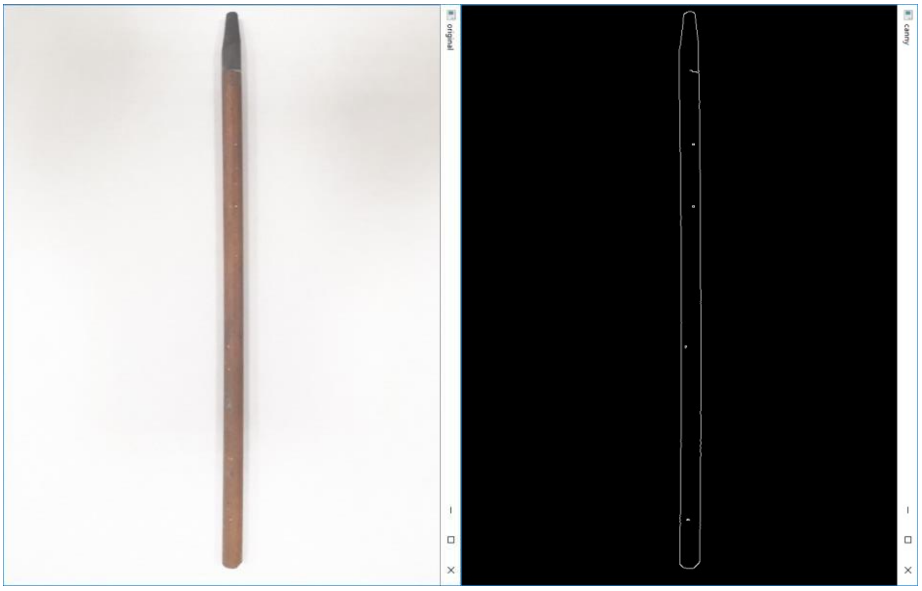

Рисунок 4.1 – Визначення контуру електрода

Рисунок 4.2 – Виділення білим кольором контуру за допомогою mask

4.2 Програмна частина роботи двох камер

Перед роботою були налаштовані та відкалібровані обидві камери.

Під час написання програми отримані знімки зберігалися у кореневій папці. Для кожної камери були задані імена: electrode_left та electrode_right.

Назви підібрані в залежності від розташування камер. Приклад фрагмента коду роботи обох камер представлений нижче.

```
#include <opencv2\opencv.hpp>
using namespace cv;
using namespace std;
int main( int , char **)
{
	// знімок із першої камери
	vector wr_params ;
	wr_params.push_back( cv:: IMWRITE_JPEG_QUALITY );
	wr_params.push_back(50 );
	VideoCapture cap(0);
	if (!cap.isOpened())
		return -1;
	Mat edges;
	Mat frame;
	cap >> frame;
	//збереження знімка 1
	imwrite( "electrode_left.jpg" , frame, wr_params);
	// знімок з другої камери
	VideoCapture cap1(2);
	if (!cap.isOpened())
		return -1;
	Mat edges 1;
	Mat frame1;
	cap1 >> frame1;
	//збереження знімка 2
	imwrite( "electrode_right.jpg" , frame1, wr_params);
	 // читаємо знімок 1 електрода
	Mat img0 = imread( "electrode_left.jpg" , 1);
		int x = 325,
		y = 25,
		width = 200,
		height = 455;
	img0 = img0 ( Rect ( x, y, width, height));
	Mat img1;
	cvtColor( img0, img1, CV_RGB2GRAY );
	// Налаштування фільтра
	Canny( img1, img1, 10, 100);
	// Знаходження контуру
	vector < vector < Point > > contours;
	findContours( img1, contours, CV_RETR_EXTERNAL , CV_CHAIN_APPROX_NONE );
	Mat mask = Mat :: zeros (img1.rows, img1.cols, CV_8UC1 );
	drawContours(mask, contours, -1, Scalar (255), CV_FILLED );
	crop.setTo(Scalar(255, 255, 255));
```

```
img0.copyTo(crop, mask);
normalize(mask.clone(), mask, 0.0, 255.0, CV_MINMAX , CV_8UC1 );
// читаємо знімок 2 електроди
Mat img2 = imread( "electrode_right.jpg" , 1);
Mat img3;
cvtColor( img2, img3, CV_RGB2GRAY );
Canny(img3, img3, 10, 100);
vector < vector < Point > > contours1;
findContours( img3, contours1, CV_RETR_EXTERNAL , CV_CHAIN_APPROX_NONE );
Mat mask1 = Mat :: zeros (img3.rows, img3.cols, CV_8UC1 );
drawContours(mask1, contours1, -1, Scalar (255), CV_FILLED );
crop1.setTo(Scalar(255, 255, 255));
img0.copyTo(crop1, mask);
normalize(mask1.clone(), mask1, 0.0, 255.0, CV_MINMAX , CV_8UC1 );
// Виведення зображень на екран
imshow( "electrode_left" , frame);
imshow( "electrode_right" , frame1);
imshow("electrodeR_canny", crop1);
imshow("electrodeL_canny", crop);
imwrite( "era1_canny.jpg" , img3);
imwrite( "era1_mask.jpg" , mask1);
waitKey();
return 0;
}
```

Результатом роботи є знімок електрода обома камерами. Знаходження їх контурів за допомогою детектора Canny та використання mask для покращення зображення. Отримані зображення представлені на рисунках 4.3 та 4.4.

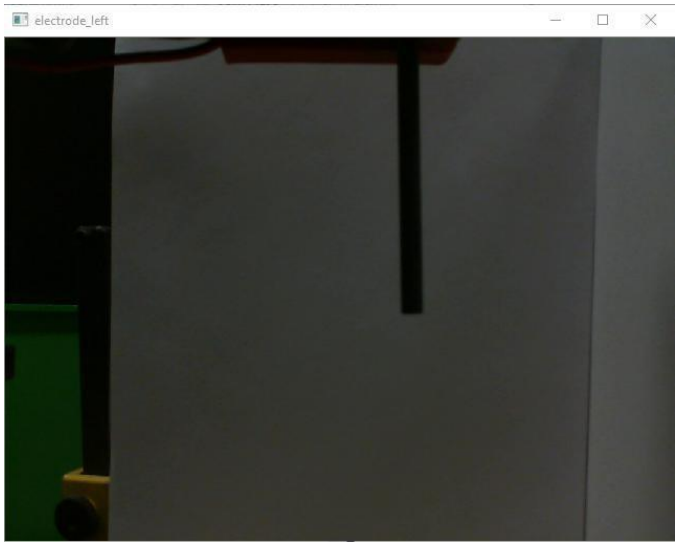

Рисунок 4.3 – Знімок електрода лівою камерою

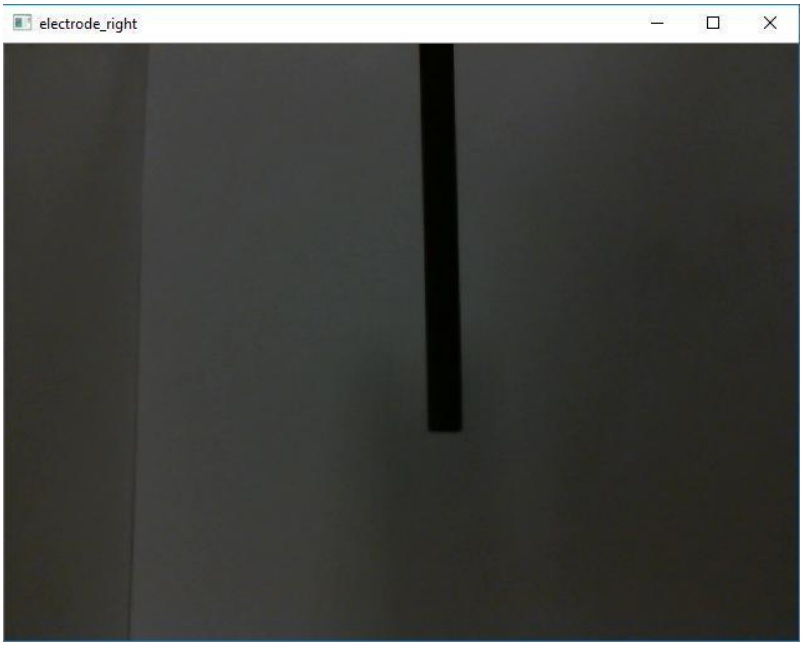

Рисунок 4.4 – Знімок електрода правою камерою

Нижче наведено знайдені контури (рисунок 4.5 та 4.6), отримані за допомогою детектора Canny. Для лівої камери зроблено обрізання заднього фону для розпізнавання лише електрода. Фрагмент коду для обрізки представлений вище.

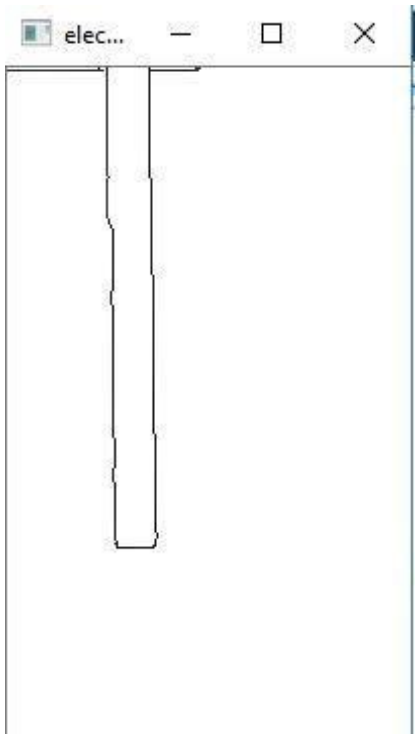

Рисунок 4.5 – Визначення контуру електрода лівою камерою

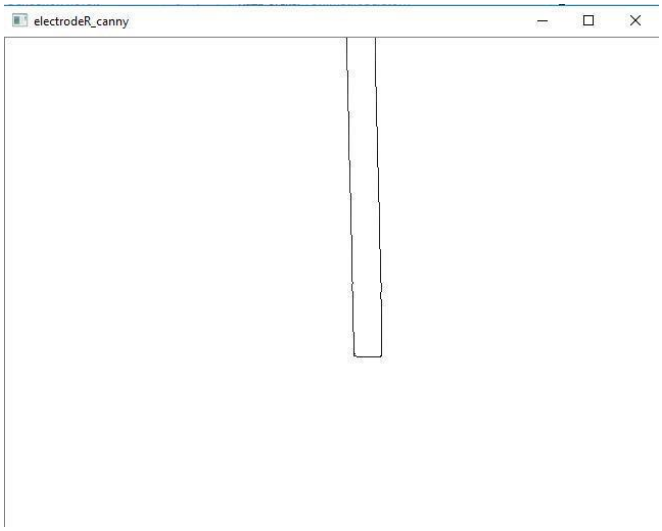

Рисунок 4.6 – Визначення контуру електрода правою камерою

Для визначення ефективності знаходження контуру було поставлено експеримент. Програма запускалася 20 разів та визначала контур зображення з урахуванням зміни освітленості. У результаті практичного експерименту було отримано 19 позитивних результатів. Ефективність роботи програми дорівнює 95%.

4.3 Пошук піксельних координат

Цифрове зображення, отримане камерою, може бути збережене у вигляді масиву в комп'ютері. Значення кожного елемента масиву (так званий піксель) – це яскравість точки зображення (або шкала сірого, якщо кольорове зображення, яскравість пікселя зображення буде червоного, зеленого та синього кольорів). Координати (u, v) кожного пікселя визначаються декартовою системою координат u - v, тобто кількістю стовпців та числом рядків пікселя в масиві. (u, v) – координати системи координат зображення у пікселях, як показано на рисунку 4.7.

Основне задача полягала у розробці методу пошуку піксельних координат. Кінцевий результат – це знаходження координат кожного контуру електрода. У ході розробки було вирішено написати програмну частину знаходження піксельних координат, а точніше координат першого чорного пікселя.

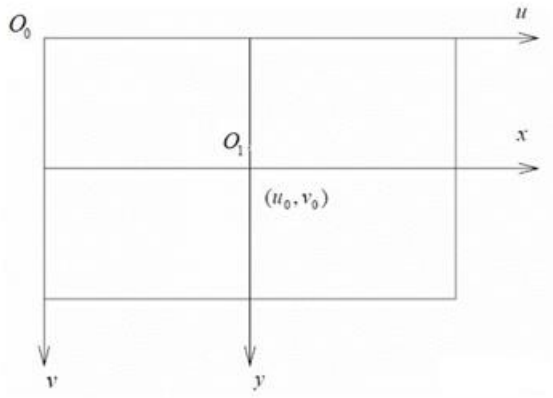

Рисунок 4.7 – Зображення системи координат та площини зображення системи координат

Фрагмент коду пошуку піксельних координат представлений нижче.

```
for ( int i = 0; i < crop.rows ; i++)
          for ( int j = 0; j <crop.cols; j++)
               if (crop.at< Vec3b >(i, j) [ 0 ] < 255 &&
                    crop.at< Vec3b >(i, j) [ 2 ] < 255 &&
                    crop.at < Vec3 b > ( i, j) [ 1 ] <255)
              {
                    crop.at < Vec3 b > ( i, j) [ 0 ] = 0;
                    crop.at < Vec3b > (i, j) [ 1 ] = 0;
                    crop.at < Vec3b > (i, j) [ 2 ] = 0;
                    k = i;
                    l = j;
                    i = crop.rows ;
                    break ;
              }
          cout << " x = " << k << endl << "y=" << l << endl;
```

Тут представлений цикл знаходження чорного пікселя. Для початку задаємося початковими координатами – це 0 та 0. У нашому коді це стовпці *i*

та рядки *j*. Далі у нас задана умова для наших *i* та *j*: збільшувати значення доки не знайдеться піксель не білого кольору. При знаходженні чорного пікселя значення координат *i* та *j* перезаписуються в *k* та *l*. У результаті ми отримуємо координати чорного пікселя.

Крім цього в коді додано функцію визначення піксельних координат шляхом натискання клавіші миші на будь-яку координату для обох зображень, отриманих з камер. Координати виводяться на екран. Фрагмент коду, що враховує цю умову, представлений нижче.

```
#include <opencv2\opencv.hpp>
#include <locale.h >
using namespace cv;
using namespace std;
int k, l, m, n = 0;
void onMouse( int event , int x , int y , int flags , void * param )
{
      cv:: Mat *im = reinterpret_cast <cv:: Mat *>( param );
      // Відправка event
      switch ( event ) {
      // event для руху мишки вниз
      case cv:: EVENT_LBUTTONDOWN :
      // Виведення пікселів у точках ( x, y )
            std::cout << "при точках (" << x << "," << y << ") значення
дорівнює: "
                  << static _cast < int >(im->at< uchar >(cv:: Point ( x
, y ))) << std::endl;
            break ;
      }
       }
```

Також в кінці коду потрібно викликати функцію setMouseCallback. Ця функція встановлює функцію зворотного виклику, яка викликається кожного разу, коли у вікні Windows відбуваються натискання миші. Нижче наведено докладне пояснення кожного параметра вищезазначеної функції OpenCV:

– winname – ім'я вікна OpenCV. Усі події миші, пов'язані з цим вікном, будуть зареєстровані;

– onMouse – функція зворотного виклику. Щоразу, коли відбуваються натискання миші по вікну ця функція зворотного виклику буде викликана;

– userdata – цей покажчик буде переданий функції зворотного виклику.

Фрагмент коду для функції зворотного виклику представлений нижче.

```
{
imshow( "electrodeR_canny" , crop1);
imshow( "electrode_right" , frame1);
imshow( "electrode_left" , frame);
setMouseCallback( "electrodeL_canny" , onMouse, reinterpret_cast < void
*>(&crop));
setMouseCallback( "electrodeR_canny" , onMouse, reinter-
pret_cast < void *>( &crop1));
waitKey();
return 0;
}
```

Результати знаходження координат представлені рисунку 4.8. Також нижче будуть представлені зображення роботи всієї програми:

– відкриття оригінальних зображень, знятих камерами (рисунок 4.9);

– знайдені контури за допомогою детектора Canny (рис. 4.9);

– одержані піксельні координати (рисунок 4.9);

– загальна робота усієї програми (рисунок 4.9).

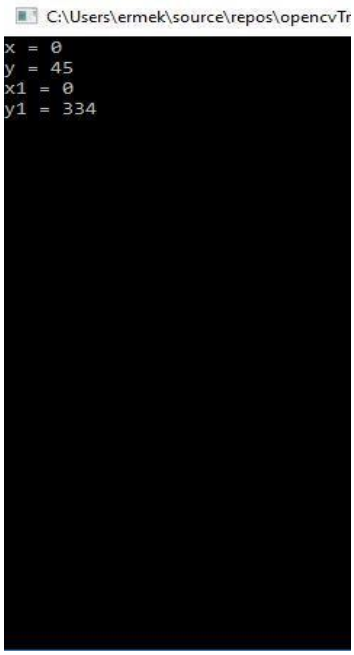

Рисунок 4.8 – Знаходження піксельних координат зображення електрода

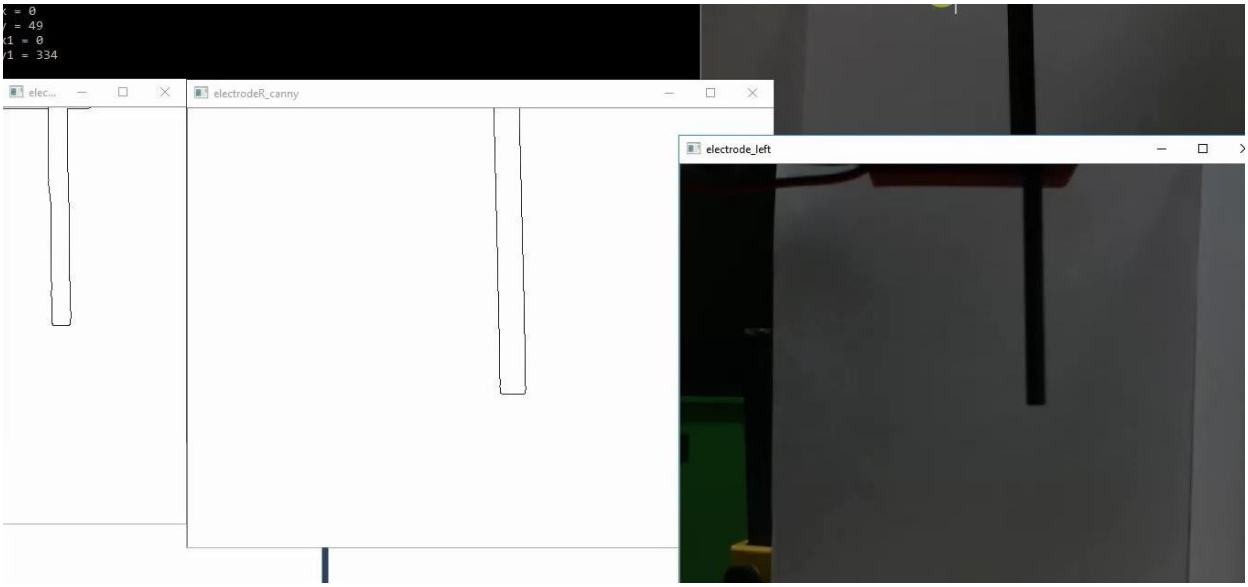

Рисунок 4.9 – Відкриття оригінальних зображень та визначення їх контурів

Зображення електрода зі знайденими піксельними координатами шляхом натискання ЛКМ та виведення їх на екран представлено на рисунку 4.10.

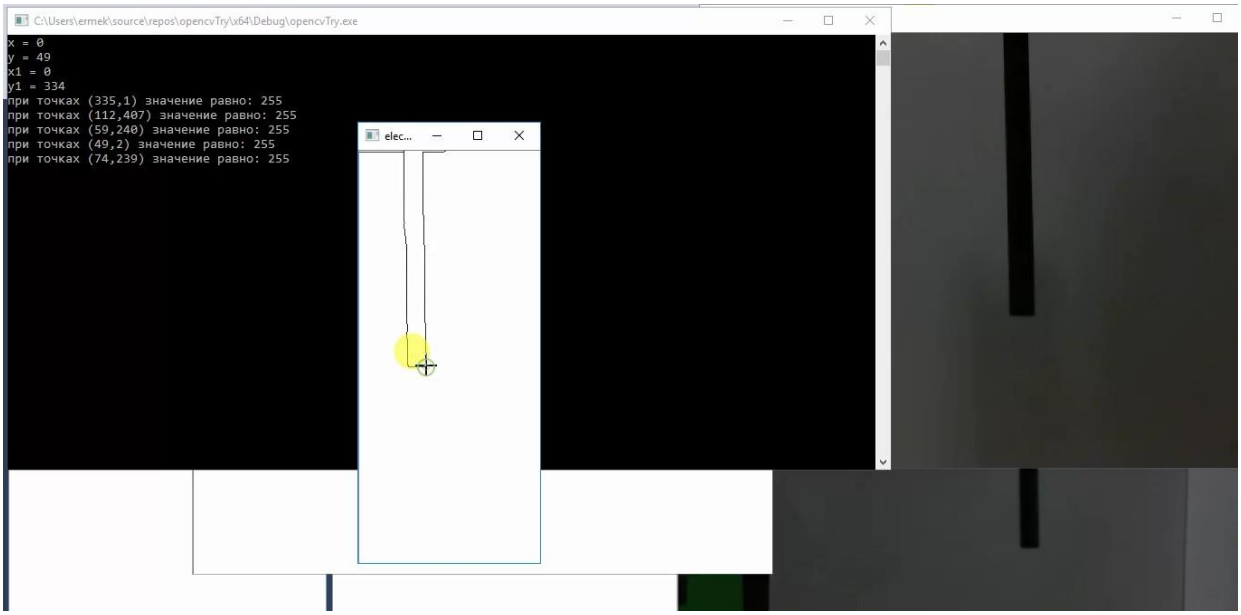

Рисунок 4.10 – Знаходження піксельних координат зображення електрода шляхом виклику функції setMouseCallback

У ході роботи було реалізовано систему визначення координат електрода. Система отримує зображення електрода з камер. Далі знаходиться контур електрода за допомогою детектора контурів Canny. В кінці, обробляючи отримане зображення контурів електрода, знаходяться координати чорних пікселів та виводяться на екран.

4.4 Охорона праці

Охорона праці являє собою комплекс заходів, які спрямовані на попередження випадків, що є загрозою для безпеки і здоров’я людей під час виконання ними роботи. Охорона праці є комплексом заходів, які включають питання медицини, організації, техніки та інших заходів, які використовуються для запобігання нещасних, травматичних випадків, небезпечних впливів, вибухів, захисту від шкідливих речовин, завдання шкоди від удару струму, уникнення помилок, що мають небезпечні наслідки для здоров’я людини та безпеки підприємства.

Суттєвим та необхідним етапом охорони праці є навчання працівників дотриманню правил безпеки, надання спорядження індивідуального захисту і спеціального обладнання.

Тому необхідно виявити можливі небезпеки, виникнення яких є вірогідним під час роботи. При розробленні даної системи такими небезпеками можуть бути:

– небезпека впливу на зір розробника: за умови невиконання правил безпеки роботи за комп’ютером, зір розробника може бути пошкодженим;

– ризик удару струму: якщо електричні складові системи поєднані некоректно, система зазнала впливу рідини чи при інших недопустимих діях, користувач може отримати удар струму, що може викликати навіть смертельні наслідки;

– небезпека травм: компоненти системи можуть бути розташовані в небезпечних місцях;

– втрата інформації при її передаванні, помилки програмного забезпечення та інші подібні проблеми можуть призвести до несподіваних наслідків в роботі системи.

Наступним етапом після того, як було визначено потенційні загрози є вживання відповідних заходів безпеки для запобігання цих загроз і зменшення кількості ризиків. Такі заходи можуть складатись з:

– використання захисних окуляр для захисту очей під час розробки ПЗ;

– приховування всіх елементів системи, де проходить високий струм за допомогою захисних кожухів, для запобігання небезпеки удару струму;

– запровадження періодичного інструктажу, а також навчання працівників правилам безпечної розробки та роботи з автоматизованою системою, її компонентами та комп’ютерами. Працівники повинні бути інформовані щодо потенційних небезпек і методів роботи з системою.

ВИСНОВКИ

У ході виконаної роботи було розроблено систему визначення координат електрода маніпулятора промислового робота. Для досягнення мети роботи було виконано такі задачі:

– проведено огляд існуючих рішень у галузі технічного зору зварювальних роботів;

– розроблено технічну частину у вигляді конструкції експериментального макету;

– проведено аналіз бібліотеки OpenCV, до якої входить як калібрування камери, так і підбір різних методів визначення контурів об'єкта;

– написано програмний код роботи системи визначення координат (мовою C++);

– проведено експерименти для визначення ефективності роботи системи.

Розроблена система може використовуватися на виробництвах, де використовують зварювальні роботи. При зміні технічної частини, система також може використовуватися для різних цілей технічного та комп'ютерного зору.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ДСТУ 3008-15. Документація. Звіти у сфері науки та техніки. структура та правила оформлення. Введ. 2015-06-22. К. Держстандарт України, 2017. 29 с.

2. Методичні вказівки з підготовки та захисту кваліфікаційної роботи здобувачами другого (магістерського) рівня вищої освіти спеціальності 174 Автоматизація, комп'ютерно-інтегровані технології та робототехніка, освітньо-професійних програм: «Комп'ютерно-інтегровані технологічні процеси і виробництва», «Комп'ютеризовані та робототехнічні системи» / Упоряд. І. Ш. Невлюдов, Р. В. Артюх, В. В. Безкоровайний, Н. П. Демська, В.В. Євсєєв, О. І. Филипенко, О. М. Цимбал. Харків: ХНУРЕ, 2024. 57 с.

3. Воловік А.В. Калібрування камери модуля визначення положення виконавчого елемента робота. «Автоматизація та приладобудування» ADED-2025, Випуск 2. С. 194-200.

4. https:// www.kuka.com/ua (дата звернення 6.09.25).

5. https://ua.wikipedia.org/wiki/Microsoft_Visual_Studio (дата звернення 17.09.25).

6. OpenCV. https://ua.wikipedia.org/wiki/OpenCV (дата звернення 3.10.25).

7. OpenCV. https://docs.opencv.org/3.0-beta/doc/tutorials/tutorials.html (дата звернення 23.08.25).

8. FANUC iRVision. Fully Integrated Plug & Play Vision System Machine Vision in 2D and 3D. https://www.fanucamerica.com/docs/default-source/robotics-files/irvision/2019_fac_irvision_brochure_digital-101419.pdf (дата звернення 21.09.25).

9. https://new.abb.com/ (дата звернення 6.09.25).

10. https:// www.kuka.com/en-us/about- kuka (дата звернення 26.08.25).

11. LifeCam Cinema. Режим доступу: https://www.microsoft.com/ accessories/ua/products/webcams/lifecam-cinema/h5d-00015 (дата звернення 6.09.25).

12. Datta Samyak. Learning OpenCV 3 Application Development. Packt Publishing, 2017. 305 p.

13. Hartley R., Zisserman A. Multiple View Geometry in Computer Vision. Cambridge University Press, 2004. 673 p.

14. Roger Y. Tsai. A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses. IEEE Jour-nal of Robotics and Automation, 3(4), 1987. P. 323-344.

15. Zhang B., Li Y. F. Automatic Calibration and Reconstruction for Active Vision Systems, Intelligent Systems, Control, and Automation. Science and Engi-neering. V. 57, Springer, 2012. P. 175.

16. Svoboda T., Martinec D., Pajdla T. A convenient multicamera self-calibra-tion for virtual environments. PRESENCE: Teleoperators and Virtual Environments 14, 4, 2005. P. 407-422.

17. Camera Calibration. Режим доступу: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_calib3d/py_calibration/ py_cali-bration.html#calibration (дата звернення 16.11.25).

18. Samarth Brahmbhatt. Practical OpenCV (Technology in Action). Apress, 2013. 345 p.

19. Daniel Lélis Baggio, Shervin Emami, David Millán. Mastering OpenCV 3 - Second Edition. Packt Publishing, 2017. 250 p.

20. Ayu Ambarwati, Rossi Passarella. Segmentasi Citra Digital Menggunakan Thresholding Otsu untuk Analisa Perbandingan Deteksi Tepi.

ANNUAL RESEARCH SEMINAR - 2016. 6 Desember 2016, Vol 2, No. 1. P. 216-226.