НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет робототехніки та приладобудування

Кафедра комп'ютерно-інтегрованих технологій виробництва приладів

До захисту допущено:
Завідувач кафедри
_____ Наталія СТЕЛЬМАХ
«12» 06 2026 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп'ютерно-інтегровані системи та технології в приладобудуванні»

спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології»

на тему: «Автоматизована система визначення класу чистоти поверхні скла»

Виконав:
студент IV курсу, групи ПБ-з21
Александров Олександр Олександрович

Керівник:
Професор, доктор технічних наук
Тимчик Григорій Семенович

Рецензент:
Старший викладач
Кравченко І. В.

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2026 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет робототехніки та приладобудування
Кафедра комп'ютерно-інтегрованих технологій виробництва приладів

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 151 «Автоматизація та комп'ютерно-інтегровані технології»

Освітньо-професійна програма «Комп'ютерно-інтегровані системи та технології в приладобудуванні»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Наталія СТЕЛЬМАХ
«12» 06 2026 р.

ЗАВДАННЯ
на дипломну роботу студенту

Александрову Олександру Олександровичу

1. Тема роботи «Автоматизована система визначення класу чистоти поверхні скла», керівник роботи Тимчик Григорій Семенович, доктор технічних наук, професор, затверджені наказом по університету від «25» травня 2026р. №1888-с

2. Термін подання студентом завершеної роботи 07 червня 2026 р.
3. Вихідні дані до роботи: сумісність з операційною системою Windows, оптичні лінзи і призми, класифікація фотографій за чистотою поверхні оптичної деталі відповідно до норм ДСТУ 11141-84, бінокулярний мікроскоп МБС-10, цифрова камера-окуляр SIGETA MDC-500 5.0 MP, драйвер.
4. Зміст роботи:
Перелік умовних позначень.
Вступ.
Розділ 1. Обладнання і способи візуальної перевірки чистоти поверхні оптичних деталей. 1.1. Нормування і визначення класів чистоти. 1.2. Обґрунтування потреби і способів автоматизації процесу. 1.3. Класифікація і технічні параметри обладнання. Порівняльний аналіз існуючих систем визначення класу чистоти поверхні скла. Висновки до розділу 1.
Розділ 2. Проєктування апаратної частини автоматизованої системи визначення класу чистоти поверхні скла. 2.1. Розробка структурної схеми автоматизованої системи. 2.2. Розробка структурної схеми загального алгоритму. 2.3. Підбір компонентів для апаратного модулю отримання зображень поверхні деталі. Висновки до розділу 2.

Розділ 3. Розробка програмного забезпечення. 3.1. Вибір та обґрунтування мов програмування і бібліотек. 3.2. Архітектура і узагальнена блок-схема програмного забезпечення. 3.3. Розробка головного вікна програми та опис його функціонал 3.4. Розробка вікна параметрів обробки. 3.5. Розробка вікна «База виявлени дефектів деталі». 3.6. Реалізація функції обробки зображення, виявлення дефект та класифікації поверхні за ДСТУ11141-84. 3.7. Демонстрація роботи програм Висновки до Розділу 3.

Розділ 4. Проєктування транспортного модулю транспортування і соортуван деталей. 4.1. Розробка структури модуля транспортування і сортування деталей4. Підбір апаратних засобів для реалізації модуля транспортування і сортуван деталей 4.

5. Перелік ілюстративного матеріалу: Порівняльна характеристика ручних автоматизованих методів контролю(А1), Структурна схема системи(А1 Структурна схема загального алгоритму(А1), Узагальнена блок-схем програмного забезпечення(А1), Скріншоти вікон розробленого програмно забезпечення(А1). Блок-схема алгоритму функції перерахунку мм в рх(А1), Бло схема алгоритму обробки зображення(А1), Блок-схема алгоритму логічно визначення класу чистоти відповідно до ДСТУ11141-84(А1), Приклади зображен поверхонь лінз до і після обробки (А1).

6. Дата видачі завдання 14 квітня 2026 р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Огляд та аналіз літературних джерел. Постановка задачі.	25.04.2026	
2	Розробка структурно-функціональної схеми	30.04.2026	
3	Вибір основних електронних елементів	06.05.2026	
4	Розробка архітектури програмного забезпечення	15.05.2026	
5	Технічні розрахунки	20.05.2026	
6	Тестування системи	30.05.2026	
7	Оформлення пояснювальної записки та графічного матеріалу	05.06.2026	

Студент Олександр АЛЕКСАНДРОВ

Керівник Григорій ТИМЧИК

АНОТАЦІЯ

Дипломна робота на тему «Автоматизована система визначення класу чистоти поверхні скла» розроблена студентом Александровим Олександром Олександровичем під керівництвом Тимчика Григорія Семеновича, доктора технічних наук, професора кафедри комп'ютерно-інтегрованих технологій виробництва приладів Національного технічного університету «Київський політехнічний інститут імені Ігоря Сікорського»

У дипломній роботі розроблено автоматизовану систему визначення класу чистоти поверхні скла. Ця операція присутня при виробництві будь-яких прецизійних виробів зі скляними елементами. Ціллю автоматизації цього кроку виробництва є підвищення повторюваності, об'єктивності і продуктивності виробництва скляних виробів.

Дипломна робота складається зі вступу, чотирьох розділів, висновків, списку джерел та додатків. Загальний обсяг становить 151 сторінка, містить 43 рисунка, 10 таблиць та 53 використаних джерела.

У вступі обґрунтовано актуальність теми, сформовано мету і завдання роботи, предмет дослідження.

У першому розділі розкрито питання контролю чистоти поверхні скла. Описана стандартна документація, описано проблему ручного виконання операції, проведено дослідження існуючих методів контролю і проведено порівняльний економічний аналіз існуючих систем заснованих на оптичних і інтерференційних технологіях.

У другому розділі розділено і описано загальну архітектуру системи, на 4 складові. Побудовано структурну схему системи і схему загального алгоритму роботи системи. Підібрано матеріальне забезпечення для модулю отримання зображення.

3й розділ описує розробку програмного забезпечення, вибір технологій для його реалізації. Побудову візуального інтерфейсу програми і її внутрішніх функцій. Розроблено, описано і реалізовано логічні алгоритми оцінки класу чистоти деталі у відповідності до вимог що описані ДСТУ 11141-84. Проведено демонстрацію роботи програмного забезпечення

У четвертому розділі описаний процес розробки структури модуля автоматичного транспортування і сортування деталей та виконаний і обґрунтований підбір конкретних апаратних комплектуючих для реалізації модуля автоматичного транспортування.

Ключові слова: автоматизована система визначення класу чистоти поверхні скла, клас чистоти поверхні, лінзи, комп’ютерний зір, модульна архітектура

ANNOTATION

The thesis on the topic "Automated system for determining the cleanliness class of a glass surface" was developed by student Aleksandrov Oleksandr Oleksandrovich under the supervision of Hryhorii Tymchyk, Doctor of Technical Sciences, Professor of the Department of Computer-Integrated Technologies for the Production of Devices at the National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”.

In the bachelor's thesis, an automated system for determining the cleanliness class of a glass surface has been developed. This operation is present in the manufacture of any precision products with glass elements. The purpose of automating this production step is to increase the repeatability, objectivity, and productivity of glass products manufacturing.

The bachelor's thesis consists of an introduction, four chapters, conclusions, a list of references, and appendices. The total volume is 151 pages, contains 43 figures, 10 tables, and 53 sources used.

In the introduction, the relevance of the topic is substantiated, the goal and objectives of the work, and the subject of research are formed.

In the first section, the issues of glass surface purity control are revealed. The standard documentation is described, the problem of manual operation execution is described, a study of existing control methods is conducted, and a comparative economic analysis of existing systems based on optical and interference technologies is performed.

In the second section, the general architecture of the system is divided and described into 4 components. A block diagram of the system and a diagram of the general operation algorithm of the system are constructed. Hardware equipment for the image acquisition module is selected.

The 3rd chapter describes the software development, the choice of technologies for its implementation. The construction of the visual interface of the program and its internal functions. Logical algorithms for assessing the purity class of a part in accordance with the requirements described in DSTU 11141-84 are developed, described, and implemented. A demonstration of the software operation is conducted.

In the fourth chapter, the process of developing the structure of the automatic transportation and sorting module of parts is described, and the selection of specific hardware components for the implementation of the automatic transportation module is carried out and justified

Keywords: *automated system for determining the cleanliness class of a glass surface, surface cleanliness class, lenses, computer vision, modular architecture*

Зміст

Вступ

В сучасному світі скло оточує нас усюди. Воно є в наших телефонах, окулярах, в кожному автомобілі, автобусі, тролейбусі, поїзді, літаку, акваріумі, ліхтарях на вулиці і лампочках в домівках, в лінзах камер телефонів, в будинках, в мікроскопах, телескопах, в медичному обладнанні і різних лазерних системах.

Однією з найважливіших характеристик скла є його гладка і чиста поверхня, бо нерівності і бруд на поверхні будуть спотворювати промені світла що проходить через нього, цим погіршуючи якість виконання однієї з основних функцій скла – пропускати світло крізь себе без або з конкретними заданими спотвореннями.

Отже однією з ключових характеристик скла, що потрібно контролювати при його виробництві є чистота його поверхні. Тільки скляних лінз у світі за різними джерелами виробляється від 500млн до 1млрд за рік, і кожну з цих лінз потрібно проконтролювати на чистоту поверхні. В рамках сучасності, коли все більше і більше аспектів різних виробництв передають на виконання системам автоматизації, просто необхідно ефективно автоматизувати операцію що виконується до 1 млрд разів на рік. [2]

У цій дипломній роботі буде розглянуто існуючі промислові системи і наукові прототипи систем автоматичної класифікації чистоти поверхні скла, буде проведено порівняльний аналіз роботи даних систем, з виокремленням найкращих економічних і ефективністних аспектів кожної із систем. Також буде здійснено проєктування автоматизованої системи класифікації чистоти поверхні скла.

РОЗДІЛ 1 ОБЛАДНАННЯ І СПОСОБИ ВІЗУАЛЬНОЇ ПЕРЕВІРКИ ЧИСТОТИ ПОВЕРХНІ ОПТИЧНИХ ДЕТАЛЕЙ

1.1. Нормування і визначення класів чистоти

В сучасному приладобудуванні, зокерма, при виробництві систем що включають в себе оптичні деталі з різних матеріалів, якість обробки поверхні оптичних деталей є одним з найважливіших чинників що визначають точність і відповідність вимогам роботи оптичної системи. Наявність таких дефектів як царапини, точки, сколи на поверхнях лінз, призм, дзеркал і тд., призводить до зміни шлях світлового потоку що проходить через деталь, що призводить до виникнення дифракційних завад, нечистого зображення, або, якщо мова про лазерні системи, передчасного руйнування оптики.[1][2]

Нормативним документом що описує класифікацію і методи контролю чистоти поверхні є ДСТУ 11141-84 «Деталі оптичні. Класи чистоти поверхні. Методи контролю». Цей стандарт поширюється на оптичні деталі зі скла, кристалів, ситаллів, полімерних матеріалів, оптичної кераміки, металів і волоконнооптичних виробів. Він встановлює класи чистоти і методи контролю поверхні деталей з параметром шороховатості Rz не більше 0,1 мкм після виконання механічної, іонної і інших видів обробки і нанесення покриттів.[7][11]

Стандарт визначає клас чистоти поверхні по кількості, розмірам і взаємному розташуванню таких дефектів як царапини, точки і сколи на поверхні контрольованої деталі. Нормування відбувається в межах робочої зони деталі, тобто світлового діаметру. В залежності від розташування деталі в приладі і призначення деталі, стандарт розділяє всі класи чистоти на 2 основні групи: [7][11]

1. Класи 0-10, 0-20, 0-40 -для поверхонь деталей (окрім волоконно-оптичних виробів), розташованих в площині дійсного зображення або в площині предмету оптичних системи приладу, наприклад шкали, візири, сітки. Також, особливістю цих класів є те що при контролі поверхня лінзи ділиться на 3 кругові зони світлового діаметру – центральну, середню і крайню, межами середньої і крайньої зон є

коцентричні кола з діаметром 1/3 і 2/3 від світлового діметру деталі. Однак, деталь не ділиться на зони якщо її світловий діаметр меньше 5 мм. Розмір дефектів в центраьлній зоні не має бути більшим ща 0,001мм, а розміри і кількість дефектів в середній і крайній зонах обмежуються таблицею 1 стандарту (таблиця 1.1.1).[7][11]

2. Класи що позначаються римськими цифрами I, II, III, IV, V, VI, VII, VIII, VIIIa, IX, IXa – для поверхонь приладів що знаходяться поза площиною дійсного зображення або поза площиною предметів оптичної системи приладу, а також ці класи чистоти класифікують волоконноптичні вироби. Контроль дефектів для цих класів чистоти проводиться на всьому світловому діаметрі деталі. Обмеження розмірів і кількості дефектів для деталей цих класів чистоти описані в таблицях 2, 3 і 4 стандарту.[7][11]

Метод контролю, для людини яка виконує контроль, згідно зі стандартом виглядає так [7][11]:

1. Очистка поверхні деталей від всіх забруднень.

2. Встановлення джерела світла з лампою накалювання потужністю 60-100Вт. Поверхню деталей слід передивлятись в косо направленому пучці світла на чорному фоні.

3. В залежності від класу чистоти налаштовується збільшення мікроскопу на збільшення з яким будуть розглядатись усі дефекти.

4. Розміри царапин і точок оцінюють порівнюючи з наборами виміряних зразків. (бо інакше людині дуже не зручно вимірювати розмір дефекту, а так людина має піднести до мікроскопу деталь що контролюється, потім підносити по черзі зразки з виміряними дефектами щоб знайти підходящий. Це займає купу часу, і дає багато простору для погрішності.)

5. Якщо клас деталі 0-10, 0-20, 0-40 – деталь ділиться на 3 зони контролю – центральну, середню і крайню.

5.1. В центральній зоні не має бути точок діаметром і царапин шириною більше 0,001мм.

5.2. В середній і крайовій зоні допуски на розміри точок контролюються таблицею 1 ДСТУ (таблиця 1.1).

6. Якщо клас деталі I, II, III, IV, V, VI, VII, VIII, VIIIa, IX, IXa:

6.1. Провести контроль по царапинам і точкам по обмеженням з таблиці 2 ДСТУ (таблиця 1.2), при цбому не враховуючи дефекти менші розміромі а дефекти з таблиці 4 ДСТУ (таблиця 1.4).

6.2. Провести контроль по великим царапинам і точкам по обмеженням з таблиці 3 ДСТУ (таблиця 1.3).

6.3. Провести контроль по скупченням дефектів:

6.3.1. Визначити площу поверхні зайняту скупченням дефектів відповідно до таблиці 2 ДСТУ (таблиця 1.2).

6.3.2. Обчислити площу дефектів зайняту точками.

6.3.3. Обчислити площу дефектів зайняту царапинами.

6.3.4. Порівняти сумарну площу зайняту дефектами з допуском.

Таблиця 1.1 Таблиця 1 стандарту 11141-84 [7][11]

мм							
Клас чистоти	Зона	Царапини		Точки			
		Ширина, мм не більше	Сумарна довжина не більше	Діаметр не більше	Кількість не більша при світловому діаметрі		
					Від 5 до 20	Більше 20 до 60	Більше 60
0-10	Середня	0,002	0,2 * св.D	0,004	1	3	5
	Крайова	0,004	0,3 * св.D	0,006	3	6	10
0-20	Середня	0,004	0,2 * св.D	0,01	1	3	5

	Крайова	0,006	0,3 * св.D	0,015	3	6	10
0-40	Середня	0,006	0,2 * св.D	0,015	1	3	5
	Крайова	0,008	0,3 * св.D	0,025	3	6	10

Таблиця 1.2 Таблиця 2 стандарту 11141-[7][11]

Клас чистоти	Царапини		Точки		Скупчення дефектів	
	Ширина мм, не більше	Сумарна довжина мм, не більше	Діаметр мм, не більше	Кількість, не більше	Діаметр обмеженої діляни, мм	Суммарна площа царапин і точок мм², не більше
I	0,004	2,0 * св.D	0,02	0,5 * св.D	1,0	0,004
II	0,006		0,05		1,2	0,006
III	0,01		0,1		2,0	0,02
IV	0,02		0,3		5,0	0,1
V	0,04		0,5		10,0	0,4
VI	0,06		0,7		25,0	3,0
VII	0,1		1,0		50,0	10,0
VIII	0,2		2,0	0,4 * св.D	-	-
VIIIa	0,3	1,5 * св.D				
IX	0,3	2,0 * св.D	3,0	0,3 * св.D		
IXa	0,4	1,5 * св.D				

Таблиця 1.3 Таблиця 3 стандарту 11141-84[7][11]

мм				
Клас чисоти	Царапини		Точки	
	Ширина	Сумарна довжина, не більше	Діаметр	Кількість, не більше
III	Від 0,006 до 0,01 включ.	1,0 * св.D	Від 0,05 до 0,1 включ.	0,2 * св.D
IV	Від 0,01 до 0,02 включ.		Від 0,1 до 0,3 включ.	
V	Від 0,02 до 0,04 включ.		Від 0,3 до 0,5 включ.	
VI	Від 0,04 до 0,06 включ.		Від 0,5 до 0,7 включ.	
VII	Від 0,06 до 0,1 включ.		Від 0,7 до 1,0 включ.	
VIII	Від 0,1 до 0,2 включ.		Від 1,0 до 2,0 включ.	
VIIIa	Від 0,2 до 0,3 включ.	0,7 * св.D		
IX		1,0 * св.D	Від 2,0 до 3,0 включ.	0,1 * св.D
IXa	Від 0,3 до 0,4 включ.	0,7 * св.D		

Таблиця 1.4 Таблиця 4 стандарту 11141-84[2][9]

мм					
Клас чистоти	Царапина, не більше	Діаметр точок, не більше	Клас чистоти	Царапина, не більше	Діаметр точок, не більше
I, II	0,001	0,002	VI	0,008	0,04
III	0,002	0,004	VII	0,01	0,1
IV	0,004	0,01	VIII, VIIIa	0,0014	0,14
V	0,006	0,02	IX, IXa	0,02	0,2

*Таблиця 1.5 порівняльна характеристика двох груп класів чистот*и

	0-10, 0-20, 0-40	I, II, III, IV, V, VI, VII, VIII, VIIIa, IX, IXa
Вироби для яких використовується	Для поверхонь деталей розташованих в площині дійсного зображення або в площині предмету оптичної системи приладу, наприклад шкали, візири, сітки	Для поверхонь деталей що знаходяться поза площиною дійсного зображення або поза площиною предметів оптичної системи приладу, а також волоконноптичні вироби
Зони при вимірюванні	Ділиться на 3 зони контролю	Не ділиться на 3 зони контролю
Скупчення дефектів	Не контролюється	Нормується для класівI, II, III, IV, V, VI, VII

1.2. Обґрунтування потреби і способів автоматизації процесу

Традиційний метод контролю чистоти поверхонь оптичних деталей, що вже тривалий час застосовується на підприємствах що виробляють ці оптичні деталі,

полягає у візуальному огляді деталі працівником відділу внутрішнього технічного контролю. Під час процесу мають бути дотримані чіткі умови такі як використання оптичних приладів з потрібним збільшенням, знаходження деталі в умовах освітлення з косонаправленим пучком світла з потужністю джерела світла 60-100Вт. Огляд виконується шляхом порівняння виявлених на деталі дефектів з виміряними зразками царапин і точок. [3][7][11]

Ця методика контролю, незалежно від тривалого використання, має такі недоліки:

1. Суб'єктивність оцінювання. Людина – не ідеальне творіння, ми постійно під впливом емоцій, того як ми виспались останньої ночі, стосунків з іншими людьми, смаку ранкової кави, новин, обстрілів, стану здоров'я. Коли задачею є в умовах серійного або масового виробництва, на якому працюють різні зміни людей, повторюване спостерігання і вимірювання дуже малих дефектів з великою точністю – різні люди не можуть забезпечити повторюване і точне вимірювання без розбіжностей між вимірюваннями, зробленими різними людьми або вимірюваннями зробленими однією людиною в різний час.

2. Втомлюваність очей і необернена шкода для зору оператора. Багатогодинна робота з мікроскопом в умовах контрастного освітлення, втомлює людське око, що по-перше при довготривалому впливі спричиняє необернену шкоду для зору, по-друге втома може спричинити людину пропустити якийсь малий дефект що призведе до неправильної класифікації деталі, що може призвести до неправильної роботи виробу в який цю деталь потім поставлять.

3. Низька продуктивність виробництва. Через обмеження людського ока і терпіння, людина має брати паузу для відпочинку. Також сам процес порівняння дефектів з зразками для визначення є довгим і містить багато простору для похибки. Всі математичні операції по підрахунку сумарної довжини царапин, перевірка на скупчення дефектів і необхідність звірятись з таблицями роблять весь процес трудоємністним і довгим, в якому одна людина виконує одразу декілька задач, постійно переключаючись між ними, що за замовчуванням не є ознаками високоефективного виробництва.

4. Малооб’ємні і нестандартизовані або відсутні записи результатів вимірювання. Статистика вимірювань і деталей обмежується максимум тим, що людина записує результат контролю в журнал у вигляді висновку, без детального опису виявлених дефектів, або ж все детально записує, що кардинально негативно впливає на швидкість роботи людини і не гарантує об’єктивність і повторюваність статистики що збирається. Такий підхід до що унеможливлює статистичний аналіз і виявлення причин дефектів на попередніх етапах виробництва.

Зважаючи на 4 фактори описані вище, можна прийти до висновку що автоматизація процесу класифікації чистоти поверхонь скла є актуальною і дозволить:[7]

1. Виключити людський фактор, що в свою чергу забезпечить високу повторюваність і об’єктивність результату.

2. Суттєво підвищить швидкість і якість контролю,

3. Зменшить вартість виробництва виробів, за рахунок заміни вартості роботи працівника на амортизацію автоматизованої системи.

4. Дасть змогу збирати потрібні статистичні дані для аналізу недоліків і помилок виробничого процесу,

5. Дозволить інтегрувати модуль визначення класу чистоти в загальну систему автоматизованого управління виробництвом.

1.3. Класифікація і технічні параметри обладнання. Порівняльний аналіз існуючих систем визначення класу чистоти поверхні скла.

Стандарт ДСТУ11141-84 був прийнятий у 1985му році, до нього був стандарт 11141-76. Зараз 2026 рік, автоматизація оточує нас повсюди, Китай має повністю автоматизовані заводи машин які називають «темними», бо роботам не потрібне світло для роботи і через відсутність потреби, на цих заводах мінімальне освітлення. Стандарт контролю чистоти поверхні існує вже більше 40ка років, людство точно задумалось про питання автоматизації цього процесу. Так само як при змозі вирішити якусь проблему, або при проєктуванні будь-якої системи, потрібно пошукати готові

існуючі рішення, перед тим як проєктувати щось нове. В цьому розділі оглянуто 2 підходи до вирішення проблеми автоматизації контролю чистоти поверхні скла і оглянуто роботу деяких приладів які її вирішують реалізовуючи ці 2 підходи. [2][3][4] [5][7][11]

Сучасні рішення для автоматизованого контролю чистоти поверхні можна розділити на 2 групи за технологією:

1. Оптико-електричні системи, робота яких полягає в обробці і алгоритмічному математичному аналізі деталізованого двовимірного зображення поверхні, отриманого засобами мікроскопії при спеціальному освітленні. Точність таких систем теоретично обмежена якістю зображення камери, яка використовується в системі.[6][8][11][12]

2. Інтерферометричні системи робота яких базується на аналізі фазового зсуву і явищі інтерференції світлових хвиль, що дозволяє отримувати надточні тривимірні моделі поверхонь виробів і аналізувати їх на предмет дефектів. Точність таких систем теоретично обмежена довжиною хвилі світла. Технологія інтерферометрії відносно дорога, але в умовах серійного виробництва на довгій дистанції себе виправдовує.[10]

Далі розглянуто декілька систем що використовують як оптико-електричну так і інтерферометрійну технологію з розбором можливостей, обмежень, переваг і недоліків цих рішень.

1. Системи ARGOS

ARGOS є гнучкою системою для автоматизованого аналізу чистоти поверхонь сферичних і асферичних лінз, також ця система може працювати з дзеркалами. Система має можливість одночасного аналізу цілих касет лінз, або індивідуального аналізу одної деталі. Система є модульною, що дозволяє змінювати головку яка робить вимірювання для адаптації роботи системи під різний розмір виробів і точність вимірювань, також можна змінювати центруючу систему для лінзи, або підставку під касету щоб або аналізувати маленькі деталі поштучно, або великі деталі, або групу деталей одночасно.[16][17][18]

Система використовує безконтактний оптичний спосіб вимірювання – отримуючи зображення через камеру великої роздільної здатності і аналізуючи його.

Система має 3 варіації розмірів, що визначає максимальний розмір деталі яку можливо вимірювати, що складає 600 на 400мм. Є три опції вимірювальної головки, що забезпечують різну роздільну здатність що скаладає 3, 5 і 7,5 мікрометрів/піксель роздільної здатності з найменьшим розміром видимого дефекту 1, 2 і 3 мікрометра, чого як раз достатньо для контролю найякістніших класів чистоти 0-10 і I відповідно до ДСТУ 1141-84. Система вже має запрограмовану підтримку стандарту ISO 10110-7 і інших стандартів, і має можливість налаштування кастомних обмежень. Виріб має всі необхідні функції автоматизації такі як автоматичне сканування, документація дефектів і архівування результатів вимірювання. [16]

На рис. 1.1 зображені системи ARGOS matrix 200 і ARGOS matrix 400 і цифрами позначені такі компоненти: [18]

- Цифрою 1 позначена головка для вимірювань, саме її заміна дозволяє підлаштовуватись під різну точність вимірювання і різні деталі.

- Цифрою 2 позначена касета з лінзами які контролюються.

- Цифрою 3 – пристосування що тримає касету з лінзами, який можна замінити на пристосування для кріплення касет різного розміру (рис. 1.2), або затискаючий механізм для інспекції індивідуальних деталей (рис. 1.3).

- Цифрою 4 позначена велика система для вимірювання деталей розмірами до 600 на 400мм, що оснащена захисним кейсом.

Характеристики пристрою: [18]

- Вимірювання деталей розміром до 600 на 400 мм

- Роздільна здатність до 3 мікрометри / піксель

- Точність вимірювання до 1 мікрометра

- Можливість одночасного вимірювання великої кількості виробів

- Підтримка стандарту ISO 10110-7 з можливим перепрограмуванням.

Переваги даної системи:

- Висока точність оцінки порівняно з оцінкою людиною вручну

- Автоматизація всіх процесів які при виконанні людиною маються неточності

- Відносна недороговизна порівняно з іншими схожими системами

- Можливість одночасної оцінки великої кількості виробів.

Недоліки даної системи:

- Потреба ручної установки лінз в пристрій і забирання їх з пристрою
- Система не підійде для екстримально точного виробництва, наприклад дзеркал і або лінз для телескопів на орбіті через недостатню точність.

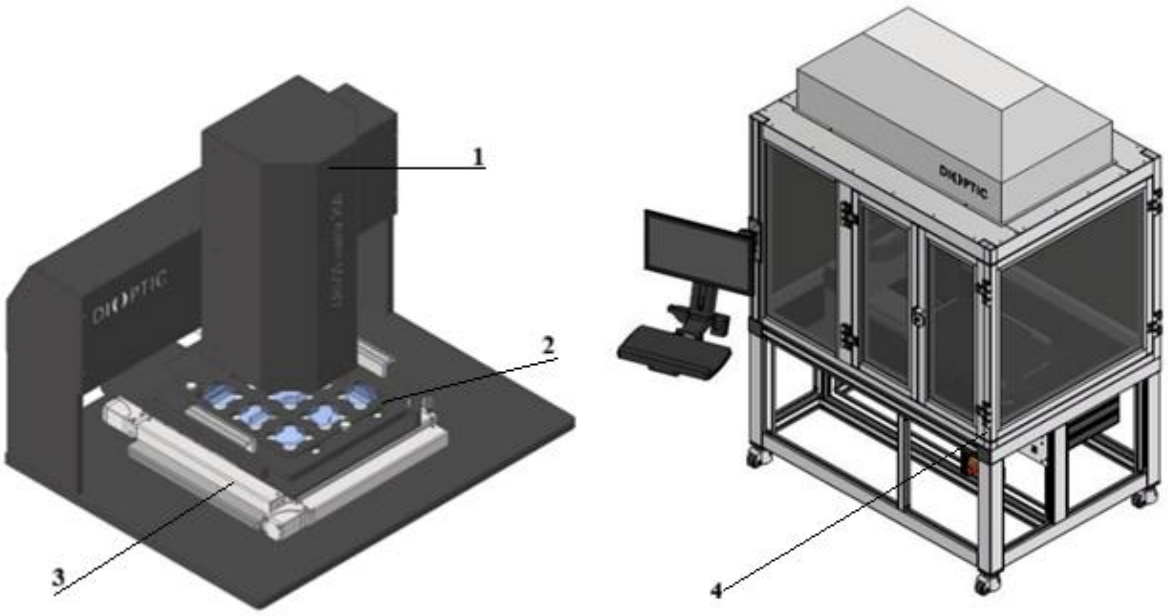


Рис. 1.1 Системи ARGOS matrix 200 (зліва) і ARGOS matrix 400 (справа)[18]

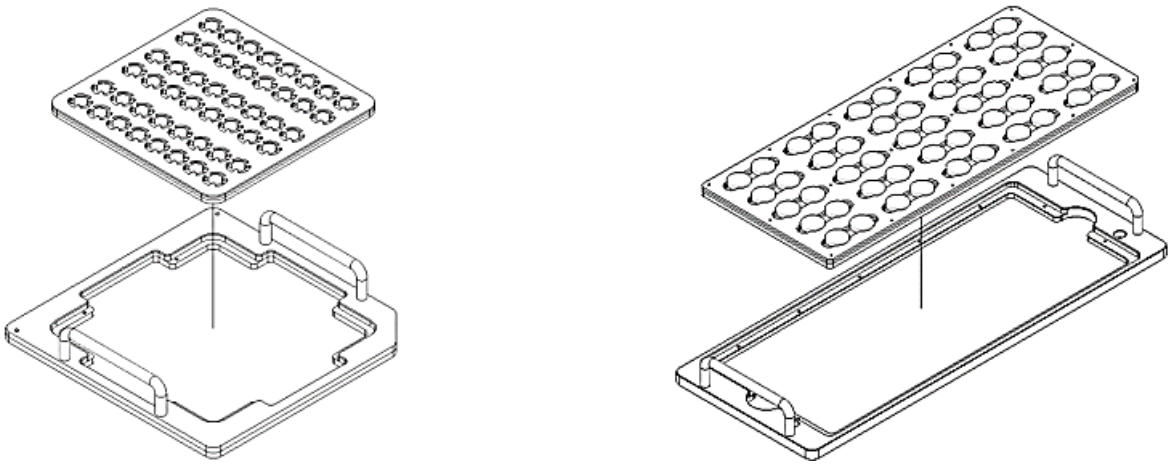


Рис. 1.2 Пристосування для каріплення касет [18]

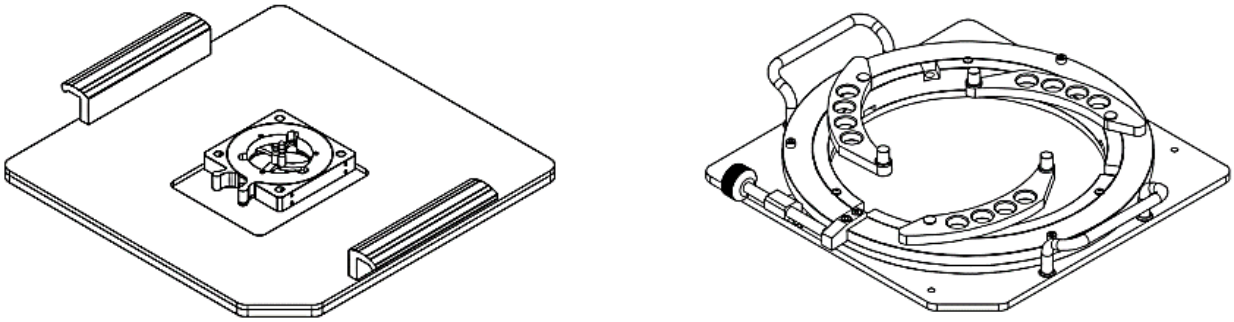


Рис. 1.3 Самоцентруючі установки для індивідуального контролю деталей діаметром до 45 мм(зліва) і до 200мм (справа)[18]

2. Системи компанії Zygo Corporation

Компанія Zygo Corporation виробляє системи що використовують інтерферометрію для аналізу оптичних виробів, один з таких виробів є ZeGage Pro HR (рис.1.4). Інтерферометр-профілометр ZeGage Pro HR відноситься до надточних і найсучасніших систем. На рис 1.4 позначені такі компоненти:[19][20]

- Цифрою 1 позначена головка виробу в якій знаходиться джерело випромінювання і матиця що зчитує відбите світло, всі компоненти головки, а саме освітлення і система генерації 3х вимірної моделі поверхні, керуються через програму, що дозволяє проводити вимірювання і збирати його результати найефективнішим способом.

- Цифрою 2 позначений модуль об’єктиву що має 2 модифікації – 1 об’єктив або автоматизована 4х позиційна турель що забезпечує максимальну автономність і універсальність.

- Цифрою 3 на позначена платформа з функціями ручного або автоматичного регулювання по 2м осям для забезпечення коректного розташування і автоматичного переміщення під час сканування деталі що закріплена на платформі.

- Цифрою 4 позначена основа профілометра, що забезпечує стабільність при вібраціях що є критично важливим для коректної роботи системи, де зміщення навіть на 1 нанометр може внести похибку що в майбутньому буде критичною.

- Всі автоматизовані компоненти системи контролюються програмним забезпеченням (рис. 1.4 позначення 5).

Пристрій використовує технологію когерентної скануючої інтерференції. Принцип роботи технологій полягає в тому що світловий потік розділяється на 2 – тестовий і референтний. Референтний відбивається від внутрішнього високоточного дзеркала. Тестовий потік відбивається від поверхні досліджуваної деталі. Референтний і тестовий потоки світла рекомбінуються і виникає інтерференційна картинка, що фіксується матрицею. Схема роботи профілометра на рис. 5. Отримавши інформацію з матриці, прилад розраховує фазовий зсув у кожній точці матриці, і будує трьохвимірну модель поверхні деталі, не обмежуючись тільки двома вимірами, а фіксуючи і глибину дефектів, що відкриває можливості для точнішого контролю ніж системи що враховують тільки 2 виміри.[19][20]

Характеристики пристрою у таблиці 1.6

Таблиця 1.6 Характеристики ZeGage Pro HR[20]

Точність вимірювання	До 3,5 нм
Швидкість сканування	10,7 – 171 мкм/сек
Матриця	1,9 млн чутливих точок
Джерело світла	Інтегроване з функцією автоматичного контролю
Стіл для фіксації деталей	Резулюється по 2м осям вручну і автоматично
Можоивості програмного забезпечення	Керування освітленням, столом для деталей, проведенян аналізу, зберігання результатів аналізу

Переваги даної системи: [19][20]

- Надвисока точність
- Наближена до максимальної автоматизація
- Повна інтеграція з електронними системами.

Недоліки даної системи:

- Ціна, що на порядки вища за системи з оптичним способом вимірювання
- Відносно повільна швидкість роботи, що випливає з необхідності надточного

вимірювання, яке в багатьох випадках є надлишковим.

Такий пристрій є актуальним тільки при серійному виробництві високотехнологічних приладів або при проведенні досліджень що потребують надвисокої точності вимірювань.

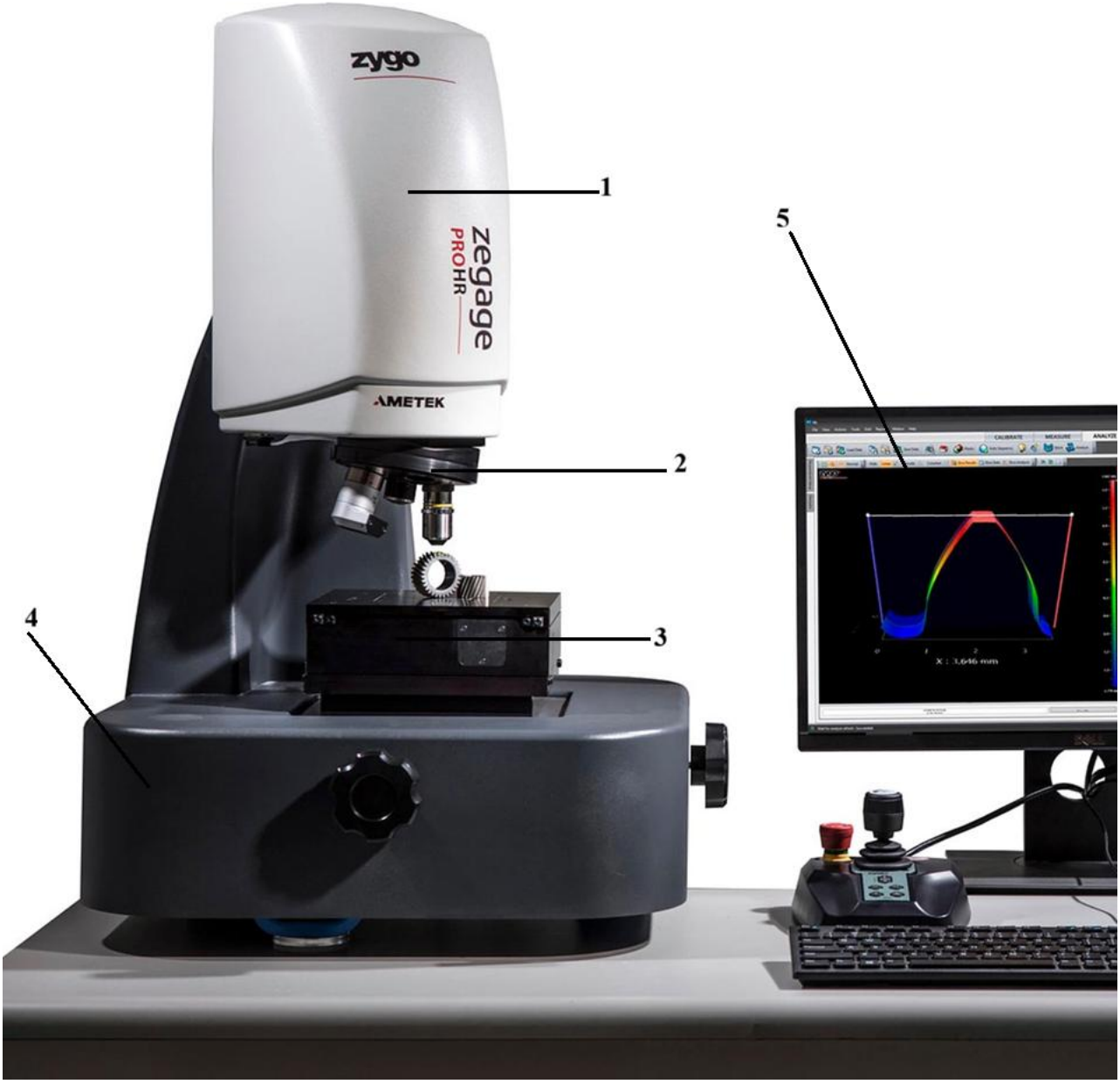


Рис. 1.4 Профілометр Zygo ZeGage Pro HR[20]

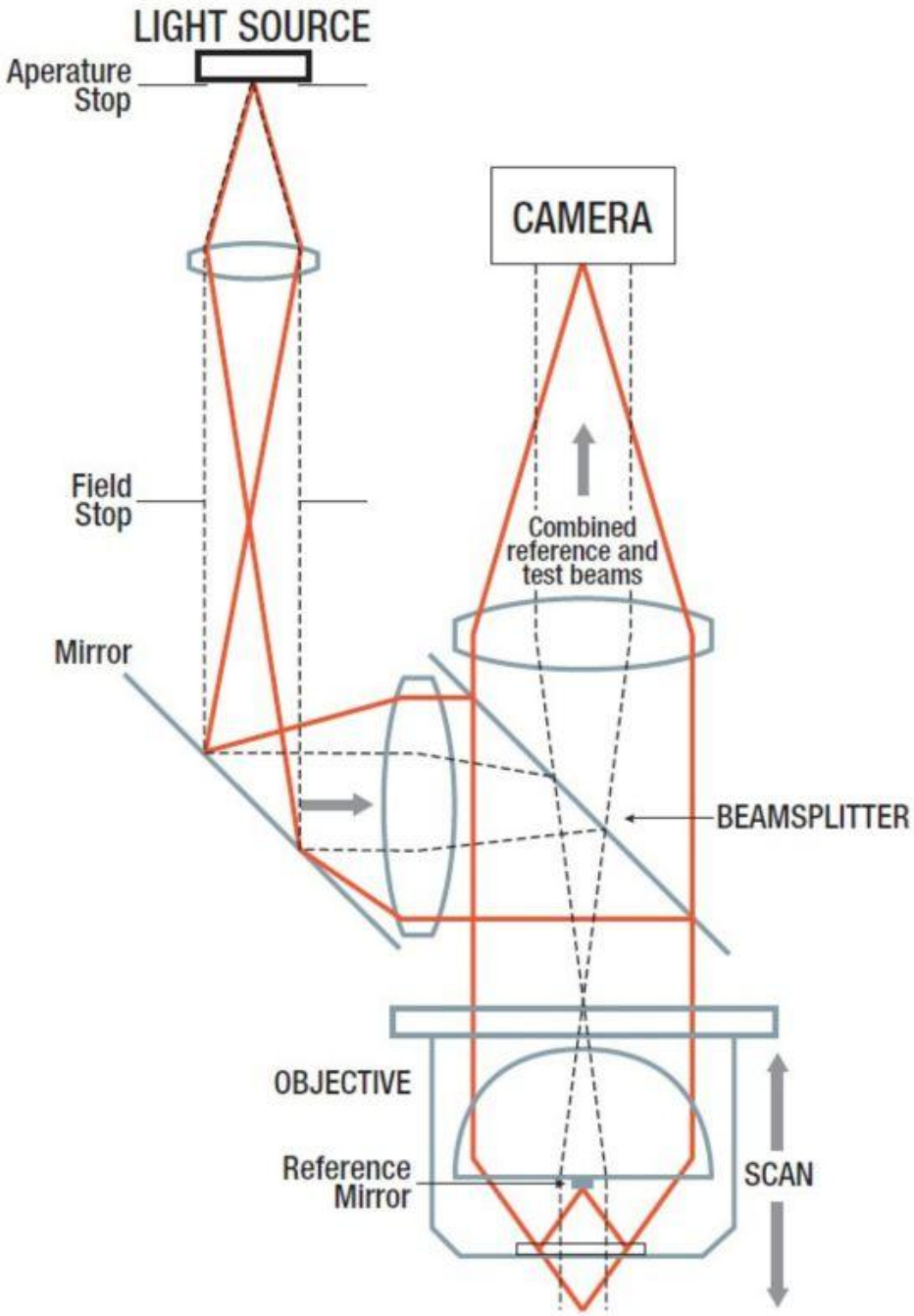


Рис. 1.5 Схема роботи профілометра [21]

Проаналізувавши спосіб контролю чистоти поверхні вручну, оптико-електричну систему ARGOS і інтерферометричний профілометр від Zygo, я склав порівняльну характеристику систем контролю чистоти (рис.1.6).

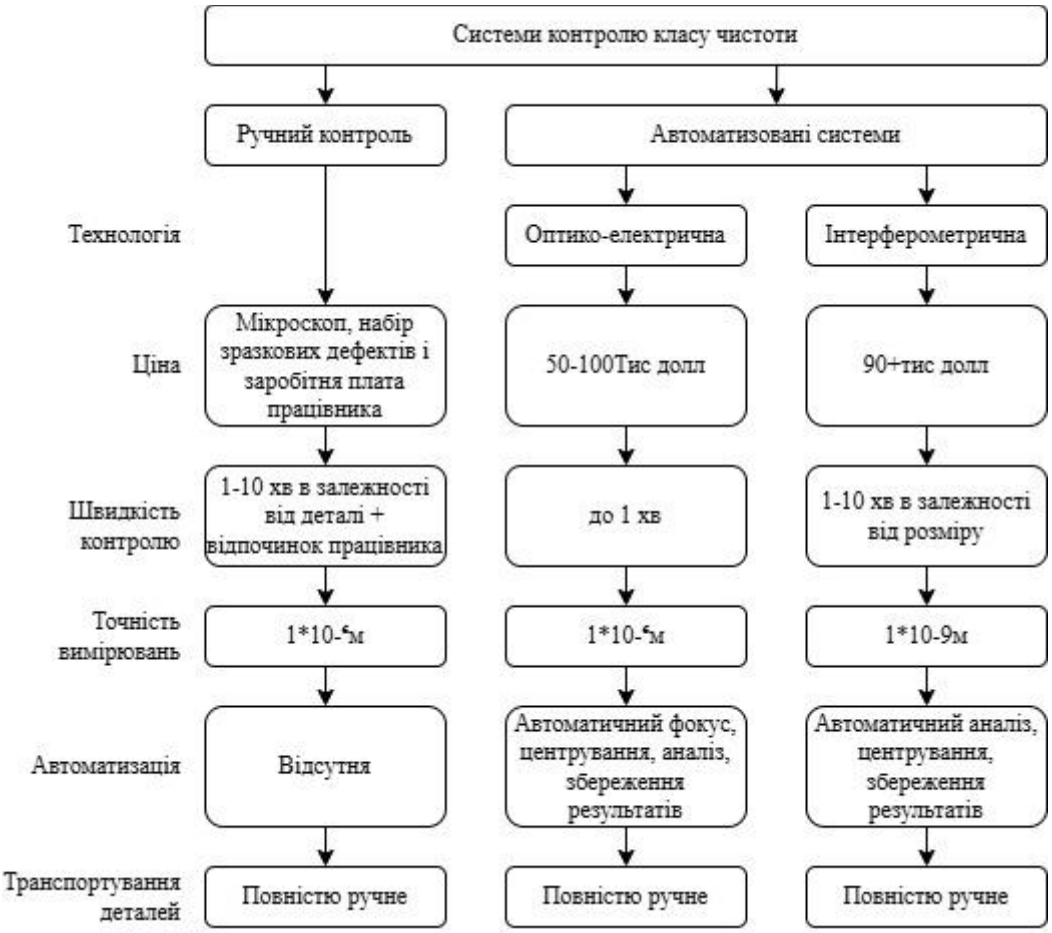


Рис.1.6 Порівняльна характеристика ручного методу контролю чистоти поверхні, оптико-електричної системи ARGOS і інтерферометричної системи Zygo

1.4. Постановка задачі

Провівши аналіз підходів до реалізації системи контролю класів чистоти можна виділити такі недоліки і переваги різних систем:

Недоліки:

- Відсутність автоматизації транспортування лінз в усіх типах систем. Скло – дуже чутливий до пошкоджень матеріал, і коли задачею системи є саме перевірка дефектів на ньому, підходити до транспортування скла потрібно з особливою обережністю. Якщо автоматизувати його транспортування в і з системи контролю класу чистоти, це виключить ризик пошкодження що позитивно вплине на відсоток браку і економіку підприємства що виробляє деталі.

- Довгий процес огляду людиною і великий час сканування профілометром. Беззапеерчно, точність яку може запропонувати профілометр є недосяжною іншими методами, але коли йде мова про розпізнавання дефектів розміром до 0,001мм, точність профілометра є на декілька порядків надлишковою, і за цю точність потрібно платити часом роботи обладнання, що збільшує вартість одного виробу. Тож автоматична система має бути побудована на оптико-електричній технології, де час, потрібний для аналізу поверхні однієї деталі, суттєво зменшується в порівнянні з іншими системами, і є багато простору для подальшої оптимізації процесу за рахунок оптимізації алгоритмів обробки зображення.

- Дороговизна існуючих автоматичних систем. Як і все обладнання для автоматизації – системи ARGOS і ZYGO мають цінники в декілька десятків тисяч долларів. Для українського виробництва в час війни купівля такого обладнання є великим і часто не виправданим фінансовим ризиком. Будь яка компанія в світі булаби рада якби технологія якою вони користуються впала в ціні в декілька разів без втрати характеристик. Отже все вказує на те що потрібно робити не найкращу систему за великі гроші, а ту, якої буде вистачати для виконання поточних задач за найменьші можливі гроші.[22]

Переваги:

- Швидкість вимірювань. З 3х розглянутих систем – оптико-електрична система працює найшвидше – це є ключовим фактором при плануванні виробництва, що є вирішальним аргументом при виборі системи яку будуємо.

- Автоматизація процесу вимірювань і автоматична фіксація його результату, що дозволяє виключити ризик виникнення людських помилок, гарантує стабільність і безперебійність виробництва і відкриває можливість до його аналізу і покращення.

Проєктована система має забезпечити повну автоматизацію операції технічного контролю чистоти поверхні у складі діючого виробничого циклу. Вхідними об’єктами системи є касети з лінзами, клас чистоти поверхонь яких не є визначеним. Система має реалізовувати поштучне інспектування деталей та подальше їх сортування на 2 потоки за ознакою відповідності або невідповідності вимозі по класу чистоти поверхні деталі.

Отже, які задачі має виконувати автоматизована система визначення класу чистоти поверхні скла:

- Модуль транспортування деталей до станції автоматичного аналізу.

- Розташування деталей в правильному місці модулю отримання зображення для ефективної роботи системи.

- Отримання зображення деталі, клас чистоти поверхні якої потрібно визначити.

- Обробка зображення і призначення класу чистоти її поверхні відповідно до ДСТУ 11141-84 в програмному модулі системи.

- Приймання деталі з модулю отримання зображення і транспортування її в потрібне місце в залежності від результату.

- Всі аспекти процесу мають керуватись однією керуючою програмою.

Сформувавши задачі можна чітко побачити що система має мати 3 модулі:

- Апаратний модуль транспортування деталей.

- Апаратний модуль отримання зображення.

- Програмний модуль обробки зображення і визначення класу чистоти.

ВИСНОВКИ ДО РОЗДІЛУ 1

1.1. Аналіз чинного стандарту ДСТУ 1141-84 встановлює чіткі та однозначні допуски для класифікації чистоти поверхонь оптичних деталей. Допуски полягають у точному контролі розмірів і зон розташування таких дефектів, як точки і царапини, а також їх скупчень. Процес фіксації і підрахунку цих дефектів є дуже трудоємністним, рутинним і потребують математичних операцій. Ці фактори створюють передумови для автоматизації процесу засобами математичного і алгоритмічного його опису.[7][11]

1.2. Візуальний контроль, що виконується працівником ВТК вручну, має багато простору для суб'єктивізму, низької продуктивності виробництва і підвищеної кількості помилок по різним людським причинам. Ці фактори обґрунтовують необхідність і актуальність автоматизації процесу контролю класів чистоти поверхонь.

1.3. Проведений аналіз існуючих систем доводить актуальність питання і можливість його вирішення.

1.4. Результати аналізу проведеного в цьому розділі є теоретичним підґрунтям для подальшого проєктування структури автоматизованої системи.

РОЗДІЛ 2 ПРОЄКТУВАННЯ АПАРАТНОЇ ЧАСТИНИ АВТОМАТИЗОВАНОЇ СИСТЕМИ ВИЗНАЧЕННЯ КЛАСУ ЧИСТОТИ ПОВЕРХНІ СКЛА

2.1. Розробка структурної схеми автоматизованої системи

Проєктування структурної схеми автоматизованої системи визначення класу чистоти поверхні скла базується на комплексному аналізі вхідних і вихідних метеріальних та інформаційних потоків, а також на декомпозиції загального технологічного процесу на окремі функціональні блоки.

Також система потребуватиме попе[реднього ручного налаштування людиною, для роботи системи з новими деталями. Без ручного налаштування потрібно використовувати мікроскоп з функціями автоматичного фокусування і регулювання наближення, що суттєво збільшить вартість системи.

На основі системного аналізу, проведеного в підрозділі 1.4, доцільно розділити архітектуру автоматизованої схеми на 4 взаємопов’язані модулі:

1. Апаратний модуль отримання зображень поверхні деталі.
2. Програмний модуль обробки зображення деталі, виявлення дефектів поверхні та класифікації деталі за чистотою поверхні.
3. Програмний модуль формування команд для промислового робота.
4. Апаратний модуль транспортування, позиціонування та сортування деталей.

Апаратний модуль отримання зображень поверхні деталі

Функцією цього модулю є реєстрація зображення поверхні деталі і введення його в програмний модуль обробки зображення.

До складу модуля входять такі елементи:

- Оптична система. Функцію оптичної системи виконує мікроскоп, що забезпечує необхідне оптичне збільшення, фокусування на поверхні досліджуваної деталі і має можливість інтеграції з цифровою камерою.
- Цифрова камера. Виконує функцію оптико-електронного перетворювача, що

трансформує світловий потік у матрицю пікселів що піддається обробці комп’ютером.

- Освітлення. Створює необхідні умови для контрастування дефектів, що є критично необхідним для ідентифікації дефектів на поверхні деталей з прозорого скла.

Програмний модуль обробки зображення деталі, виявлення дефектів поверхні та класифікації деталі за чистотою поверхні

Модуль є інтелектуальним ядром системи, що виконує такі функції:

- Імпорт зображення з цифрової камери.

- Обробка зображення. Усунення шумів, виділення дефектів.

- Логіко-математична обробка отриманих даних про дефекти, визначення класу чистоти деталі шляхом порівняння з критеріями стандарту ДСТУ 11141-84.

Програмний модуль формування команд для промислового робота

Модуль є програмою що керує роботою апаратного модуля транспортування. Виконує такі функції:

- Подання команд на захват деталі.

- Подання команд на позиціонування деталі.

- Подання команд на відпускання деталі.

- Формування команди на позиціонування деталі в залежності від результату роботи програмного модуля обробки зображення деталі.

Апаратний модуль транспортування, позиціонування і сортування деталей.

Цей модуль є виконавчою підсистемою що реалізує фізичне переміщення, прицизійне позиціонування в робочій зоні модулю отримання зображення, і розділення за результатами класифікації на два потоки деталей – об’єктів контролю всередині автоматизованої системи визначення класу чистоти поверхні скла. До складу модуля входять:

- Промисловий робот.

- Робочий орган промислового робта. Захватуючий механізм, здатний забезпечити захват і утримання скляних деталей без їх пошкодження.

- Вхідний та вихідні потоки деталей.

Врахувавши всі вимоги і складові системи, було складено структурну схему автоматизованої системи визначення класу чистоти поверхні скла (рис. 2.1).

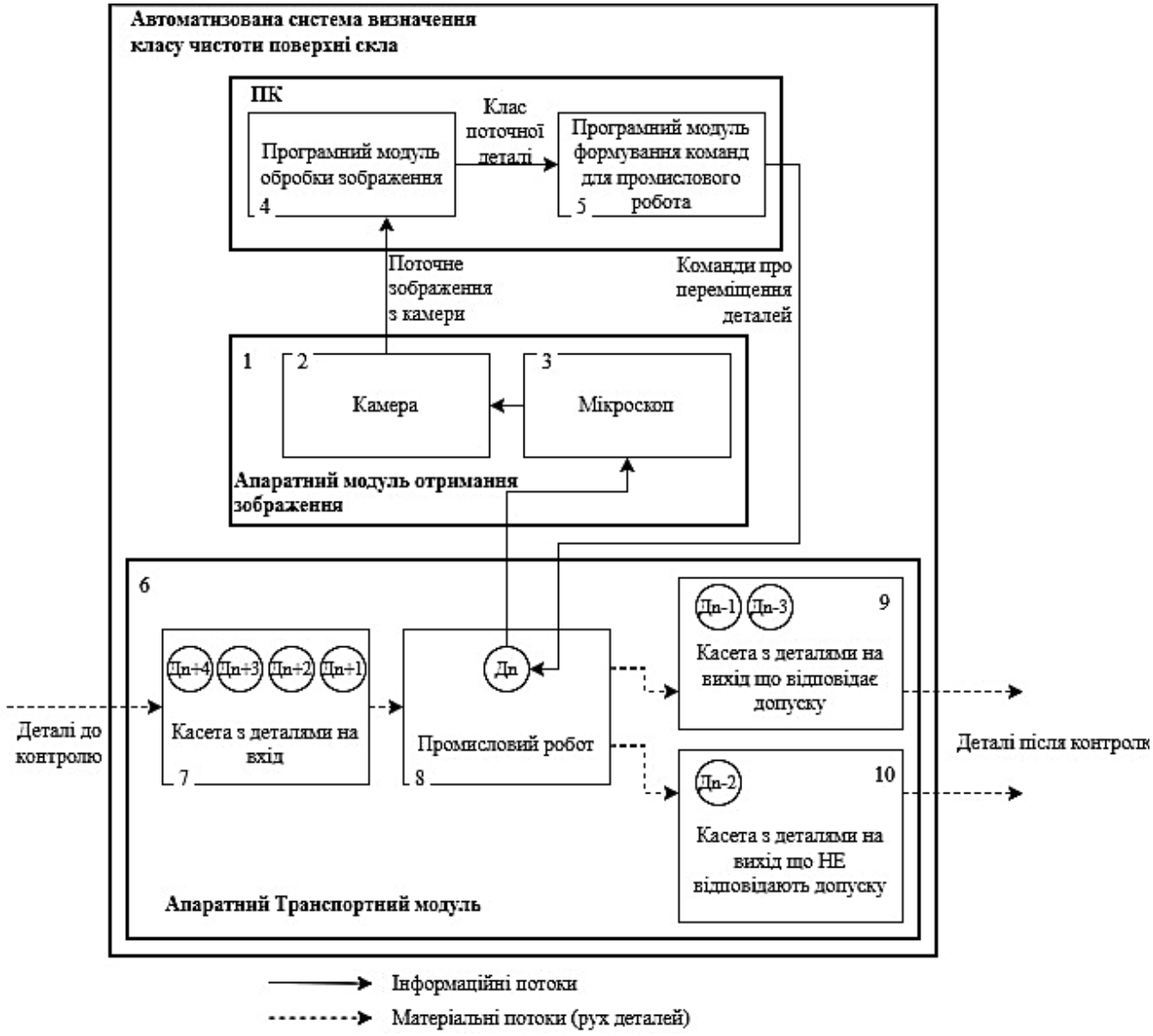


Рис. 2.1. Структурна схема автоматизованої системи визначення класу чистоти поверхні скла.

Пояснення до елементів схеми (див. позначення 7 на рис. 2.1):

1. Апаратний модуль отримання зображення. Він складається з камери і мікроскопа, Промисловий робот переміщає деталі в цей модуль для отримання

зображення поверхні деталі.

2. Камера в апаратному модулі отримання зображення. Через мікроскоп матриця камери сприймає відбите від поверхні деталі світло і фіксує зображення що потім буде оброблятись в програмному модулі обробки зображення.

3. Мікроскоп. Забезпечує необхідне для розпізнавання маленьких дефектів наближення.

4. Програмний модуль обробки зображення. Це програмне забезпечення яке на вхід отримує зображення поверхні деталі і на вихід видає клас чистоти. Всередині відбувається аналіз дефектів на поверхні деталі їх співставлення з вимогами в ДСТУ11141-84.

5. Програмний модуль формування команд для промислового робота. Цей модуль теж є частиною програмного забезпечення, він отримує висновки стосовно класів чистоти деталей і приймає рішення про рухи промислового робота, що пливають на положення деталі.

6. Апаратний транспортний модуль. Це модуль системи що відповідає за фізичне переміщення деталей клас чистоти поверхні який потрібно перевірити. Робот бере деталь з вхідної касети, переміщає її в модуль отримання зображення і в залежності від результату обробки зображення поверхні деталі, переміщає деталь на одну з вихідних касет.

7. Касета з деталями на вхід. Задача цього елементу – приймати деталі з попередніх операцій виробництва і накопичувати їх.

8. Промисловий робот. Складається з маніпулятора і робочого органу – захватуючого пристрою. Фізично захоплює і переміщує деталі всередині системи.

9. Вихідний касетний накопичувач для деталей що пройшли перевірку і відповідають допуску на клас чистоти поверхні.

10. Вихідний касетний накопичувач для деталей що пройшли перевірку і не відповідають допуску на клас чистоти поверхні.

11. Пунктирними стрілками на схемі показані матеріальні потоки, тобто шлях руху деталей. Деталі приходять до системи в касетах, промисловий роботи їх захоплює, поміщає в модуль отримання зображення, камера спрацьовує,

промисловий робот переміщає деталь в одну з вихідних касет в залежності від результату аналізу.

12. Стрілками на схемі позначені інформаційні потоки що формують зворотній зв’язок системи. Деталь має дефекти, камера знімає зображення цих дефектів через мікроскоп, передає зображення на обробку в програмний модуль обробки зображення, результати обробки, тобто клас чистоти деталі, передається в модуль формування команд для промислового робота, який в свою чергу в залежності від класу чистоти деталі формує команду на переміщення деталі для промислового робота, який впливає на деталь, переміщуючи її.

2.2. Розробка структурної схеми загального алгоритму

Функціонування розроблюваної системи визначення класу чистоти поверхні скла має поетапний характер і базується на чіткій взаємодії апаратних і програмного модулів. Для формалізації логіки роботи системи було розроблено структурну схему загального алгоритму системи (рис. 2.2).

Структурна схема загального алгоритму відображає наскрізний процес визначення класу чистоти поверхні скла – від моменту приходу касети з некласифікованими виробами, до розподілу виробів в залежності від класу чистоти.

Опис логіки алгоритму роботи системи доцільно розділити на декілька послідовних стадій:

- Ініціалізація та первинне налаштування системи перед контролем партії виробів.
- Перший транспортний цикл.
- Оптико-електрична реєстрація і цифровий аналіз.
- Другий транспортний цикл – сортування.

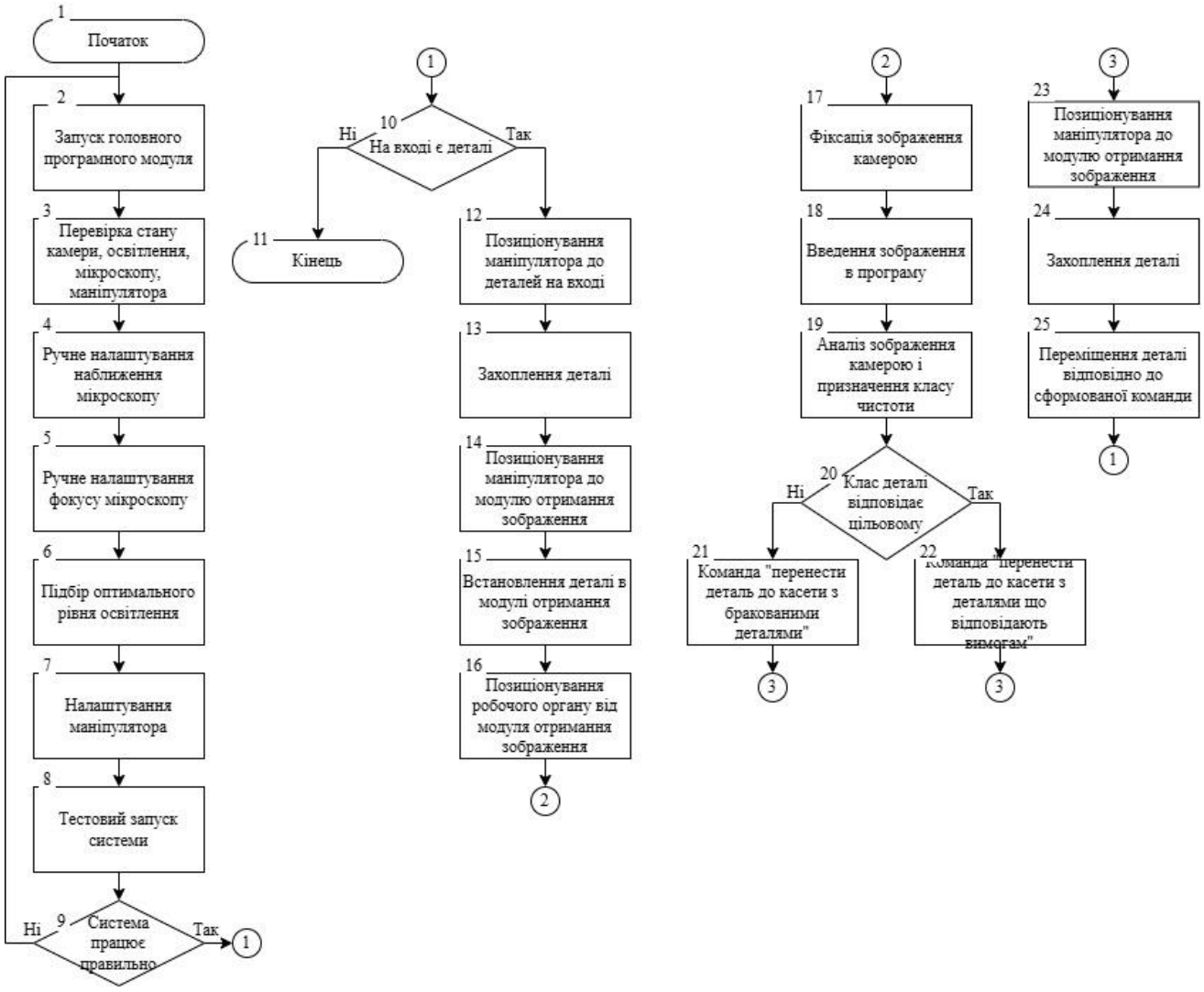


Рис. 2.2 Структурна схема загального алгоритму автоматизованої системи визначення класу чистоти поверхні скла

Стадія ініціалізації та первинного налаштування системи перед контролем партії виробів

Ця стадія алгоритму передбачає підготовку автоматизованої системи людиною перед початком роботи з новою партією виробів, вона включає такі кроки:

- Запуск головного програмного модуля (рис.2.2 позначення 2).

- Перевірка стану камери, освітлення, мікроскопа, маніпулятора (рис.2.2 позначення 3).

- Ручне налаштування наближення мікроскопу в залежності від цільового класу чистоти, відповідно до вказівок в стандарті ДСТУ11141-84 (рис.2.2 позначення 4).

- Налаштування фокусу мікроскопу під формфактор деталей в партії (рис.2.2 позначення 5).

- Підбір оптимального рівня освітлення (рис.2.2 позначення 6).

- Налаштування маніпулятора. Збереження позиції маніпулятора для встановлення деталей в модуль отримання зображення і приймання їх після проведення аналізу. Збереження позиції маніпулятора при якій робочий орган знаходиться якумога ближче до деталі в модулі отримання зображення, але робочого органу не видно на зображенні (рис.2.2 позначення 7).

- Тестовий запуск системи, якщо фото в фокусі, дефекти виявляються і маніпулятор правильно встановлює і приймає деталі з модулю отримання зображення – можна запускати автоматичну роботу системи (рис.2.2 позначення 8, 9, 10, 11).

Перший транспортний цикл

Після успішного старту система переходить до виконання ітераційного циклу оцінки класу чистоти поверхні кожної деталі. Перший транспортний цикл полягає в переміщення маніпулятором деталі з вхідного потоку в модуль отримання зображення і включає такі кроки:

- Переміщення робочого органу промислового робота до вхідного потоку некласифікованих деталей (рис.2.2 позначення 12).

- Захоплення деталі робочим органом промислового робота (рис.2.2 позначення 13).

- Переміщення робочого органу промислового робота до модуля отримання зображення (рис.2.2 позначення 14).

- Встановлення деталі в модуль отримання зображення (рис.2.2 позначення 15).

- Переміщення робочого органу в позицію коли його не видно на зображенні (рис.2.2 позначення 16).

Оптико-електрична реєстрація і цифровий аналіз:

Ця стадія задіює апаратний модуль отримання зображення і програмний модуль обробки зображення для винесення вердикту стосовно відповідності або невідповідності поточної деталі цільовому класу чистоти.

- Фіксація зображення камерою (рис.2.2 позначення 17).

- Введення зображення в програму (рис.2.2 позначення 18).

- Аналіз зображення програмою і призначення класу чистоти (рис.2.2 позначення 19).

- Винесення висновку і формування команди стосовно місця подальшого призначення деталі (рис.2.2 позначення 20, 21, 22).

Другий транспортний цикл – сортування:

- Переміщення робочого органу маніпулятора на позицію приймання деталі з модуля отримання зображення (рис.2.2 позначення 23).

- Захоплення деталі робочим органом маніпулятора (рис.2.2 позначення 24).

- Переміщення робочого органу маніпулятора до касети для деталей що відповідають вимогам класу чистоти, або до касети для деталей що не відповідають вимогам класу чистоти в залежності від попередньо сформованої команди (рис.2.2 позначення 25).

2.3. Підбір компонентів для апаратного модулю отримання зображень поверхні деталі

Ефективність та метрологічна надійність роботи системи визначення класу чистоти поверхні скла безпосередньо залежать від первинної графічної інформації. Апаратний модуль отримання зображення виконує роль первинного вимірювального перетворювача, який повинен трансформувати фізичні мікродефекти скляної поверхні у цифровий сигнал без внесення додаткових спотворень.

Для реалізації цього процесу приведено системний аналіз технічних завдань модуля, на основі результатів якого здійснено раціональний підбір оптичної системи та цифрової камери.

При проєктуванні та комплектації модуля отримання зображення було визначено такі ключові технологічні та інженерні завдання:

- Забезпечення достатньої роздільної здатності для ідентифікації дефектів розміром до 0,001мм.

- Сумісність з промисловим роботом. Конструкція оптичного блока має мати достатню робочу відстань для безперешкодного позиціонування лінз промисловим роботом.

- Оптична система має бути сумісна з камерою.

- Програмно-апаратна інтеграція. Цифрова камера має підтримувати високошвидкісну передачу даних до керуючої програми для подальшої обробки зображень.

Вибір, обґрунтування та характеристики оптичної системи

Як базову оптичну систему обрано стереоскопічний мікроскоп МБС-10 (рис. 2.3). Даний прилад є промисловим оптичним стандартом для проведення високоточних інспекцій та препарувальних робіт. Його основні технічні характеристики наведено в таблиці 2.1.[25]

Рис. 2.3 Стереоскопічний мікроскоп МБС-10 [25]

Таблиця 2.1. Технічні характеристики стереоскопічного мікроскопу МБС-10 [25]

Параметр	Значення параметра
Збільшення, крат, в межах	3,3 – 100,8
Лінійне поле зору, мм, в межах	39-2,4
Робоча відстань, мм, не меньше	95
Джерело світла Галогенна лампа РН8-20-1	12/20Вт
Маса прибору, кг, не більше	8
Тип освітлення	Відбите або прохідне світло

Технічне обґрунтування вибору МБС-10 для автоматизованої системи: [25]

- Велика робоча відстань більше 95мм, що є критичним фактором для інтеграції з апаратним модулем транспортування. Це дозволяє робочому органу промислового

робота вільно та безпечно переміщувати лінзи діаметром від 5 мм, виключаючи ризик механічного зіткнення деталей чи вузлів робота з оптикою.

- Універсальність системи освітлення. Наявність вбудованих блоків прохідного та відбитого світла дозволяє гнучко налаштувати схему косого або темнопільового освітлення, що є обов'язковою умовою для виявлення прозорих дефектів на прозорому фоні оптичного скла.

Вибір, обґрунтування та характеристики оптико-електронного перетворювача - камери

Для реєстрації спрямованого мікроскопом оптичного зображення та його конвертації у цифровий формат обрано спеціалізовану цифрову камеру-окуляр SIGETA MDC-500 5.0 MP. Зображення SIGETA MDC-500 5.0 MP на рис. 2.4. Технічні параметри сенсора та інтерфейсної частини камери наведено в таблиці 2.3.1.[26]

Рис. 2.4 SIGETA MDC-500 5.0 MP [26]

Таблиця 2.2 Технічні характеристики камери-окуляра SIGETA MDC-500 5.0 MP [26]

Параметр	Значення параметра
Тип матриці	CMOS
Розмір сенсора	4,96 x 3,54 мм
Максимальна роздільна здатність матриці	2592 x 1994 px
Роздільна здатність	5 МП
Розмір пікселя	2,2 x 2,2 мкм
Швидкість зйомки	3 кадри/сек при 2592 x 1994 px

Динамічний діапазон	69Дб
Світлочутливість для довжини хвилі 550нм	550 нм: 3В/люкс-сек
Спектральний діапазон	380-650 нм (з ІЧ фільтром)
Тип затвора	ERS
Баланс білого	є, авто
Контроль експозиції	є, авто
Інтерфейс передачі даних	USB 2.0
Швидкість передачі даних	480 Мб/с
Живлення	USB
Оптичне збільшення	10х
Посадковий діаметр	23,2мм
Тип кріплення	До оптичного порту 23,2мм, адаптер 30мм, адаптер 30,5мм
Робоча температура	Від -10 °С до +50 °С
Температура зберігання	від -20 °С до +60 °С
Допустима вологість при роботі	30-80 %
Допустима вологість для зберігання	10-60 %

Технічне обґрунтування вибору SIGETA MDC-500 5.0 MP для автоматизованої системи: [26]

- Висока дискретизація 5 МП. Максимальна роздільна здатність 2594 х 1944 рх у поєднанні з малим розміром пікселя 2,2мкм і приближенням мікроскопа до 100,8 разів забезпечує високу щільність просторової вибірки. Це дозволяє програмному модулю фіксувати дефекти розміром до 0,001мм, що задовільняє вимоги ДСТУ11141-84 щодо точності вимірювань.

- Протокол передачі даних та інтерфейс USB 2.0. Швидкість передачі даних 480 Мб/с дозволяє реалізувати стабільний режим передачі зображення. Живлення від порту USB спрощує загальну схему електроживлення комплексу, усуваючи потребу в додаткових силових блоках.

- Конструктивна сумісність з мікроскопом МБС-10. Відсутність потреби проєктування додаткових перехідних оптико-механічних вузлів.

Оцінка сумісної роботи компонентів у складі автоматизованої системи

Спільна інтеграція мікроскопа МБС-10 та камери SIGETA MDC-500 5.0 MP повністю перекриває завдання модуля отримання зображення в автоматизованій системі визначення класу чистоти поверхні скла. Формування вимірювального тракту: світловий потік, відбитий від дефектів лінзи, проходить крізь оптичну систему МБС-10 із налаштованим коефіцієнтом збільшення, після чого додатково збільшується оптикою камери і фокусується на CMON матриці яка фіксує світло і надсилає зображення в програму для оцінки класу чистоти.[25][26]

ВИСНОВКИ ДО РОЗДІЛУ 2

2.1. На основі системного аналізу розроблено структурну схему автоматизованої системи що розділена на 3 модулі: апаратний модуль отримання зображень, програмний модуль обробки зображень і апаратний модуль транспортування і сортування деталей. Така модульна архітектура забезпечує максимальну зручність використання, дозволяє швидко ідентифікувати несправності у разі їх виникнення і залишає можливості для покращення системи.[24]

2.2. Роботу автоматизованої системи описано в структурній схемі загального алгоритму, яка чітко регламентує послідовність технологічних процесів від стадії ручної ініціалізації системи і її налаштування, до роботи транспортних циклів, процесу отримання зображення, його обробки і сортування деталей за ознакою відповідності допускам ДСТУ 11141-84.

2.3. Технічно обґрунтовано вибір стереоскопічного мікроскопа МБС-10 в якості оптичної системи. Визначено що його параметр робочої відстані від 95 мм є критичним для автоматизованого транспортування деталей виробничим роботом, оскільки надає достатньо простору для руху, унеможливлюючи механічне пошкодження оптичної системи або деталей під час позиціонування лінз. Наявність вбудованого освітлення дозволяє гнучко налаштувати схему освітлення для якісного контрастування дефектів.

2.4. Технічно обґрунтовано вибір оптико-електронного перетворювача – цифрової камери SIGETA MDC-500 5.0 MP з роздільною здатністю 5МП. Параметри CMON матриці, а саме розмір пікселя 2,2 х 2,2 мкм і власний зум 10х у комбінації з оптичним збільшенням мікроскопа забезпечують достатню якість зображення для виявлення на зображенні дефектів розмірами до 0,001мм, повністю задовільняючи вимоги ДСТУ 11121-84 щодо точності вимірювань.

РОЗДІЛ 3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Вибір та обґрунтування мови програмування і бібліотек

Для розробки програмного забезпечення системи визначення класу чистоти поверхні скла мовою програмування було вибрано Python. Вибір обґрунтовано такими первагами:

Простота використання

Python є мовою високого рівня, що означає що ця мова програмування було розроблена для максимальної зручності людини, на противагу мовам низького рівня, на кшталт Assembler, що розроблялись максимально наближено до машинного коду. Python має купу вбудованих функцій, структур даних і методів, використання яких в порівнянні з мовами низьких рівнів кардинально зменшує об’єм коду що потрібний для реалізації функціоналу, що значно полегшує написання коду програміста. Зручність і розповсюдженість цієї мови програмування спонукнула людей створювати бібліотеки функцій, використання яких ще більше полегшує написання програми.[27]

Перелік бібліотек що були використані і реалізованого за допомогою инх функціоналу:

- OpenCV. Назва розшифровується як “Open Computer Vision Library”. Це бібліотека для комп’ютерного зору, роботи з зображеннями і реалізації машинного навчання. Вона містить понад 2500 алгоритмів що спрощують роботу програміста. Її функціонал був використаний для обробки зображень поверхонь лінз, а саме для фільтрації шумів, адаптивної бінаризації, пошуку і обробки контурів дефектів.[28]

- NumPy. Бібліотека була використана для роботи з зображеннями у формі масивів з числами.[29]

- Tkinter. Ця бібліотека надає функціонал для побудови графічного інтерфейсу. Основне вікно програми, вікна налаштувань і списку дефектів і все що знаходиться всередині цих вікон і з чим може взаємодіяти користувач було зроблено за рахунок функцій цієї бібліотеки.[30]

- Pillow. Бібліотека Pillow має функціонал обробки зображень, але в даній

програмі вона використовувалась для перетворення формату зображення OpenCV на формат що підходить для відображення в візуальному інтерфейсі Tkinter.[31]

Обраний набір інструментів забезпечує необхідні можливості для написання програми з потрібним функціоналом.[13][14][15][23]

3.2. Архітектура і узагальнена блок-схема програмного забезпечення

У цьому розділі описана архітектура розробленого програмного забезпечення і розроблена узагальнена блок-схема програмного забезпечення. Описана робота всієї програми від введення зображення деталі з дефектами до отримання результату обробки і збереження обробленого зображення.

Узагальнена блок-схема зображена на рис. 3.1. Пояснення елементів блок-схеми:

- Блок 1 – початок роботи.

- Блок 2 «Запуск програми». Початок виконання коду – запуск програми, створюється об’єкт вікна програми з усіма складовими і іншими вікнами, відмальовується графічний інтерфейс.

- Блок 3 «Завантаження зображення поверхні деталі». Вибір файлу зображення поверхні деталі на якому шукати дефекти.

- Блок 4 «Налаштування співвідношення мм/px». Ввід користувачем значення співвідношення мм/px у вікні «Параметри обробки».

- Блок 5 «обробка зображення». Відбувається по натисканню кнопки «обробити» в інтерфейсі основного вікна. З зображення прибираються шуми, відбувається адаптивна бінаризація, пошук контурів дефектів, обробка інформації про контури і формування результатів обробки.

- Блок 6 «Виведення результатів обробки – класу чистоти деталі». Виведення сформованого результату обробки, в формі що зрозуміла людині і відповідає формату описаному ДСТУ 11141-84 «Р Х/Х», де Х/Х це позначення класів чистоти за ознаками царапин і точок, у вікно «Результати обробки».

- Блок 7 «Збереження обробленого зображення». Можливість зберегти оброблене зображення з виділеними дефектами у файл .png або .jpg для

систематизації результатів.

Рис. 3.1 Узагальнена блок-схема програмного забезпечення

3.3. Розробка головного вікна програми та опис його функціоналу

Головне вікно програми має забезпечити реалізацію всіх функцій зображених на схемі на рис. 3.1. і надати користувачу можливість взаємодіяти з зображенням.

Перелік функцій головного меню і їх опис:

- load_image(self) - функція завантаження зображення для обробки, відкриває системне діалогове вікно провідника з можливістю вибору файлу зображення для завантаження, після вибору зображення відображає його в зоні «Вхідне зображення»

програми (рис. 3.2, 3.3. позначення 6). Спрацьовує при натисканні кнопки «Завантажити фото» (рис. 3.2, 3.3. позначення 2).

- open_processing_settings_menu(self) – функція відкриття вікна «Параметри обробки». Відкриває вікно з параметрами обробки зображення. Спрацьовує при натисканні кнопки «Параметри обробки» (рис. 3.2, 3.3. позначення 3).

- open_defects_table_window(self) – функція відкриття вікна «База виявлених дефектів на лінзі». Відкриває вікно з таблицею з переліком виявлених в процесі обробки дефектів. Спрацьовує при натисканні кнопки «Переглянути дефекти» (рис. 3.2, 3.3. позначення 4).

- process_image_trigger(self) – функція обробки зображення. По результату її роботи отримуємо список знайдених на зображенні контурів. Спрацьовує при натисканні зеленої кнопки «ОБРОБИТИ» (рис. 3.2, 3.3. позначення 7).

- redraw_contours_canvas(self) – функція перерисовує поле «Оброблене зображення» у відповідності з поточним результатом обробки і поточними налаштуваннями відображення контурів дефектів. Спрацьовує після закінчення обробки зображення, при зміні налаштувань відображення дефектів (рис. 3.2, 3.3. позначення 9).

- update_params_pane(self) – функція оновлення відображення поточних параметрів обробки в полі «параметри обробки», викликається щоразу при збереженні параметрів обробки.

- update_results_zone(self) – функція оновлення поля «Результати обробки». Оновлює поле «Результати обробки» вписавши туди назву оброблюваного зображення кількість і тип знайдених дефектів і значення визначеного класу чистоти. Спрацьовує по закінченню обробки зображення.

- save_img(self) – функція збереження зображення з поля «Оброблене зображення». Викликає системне вікно провідника з можливістю вибрати місце і назву для збереження зображення з поля «Оброблене зображення» (рис. 3.2, 3.3. позначення 8). Спрацьовує при натисканні кнопки «Зберегти оброблене зображення» (рис. 3.2, 3.3. позначення 12).

- log_message(self) – функція додавання нового сповіщення в поле

«Сповіщення» (рис. 3.2, 3.3. позначення 13). В якості аргумента приймає текст і в момент виклику створює сповіщення з поточним часом і наданим текстом. Викликається впродовж роботи програми в різних функціях.

- clear_console_history(self) – функція очищає всі сповіщення(рис. 3.2, 3.3. позначення 15) в полі «Сповіщення» (рис. 3.2, 3.3. позначення 13). Спрацьовує при натисканні кнопки «Очистити історію» (рис. 3.2, 3.3. позначення 14).

Для побудови візуального інтерфейсу програми, основною частиною якого є головне вікно програми, використовувалась бібліотека Tkinter. Всі об’єкти Tkinter, тобто кнопки, місця для відображення зображення і текстові поля мають знаходитись всередині об’єкту що називається Frame. Для кожної зони програми є свій Frame в якому знаходяться його елементи.

Вікно тільки запущеної програми зображене на рис. 3.2., вікно програми після завантаження зображення, виконання його обробки і збереження результату зображене на рис. 3.3.

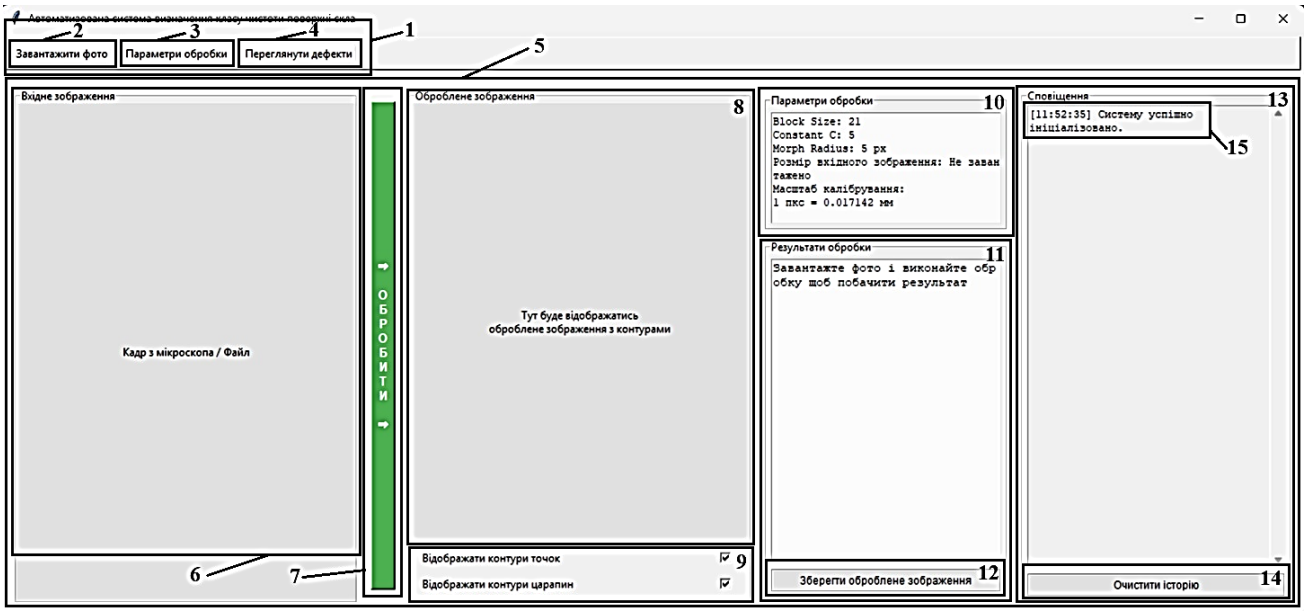


Рис. 3.2 Вікно програми після запуску

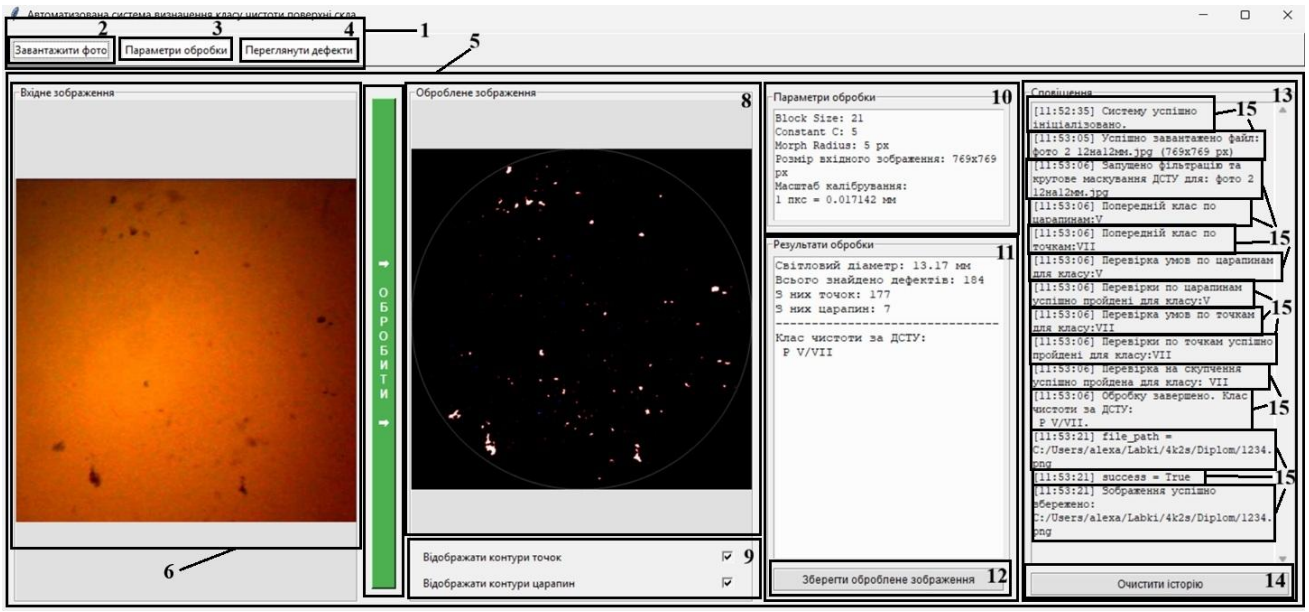


Рис. 3.3 Вікно програми після завантаження зображення, його обробки і збереження результату

Вікно програми має такі елементи (див. позначення на рис. 3.2 і рис. 3.3):

1. Панель інструментів з кнопками завантаження зображення для обробки, відкриття вікна параметрів обробки і відкриття вікна зі списком дефектів.

2. Кнопка «Завантажити фото». При натисканні викликається функція load_image(self), відбувається виклик системного вікна вибору файлу, в якому користувач обирає файл зображення для обробки.

3. Кнопка «Параметри обробки». При натисканні спрацьовує функція open_processing_settings_menu(self) що відкриває вікно «Параметри обробки» в якому користувач може налаштувати параметри обробки зображення і співвідношення мм/px.

4. Кнопка «Переглянути дефекти». При натисканні спрацьовує функція open_defects_table_window(self) що відкриває вікно «База виявлених дефектів на лінзі» в якому у вигляді таблиці вказані всі виявлені на зображення дефекти.

5. Основне робоче поле програми в якому розташовані поле з відображенням вхідного зображення, кнопка запуску процесу обробки зображення, поле з обробленим зображенням, налаштування відображення дефектів на зображення, поле з параметрами обробки зображення, поле з результатами обробки зображення, кнопка

збереження обробленого зображення в файл, поле сповіщень і кнопка очищення історії сповіщень.

6. Поле «Вхідне зображення». Тут буде відображатись зображення після його завантаження.

7. Кнопка «Обробити». При натисканні спрацьовує функція process_image_trigger(self) яка перевіряє наявність завантаженого зображення і запускає обробку по поточним налаштуванням.

8. Поле «Оброблене зображення». Після виконання обробки вхідного зображення спрацьовує функція redraw_contours_canvas(self) і в цьому полі буде відображатись бінаризоване (чорно-біле) зображення з виділеними синім кольором контурами царапин і червони кольором контурами точок.

9. Поле налаштувань відображення контурів. Тут можна вимкнути або ввімкнути відображення контурів дефектів в полі «Оброблене зображення».

10. Поле «Параметри обробки». Це текстове поле яке відображає поточні налаштування обробки зображення і оновлюється шляхом запуску функції update_params_pane(self) при збереження нових налаштувань у вікні «параметри обробки».

11. Поле «Результати обробки». Після обробки зображення викликаєтсья функція update_results_zone(self) що відображає призначений деталі клас чистоти разом з деякими проміжними параметрами.

12. Кнопка «Зберегти оброблене зображення». При натисканні викликаєтсья функція save_img(self) і відкривається системне вікно в якому можна зберегти поточне зображення із поля «Оброблене зображення».

13. Поле «Сповіщення». Сюди будуть виводитись сповіщення про роботу програми, такі як: запуск програми, успішне або неуспішне завантаження зображення, збереження налаштувань, обробка зображення, успішне чи неуспішне збереження зображення.

14. Кнопка «Очистити історію» викликає функцію clear_console_history(self) що очищує всі записи з поля «Сповіщення».

15. Різні варіанти сповіщень.

3.4. Розробка вікна параметрів обробки

Вікно «Параметри обробки» відкривається при виконанні функції open_processing_settings_menu(self) яка викликається при натисканні кнопки «Параметри обробки» (рис. 3.2, 3.3. позначення 3) на панелі інструментів в головному вікні програми.

Призначення вікна «Параметри обробки» тому щоб надати користувачу можливість налаштовувати параметри обробки зображення і співвідношення мм/px пере запуском обробки зображення для підготовки системи перед новою партією виробів.

Всього у вікні «Параметри обробки» можна налаштувати 4 параметри:

- Розмір блоку. Значення за замовчуванням 21, можливі значення 3-99. Використовується для функції адаптивної бінаризації зображення cv2.adaptiveThreshold().[33]

- Константа зсуву. Значення за замовчуванням 5, можливі значення 0-50. Використовується для функції адаптивної бінаризації зображення cv2.adaptiveThreshold().[33]

- Радіус об’єднання фрагментів. Значення за замовчуванням 5, можливі значення 1-30. Використовується для створення ядра для роботи функції морфологічного закриття (ерозія дилатованого зображення).

- Калібрування оптичного масштабу. Критичний для визначення класу чистоти параметр, використовується для перерахунку розмірів дефектів з пікселів в міліметри. Має 2 активних поля для вводу чисел – мм і px, програма сама обчислює їх співвідношення.

Параметри розміру блоку, константи зсуву і радіусу об’єднання фрагментів мають повзунок і поле для ручного вводу значення.

Вікно «параметри обробки» використовує такі функції:

- recalculate_scale(*args). Функція перерахунку співвідношення міліметр/піксель. Блок-схема функції на рис. 3.4. Функція отримує поточні значення

з полів вводу мм і px в вікні «Параметри обробки» за умови що вони існують (схема на рис. 3.4, блоки 3, 4, 5, 6, 7, 8), перевіряє їх на адекватність значення (схема на рис. 3.4, блок 9), за умови що значення більші нуля, обчислює співвідношення міліметр/піксель (схема на рис. 3.4, блок 11), оновлює його відображення в полі «Параметри обробки» в головному вікні програми і повертає обчислене значення (схема на рис. 3.4, блок 14) для подальшого використання функцією обчислення розміру дефектів.

- reset_defaults() - функція скидання налаштувань в вікні «Параметри обробки» до значень за замовчуванням. Спрацьовує при натисканні кнопки «Скинути за замовчуванням» (рис. 3.5. позначення 7) у вікні «Параметри обробки», і скидає значення розміру блоку, константи зсуву і радіусу об'єднання фрагментів до значень за замовчуванням, за умови що навпроти цих параметрів не стоять галочки (рис. 3.5. позначення 4), якщо стоїть галочка, то функція reset_defaults() не скидає значення цього параметру.

- save_variables_state() – функція збереження поточних значень параметрів обробки. Спрацьовує при натисканні кнопки «Зберегти параметри обробки» (рис. 3.5. позначення 8) у вікні «Параметри обробки», зберігає значення що записані в полях вікна «Параметри обробки» для подальшого їх використання в процесі обробки зображення. Якщо просто змінити значення параметра повзунком, або ввести нове значення вручну і не викликати функцію save_variables_state() не натиснувши кнопку «Зберегти параметри обробки», значення не будуть застосовані і не змінять результату обробки.

- process_image_trigger(self) – функція обробки зображення. По результату її роботи отримуємо список знайдених на зображенні контурів. Спрацьовує при натисканні зеленої кнопки «ОБРОБИТИ» (рис. 3.2, 3.3. позначення 7) в головному вікні програми», і при натисканні кнопки «Обробити» (рис. 3.5. позначення 9) в вікні «Параметри обробки».

- .destroy() – вбудований метод Tkinter що видаляє вікно по відношенню до якого Викликається. Викликається по відношенню до вікна «Параметри обробки» при натисканні кнопки «Закрити» (рис. 3.5. позначення 10).[34]

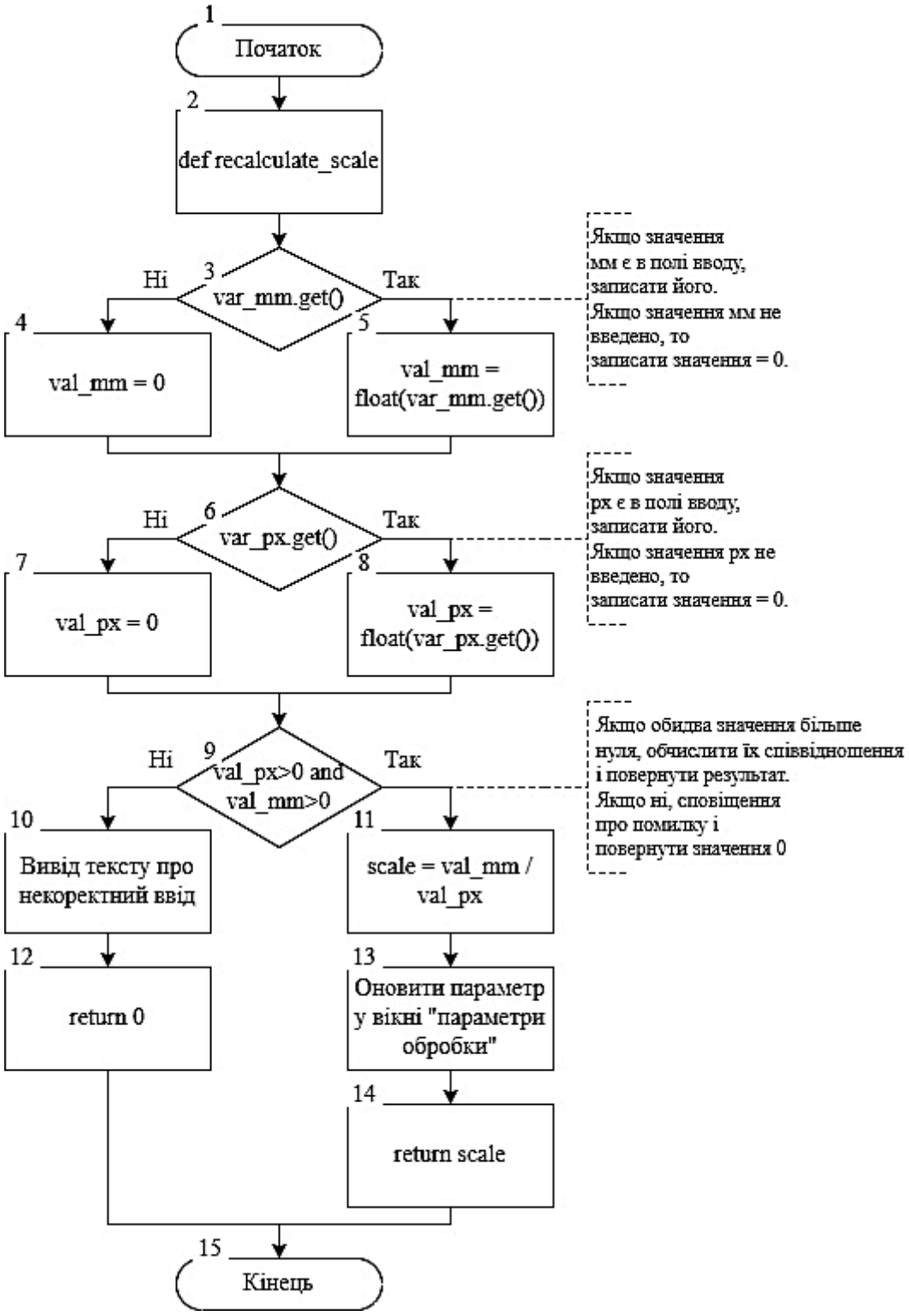


Рис. 3.4 Блок-схема алгоритму перерахунку співвідношення міліметр/піксель recalculate_scale()

Вікно «Параметри обробки» зображено на рис. 3.5.

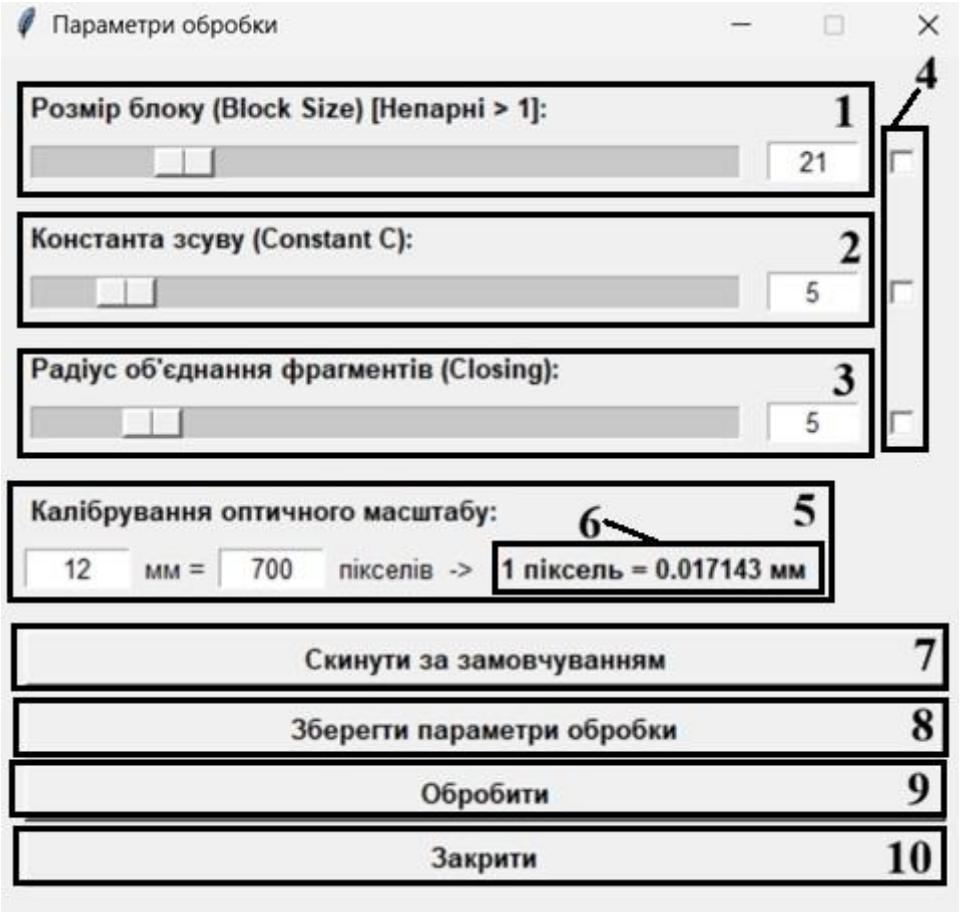


Рис. 3.5 Вікно «Параметри обробки»

Вікно «Параметри обробки» має такі елементи (див. позначення на рис. 3.5):

1. Поле регулювання параметру «Розмір блоку». Складається з назви параметру "Розмір блоку (Block Size) [Непарні > 1]:", повзунка для регулювання параметру і текстового поля з можливістю його зміни для регулювання параметру.

2. Поле регулювання параметру «Константа зсуву». Складається з назви параметру "Константа зсуву (Constant C):", повзунка для регулювання параметру і текстового поля з можливістю його зміни для регулювання параметру.

3. Поле регулювання параметру «Радіус об’єднання фрагментів». Складається з назви параметру "Радіус об'єднання фрагментів (Closing):", повзунка для регулювання параметру і текстового поля з можливістю його зміни для регулювання параметру.

4. З "чекбокси" навпроти кожного з 3х попередніх параметрів. Потрібні для блокування скидання параметру до значення за замовчуванням.

5. Поле регулювання параметру оптичного масштабу. Складається з назви, 2х

полів для вводу числових значень міліметрів і пікселів і графи що позначена цифрою 6.

6. Графа що відображає і постійно автоматично оновлює обчислене співвідношення міліметри/пікселі за рахунок виклику функції recalculate_scale(*args).

7. Кнопка «Скинути за замовчуванням». При натисканні викликається функція reset_defaults(), і скидає параметри «Розмір блоку», «Константа зсуву» і «Радіус об’єднання фрагментів» до параметрів за замовчуванням.

8. Кнопка «Зберегти параметри обробки». При натисканні викликається функція save_variables_state(), і поточно введені в графи параметри зберігаються для подальшого їх використовування під час обробки зображення.

9. Кнопка «Обробити». При натисканні викликається функція - process_image_trigger(self), і запускається обробка зображення.

10. Кнопка «Закрити». При натисканні викликається вбудований в Tkinter метод .destroy(), що знищує вікно «Параметри обробки».[34]

3.5. Розробка вікна «База виявлених дефектів деталі»

Вікно «База виявлених дефектів деталі» відкривається при виконанні функції open_defects_table_window(self) яка викликається при натисканні кнопки «Переглянути дефекти» (рис. 3.2, 3.3. позначення 4) на панелі інструментів в головному вікні програми.

Призначення вікна – відображення списку всіх виявлених на зображенні дефектів у вигляді таблиці з параметрами кожного дефекту для зручності користувача.

Після кожного запуску функції обробки зображення програма створює структуру даних в яку записані всі дефекти і деякі їх характеристики для подальшого визначення класу чистоти, таблиця у вікні «База виявлених дефектів деталі» і відображаю всі ці записи і деякі збережені параметри. Таблиця відображає такі параметри дефектів:

- ID дефекту – унікальний ідентифікаційний номер що призначається кожному

дефекту.

- Тип дефекту – значення цього поля може бути «Точка» або «Царапина», ця класифікація проводиться в функції обробки зображення при формуванні структури даних з інформацією про всі дефекти. Точка від царапини розрізняється за описаною в ДСТУ11141-84 ознакою співвідношення ширини дефекту до його довжини має бути більше ніж 3:1.[7]

- Довжина (px) – довжина дефекту в пікселях.

- Довжина (мм) – довжина дефекту в міліметрах. Обсичлюється зі значення Довжина (px) за допомогою співвідношення міліметр/піксель.

- Ширина (px) – ширина дефекту в пікселях.

- Ширина(мм) – довжина дефекту в міліметрах. Обсичлюється зі значення Ширина (px) за допомогою співвідношення міліметр/піксель.

- Співвідношення – співвідношення ширини дефекту до його довжини обчислене з значень довжина(px) і ширина(px).

Вікно «База виявлених дефектів деталі» використовує такі функції:

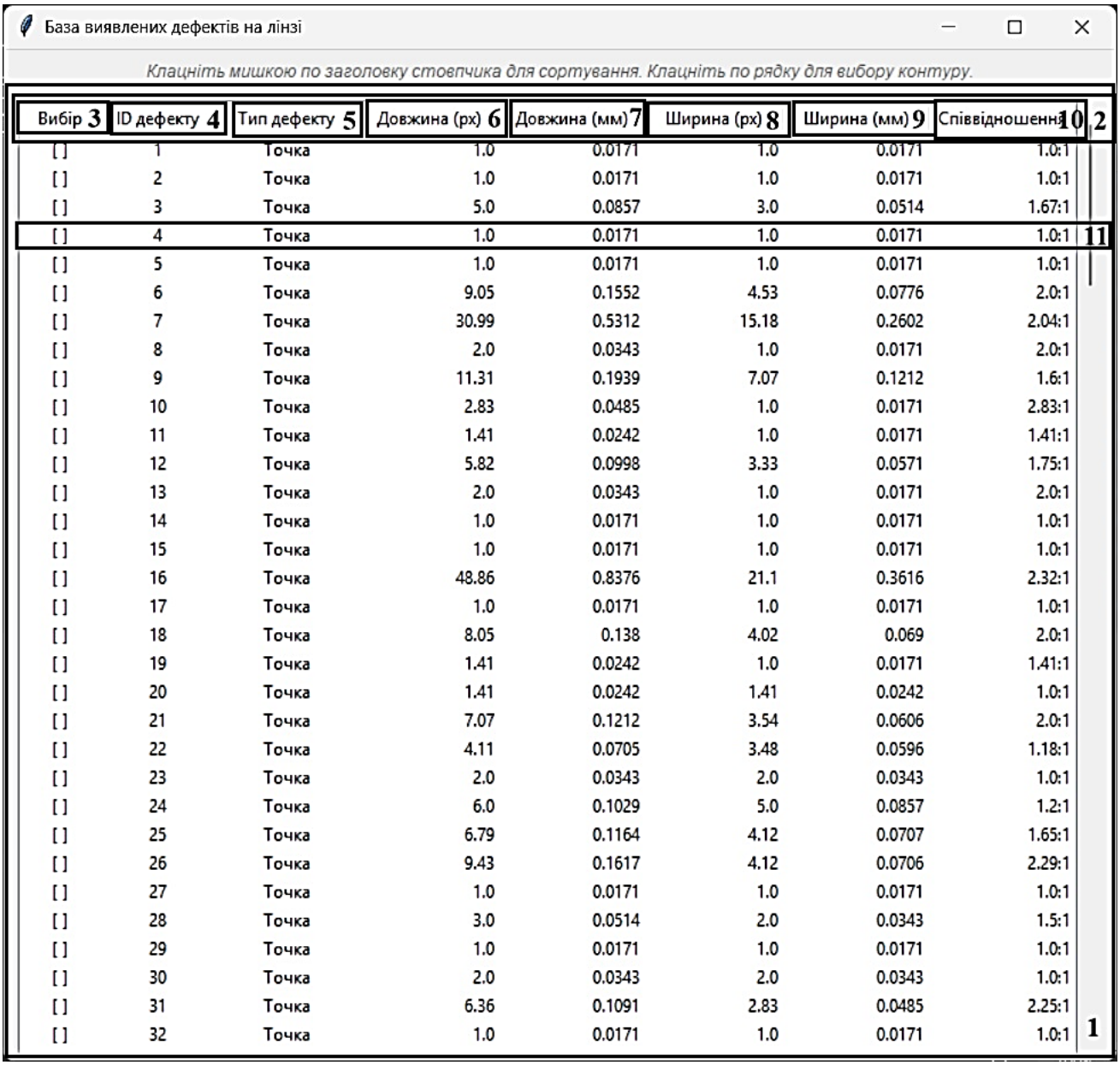
- sort_table_column(self, col_name, reverse_sort) – функція сортування таблиці за ознакою. Приймає в якості аргументу назву стовпчика і напрямок сортування. Спрацьовує при натисканні на будь-який заголовок таблиці (рис. 3.6. позначення 3-10), змінює порядок відображення дефектів в таблиці на відсортований у порядку зростання або спадання параметру на який було натиснуто.

- on_table_row_click(self) – функція реалізує підсвічування вибраних дефектів жовтим кольором в полі «Оброблене зображення». Реалізовано через спостереження за кліками користувача, якщо був клік на рядок то статус параметру «Вибір» змінюється на протилежний і викликається функція redraw_contours_canvas(self), яка зважає на статус поля «Вибір» в параметрах дефекту і відмальовує його відповідним кольором.

- refresh_table_data(self) – при виклику виконує оновлення вікна «База виявлених дефектів деталі», потрібно для динамічного оновлення таблиці при відкритому вікні дефектів після кожної обробки зображення. Викликається іншими

функціями програми.

Вікно «База виявлених дефектів» зображено на рис. 3.6.



Вибір	ID дефекту	Тип дефекту	Довжина (px)	Довжина (мм)	Ширина (px)	Ширина (мм)	Співвідношення
[ ]	1	Точка	1.0	0.0171	1.0	0.0171	1.0:1
[ ]	2	Точка	1.0	0.0171	1.0	0.0171	1.0:1
[ ]	3	Точка	5.0	0.0857	3.0	0.0514	1.67:1
[ ]	4	Точка	1.0	0.0171	1.0	0.0171	1.0:1
[ ]	5	Точка	1.0	0.0171	1.0	0.0171	1.0:1
[ ]	6	Точка	9.05	0.1552	4.53	0.0776	2.0:1
[ ]	7	Точка	30.99	0.5312	15.18	0.2602	2.04:1
[ ]	8	Точка	2.0	0.0343	1.0	0.0171	2.0:1
[ ]	9	Точка	11.31	0.1939	7.07	0.1212	1.6:1
[ ]	10	Точка	2.83	0.0485	1.0	0.0171	2.83:1
[ ]	11	Точка	1.41	0.0242	1.0	0.0171	1.41:1
[ ]	12	Точка	5.82	0.0998	3.33	0.0571	1.75:1
[ ]	13	Точка	2.0	0.0343	1.0	0.0171	2.0:1
[ ]	14	Точка	1.0	0.0171	1.0	0.0171	1.0:1
[ ]	15	Точка	1.0	0.0171	1.0	0.0171	1.0:1
[ ]	16	Точка	48.86	0.8376	21.1	0.3616	2.32:1
[ ]	17	Точка	1.0	0.0171	1.0	0.0171	1.0:1
[ ]	18	Точка	8.05	0.138	4.02	0.069	2.0:1
[ ]	19	Точка	1.41	0.0242	1.0	0.0171	1.41:1
[ ]	20	Точка	1.41	0.0242	1.41	0.0242	1.0:1
[ ]	21	Точка	7.07	0.1212	3.54	0.0606	2.0:1
[ ]	22	Точка	4.11	0.0705	3.48	0.0596	1.18:1
[ ]	23	Точка	2.0	0.0343	2.0	0.0343	1.0:1
[ ]	24	Точка	6.0	0.1029	5.0	0.0857	1.2:1
[ ]	25	Точка	6.79	0.1164	4.12	0.0707	1.65:1
[ ]	26	Точка	9.43	0.1617	4.12	0.0706	2.29:1
[ ]	27	Точка	1.0	0.0171	1.0	0.0171	1.0:1
[ ]	28	Точка	3.0	0.0514	2.0	0.0343	1.5:1
[ ]	29	Точка	1.0	0.0171	1.0	0.0171	1.0:1
[ ]	30	Точка	2.0	0.0343	2.0	0.0343	1.0:1
[ ]	31	Точка	6.36	0.1091	2.83	0.0485	2.25:1
[ ]	32	Точка	1.0	0.0171	1.0	0.0171	1.0:1

Рис. 3.6 Вікно програми «База виявлених дефектів»

Вікно має такі елементи(див. позначення на рис. 3.6):

1. Поле таблиці з її заголовками і записами про дефекти.

2. Поле з заголовками таблиці.

3. 3, 4, 5, 6, 7, 8, 9, 10 – заголовки окремих стовпчиків таблиці що відображають деякі параметри об’єкту дефекту. При натисканні на будь-який заголовок викликається функція sort_table_column() і вся таблиця сортується в порядку зростання або спадання параметру на заголовок якого було натиснуто.

4. 11. Рядок – запис про дефект.

3.6. Реалізація функції обробки зображення, виявлення дефектів та класифікації поверхні за ДСТУ11141-84

Функція обробки зображення і класифікації поверхні є основною частиною всього програмного модуля автоматизованої системи визначення класу чистоти поверхні скла. Саме ця функція виконує основну частину роботи – підрахунок дефектів.

Функція запускається при натисканні на кнопку «Обробити» в головному вікні програми, або на кнопку «Обробити» у вікні «Параметри обробки». Вона використовує методи OpenCV і NumPy для пошуку дефектів і математичних обчислень необхідних для надання класу.

Алгоритм пошуку дефектів і оцінки класу чистоти складається з взаємодії 2х функцій:

- 1ша функція process_image_trigger(self) на вхід отримує зображення і в проміжному етапі роботи видає структуру даних з усіма дефектами, потім викликає функцію evaluate_dstu_class(self, light_diameter_mm), передаючи туди структуру даних з дефектами і обчислене значення радіусу світлового діаметра, отримуючи назад класи чистоти деталі за ознаками царапин і точок. Після отримання результату аналізу, функція виводить результат в поле «Результати обробки» основного вікна програми.

- evaluate_dstu_class(self, light_diameter_mm) є 2ю частиною алгоритму аналізу зображення, вона в собі має базу вимог до кожного з класів чистоти. В ході роботи функція оцінює чи проходить отриманий набір дефектів вимоги для класів чистоти, починаючи з найвимогливішого класу I і рухаючись вниз по класам, зменьшуючи вимоги, до тих пір допоки не знайде клас всі вимоги якого виконуються набором дефектів. Потім функція повертає 2 значення класу чистоти – за ознакою царапин і за ознакою точок

Блок-схема алгоритму функції process_image_trigger(self) зображена на рис 3.7, 3.8, 3.9.

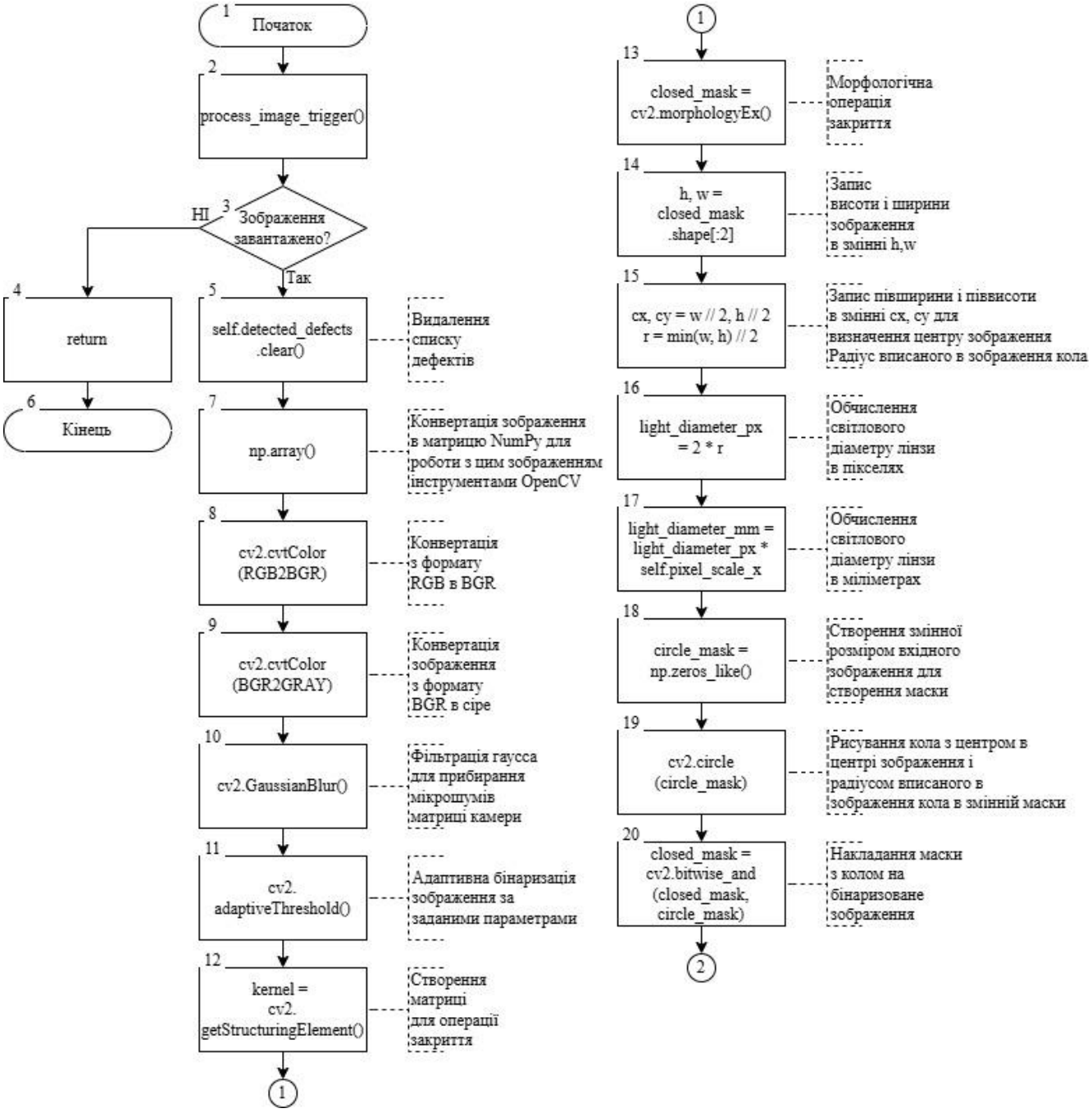


Рис. 3.7 Блок-схема алгоритму функції process_image_trigger(self) частина 1

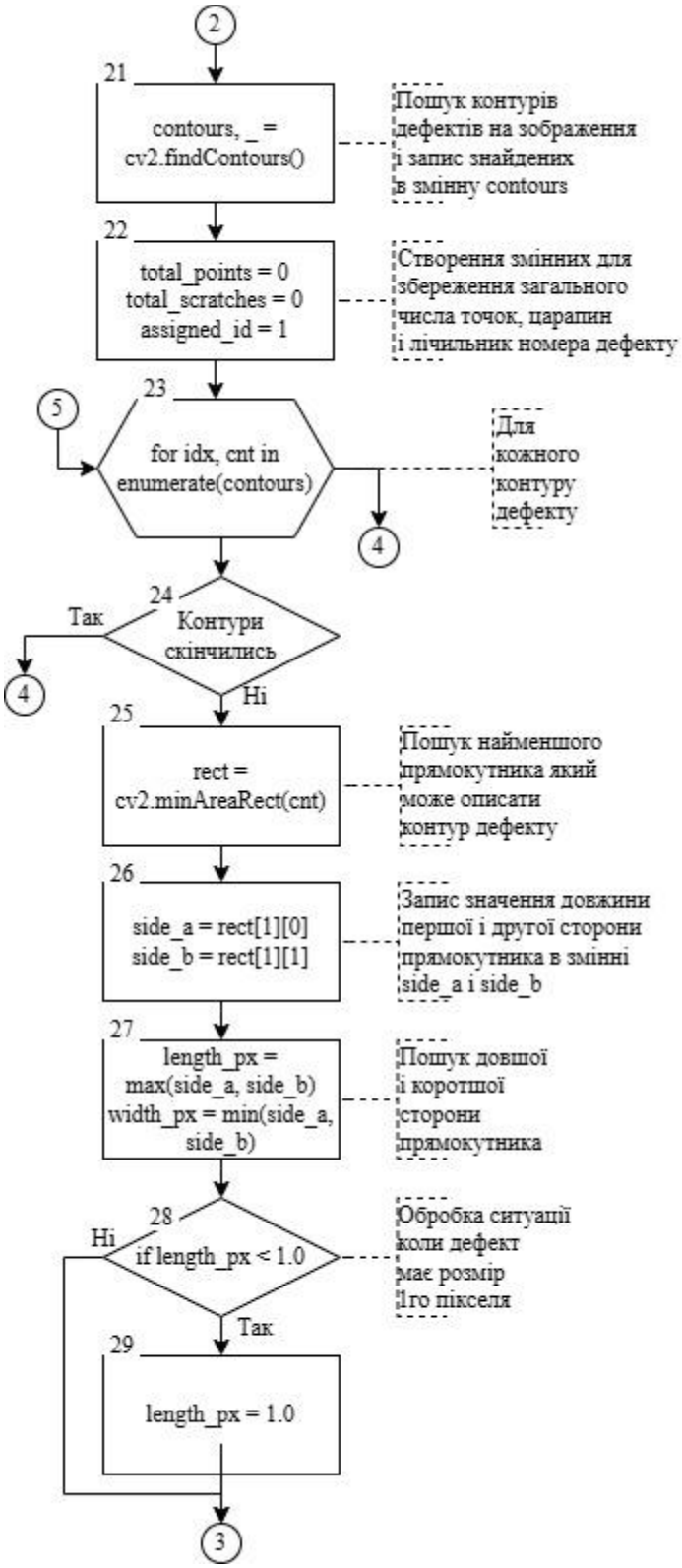


Рис. 3.8 Блок-схема алгоритму функції process_image_trigger(self) частина 2

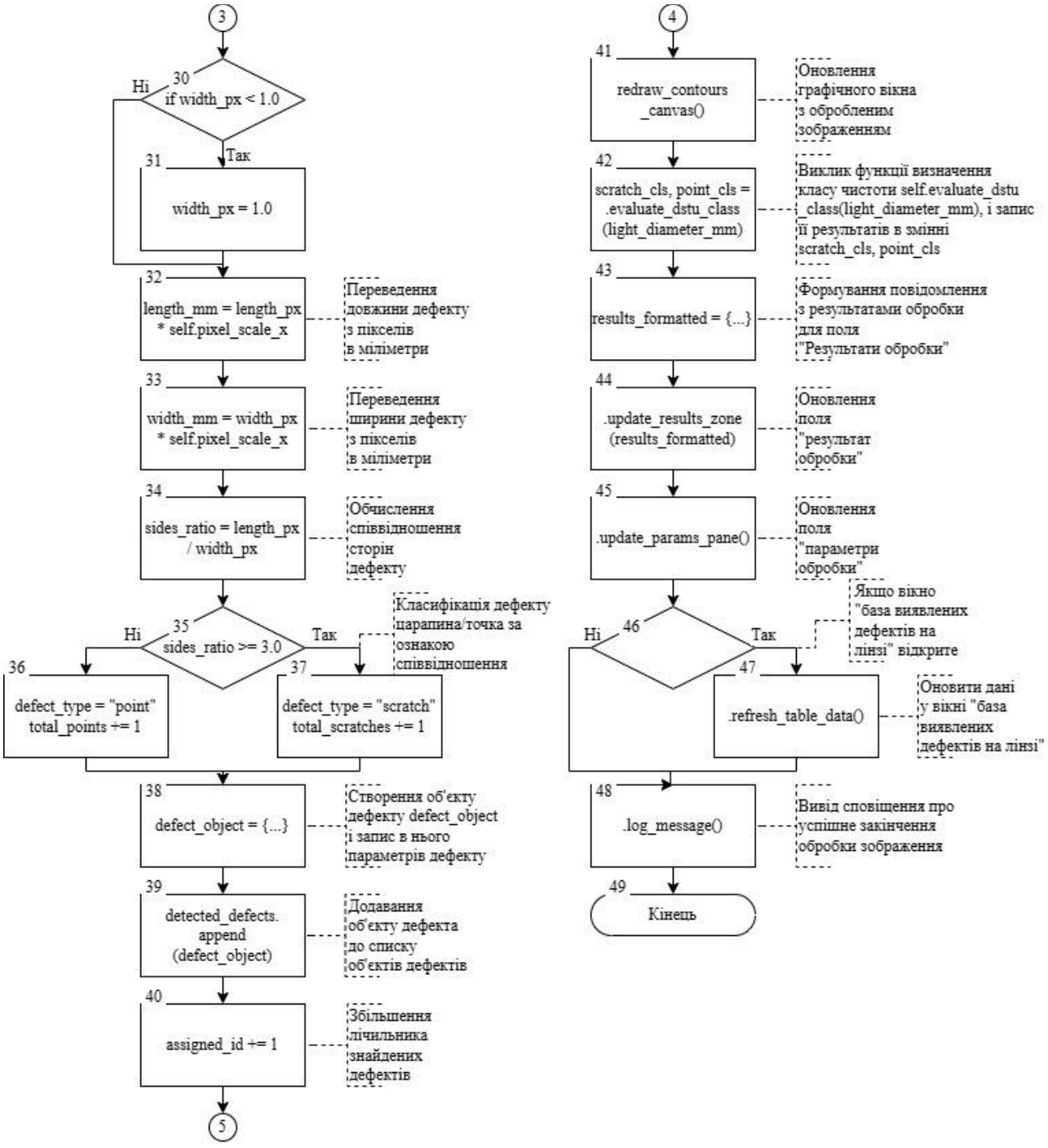


Рис. 3.9 Блок-схема алгоритму функції process_image_trigger(self) частина 3

Пояснення блоків блок-схеми алгоритму функції process_image_trigger(self) (див. позначення на рис 3.7, 3.8, 3.9) за їх номерами:

1. Початок виконання функції.
2. Назва функції.
3. Перевірка на те чи завантажений об’єкт зображення в програму.
4. Якщо зображення не було завантажено – процес обробки не починається.
5. Якщо зображення було завантажено – очищення списку виявлених дефектів.

6. Закінчення виконання функції.

7. Конвертація об’єкту зображення в прийнятний для обробки функціями бібліотеки OpenCV формат функцією np.array().[35]

8. Конвертація колірної схеми зображення з RGB в BGR. Це спричинено особливістю того як різні бібліотеки зберігають зображення.[36]

9. Конвертація колірної схеми зображення з кольорового в сіре. Потрібно для роботи функції адаптивної бінаризації.

10. Обробка зображення функцією фільтрації Гаусса cv2.GaussianBlur(), потрібно для видалення мікрошумів на зображенні спричинених матрицю камери.[37]

11. Виконання адаптивної бінаризації зображення функцією cv2.adaptiveThreshold() з значеннями з вікна «Параметри обробки» (рис. 3.5. позначення 1, 2) в якості аргументу функції. Ця дія є першим суттєвим кроком в алгоритмі пошуку дефектів, вона залишає на зображенні тільки білі і чорні пікселі, де білі пікселі це дефекти поверхні лінзи, а чорні це чистий фон.[33]

12. Створення ядра-матриці kernel = cv2. getStructuringElement() для операції морфологічного закриття. Об’єкт kernel буде в подальшому використаний як аргумент в функції морфологічного закриття, що потрібна для прибирання шумів і об’єднання малих царапин.[38]

13. Виконання операції морфологічного закриття closed_mask = cv2.morphologyEx() і запис її результату в змінну closed_mask. Морфологічне закриття – операція що є результатом 2х обробок зображення – дилатації (розширення) і ерозії(звуження). Практично візуально збільшує всі дефекти і потім пропорційно зменшує їх. Якщо у процесі збільшення 2 дефекти об’єднались в 1, то вони залишаться одним цілим при зменшенні. Це потрібно для об’єднання царапин і більш коректного розпізнавання великих дефектів. [38]

14. Отримуємо розміри зображення і записуємо їх в змінні h, w =closed_mask.shape[:2]. Потрібно щоб потім вписати в зображення коло і оцінювати дефекти в ньому.[39]

15. Обчислення піввисоти і півширини зображення щоб знайти його центр, що стане центром вписаного кола cx, cy = w // 2, h // 2. Обчислення радіусу кола що можна

буде вписати шляхом знаходження половини найменшої сторони зображення r = min(w, h) // 2.

16. Обчислення світлового діаметру лінзи в пікселях за формулою light_diameter_px = 2 * r. Ця величина буде передаватись як аргумент в функцію визначення класу чистоти за скупченням дефектів.

17. Обчислення світлового діаметру лінзи в міліметрах, перевівши значення світлового діаметру лінзи в пікселях light_diameter_mm = light_diameter_px * self.pixel_scale_x за допомогою обчисленого в вікні «Параметри обробки» параметра «Калібрування оптичного масштабу» (рис. 3.5. позначення 6). Для функції оцінки класу за ДСТУ потрібне саме значення в міліметрах, бо стандарт нормує дефекти по значенню в міліметрах.

18. Створення тимчасової змінної circle_mask = np.zeros_like() з усіма чорними пікселями. Потрібно для накладання на оригінальне зображення маски кола для імітації фото лінзи.[40]

19. Рисування кола з визначеними у блоці №15 координатами центру cx, cy і радіусом r, у змінній cv2.circle(circle_mask).

20. Накладання маски з колом на оригінальне зображення що пройшло адаптивну бінаризацію і морфологічне закриття closed_mask = cv2.bitwise_and (closed_mask, circle_mask). Тепер всі дефекти що не в колі не будуть рахуватись при аналізі, що дозволяє оцінювати саме круглі лінзи.[42]

21. Пошук контурів дефектів на зображенні і запис результатів пошуку в змінну contours, _ = cv2.findContours(). Тепер в змінній contours міститься масив контурів, кожен з яких є масивом координат всіх точок що утворюють межу контуру. Маючи ці інформацію можна далі досліджувати контури, знаходити розміри дефектів в них, рахувати ці дефекти і тд.[43]

22. Створення змінних що є лічильниками кількості знайдених точок, кількості знайдених царапин і лічильник що буде використовуватись для призначення дефектам унікального ідентифікаційного номеру.

23. Цикл що буде проходитись по кожному знайденому контуру, отримувати з нього інформацію, аналізувати, виконувати математичні операції і створювати об’єкт

дефекту для кожного контуру.

24. Перевірка на те чи є ще контури в списку, якщо контури скінчились то вихід з циклу в блоці 23.

25. Пошук найменшого прямокутника в який можна вписати контур за допомогою функції rect = cv2.minAreaRect(cnt). Це потрібно для обчислення довжини і ширини дефекту, що потрібно для його класифікації царапина/точка і для обчислення класу чистоти поверхні.[44]

26. Після знаходження найменшого описаного навколо контуру дефекту прямокутника, довжини його сторін записуються в змінні side_a = rect[1][0] side_b = rect[1][1]. Ці довжини потім і стануть довжиною і шириною дефекту.

27. Визначення довжини і ширини дефекту шляхом знаходження довшої і коротшої сторін описаного прямокутника length_px = max(side_a, side_b), width_px = min(side_a, side_b).

28. Обробка ситуації коли розмір дефекту дорівнює одному пікселю і функція cv2.minAreaRect() обчислює його розмір як 0. Дефект з розміром 0 – неіснуючий дефект, а такого не може бути, тож в блоці 29 йому призначається значення довжини 1.[44]

29. Призначення дефекту довжини 1 у випадках коли програма некоректно обчислила його довжину.

30. Обробка ситуації некоректного визначення ширини функцією cv2.minAreaRect().[44]

31. Призначення дефекту ширини 1 у випадках неправильного її попереднього визначення.

32. Перерахунок довжини дефекту з пікселів в міліметри з використанням обчисленого в вікні «Параметри обробки» параметра «Калібрування оптичного масштабу» (рис. 3.5. позначення 6) і запис результату в змінну length_mm = length_px * self.pixel_scale_x.

33. Перерахунок ширини дефекту з пікселів в міліметри width_mm = width_px * self.pixel_scale_x.

34. Обчислення співвідношення сторін дефекту і запис результату в змінну

sides_ratio = length_px / width_px.

35. Оператор if для класифікації дефекту царапина/точка в залежності від обчисленого в блоці 34 параметру відношення сторін.

36. Якщо співвідношення сторін менше за 3,0 – дефект класифікується як точка.

37. Якщо співвідношення сторін більше або дорівнює 3,0 – дефект класифікується як царапина. [7][11]

38. Створення об’єкту дефекту. Об’єкт дефекту містить в собі знайдений контур дефекту і деяку іншу інформацію, а саме: унікальний ідентифікаційний номер, тип царапина/точка, довжина в пікселях, довжина в міліметрах, ширина в пікселях, ширина в міліметрах, співвідношення сторін, площа дефекту що обчислюється за допомогою функції cv2.contourArea(), також булеве значення того чи вибраний дефект в списку дефектів в вікні «База виявлених дефектів на лінзі» і індекс контуру дефекту в списку знайдених контурів.[45]

39. Додавання сформованого в 38 блоці об’єкта дефекту в список дефектів.

40. Збільшення значення лічильника унікальних ідентифікаційних номерів для дефектів.

41. Виклик функції redraw_contours _canvas() що перемальовує контури на зображенні в головному вікні в зоні «Оброблене зображення» (рис. 3.2, 3.3. позначення 8).

42. Виклик функції визначення класів чистоти за дефектами і запис її результатів в змінні scratch_cls, point_cls = self.evaluate_dstu_class(light_diameter_mm). Функція приймає на вхід світловий діаметр лінзи що на фото і відповідно що нього в вимог в ДСТУ обробляє список знайдених на зображенні дефектів і повертає ці 2 значення.

43. Формування тексту з результатами аналізу дефектів для його виводу в поле «Результати обробки» головного вікна (рис. 3.2, 3.3. позначення 11). Текст має такі складові: світловий діаметр деталі, загальна кількість знайдених дефектів, кількість точок з них, кількість царапин з них і класи чистоти по царапинам і точкам за ДСТУ.

44. Запуск функції self.update_results_zone(results_formatted) що оновлює поле результатів обробки з сформованим в блоці 43 текстом з результатами обробки.

45. Запуск функції self.update_params_pane() що оновлює поле параметрів обробки.

46. Перевірка на те чи відкрите вікно «База виявлених дефектів на лінзі» з таблицею виявлених дефектів.

47. Якщо вікно «База виявлених дефектів на лінзі» відкрите то оновити в ньому дані відповідно до списку знайдених дефектів

48. Вивід в поле «Сповіщення» сповіщення про успішну обробку зображення за допомогою функції self.log_message()

49. Кінець функції обробки зображення.

Блок-схема алгоритму функції evaluate_dstu_class(self, light_diameter_mm) зображена на рис. 3.11, 3.12, 3.13, 3.14, 3.15, 3.16.

Функція evaluate_dstu_class(self, light_diameter_mm) приймає як аргумент обчислений в функції process_image_trigger(self) світловий діаметр лінзи і на основі нього і списку дефектів оцінює деталь на основі вимог що описані в таблицях 2, 3, 4 ДСТУ11141-84.

Всі вимоги з таблиць №2, 3, 4 ДСТУ 11141-84 (таблиця 1.1, 1.2, 1.3) в програмі зберігаються в змінній dstu_classes_requirements={} що являє собою словник словників словників, тобто тричі вкладений словник. Словники в Python є об’єктами класу dict і працюють за принципом ключ – значення, співставляючи їх. Щоб отримати значення потрібно звернутись до змінної словника вказавши ключ. Ця змінна є основою алгоритму визначення класу чистоти бо описує правила за якими визначається клас. Опис її рівнів вкладеності:

- У Верхньому рівні вкладеності словника dstu_classes_requirements={} знаходяться 12 об’єктів рядків str яка містить, що відображають 11 класів чистоти з стандарту з I по IXa, і ще один об’єкт 'NO_CLASS' що потрібен для призначення браку деталям. Значеннями що відповідають ключам цих словників є інші словники – 2й рівень вкладеності.

- В кожному з 12ти словників що відображають клас чистоти є 3 значення - об’єкти які називаються ‘table1’, ‘table2’ і ‘table3’, що теж є словниками. Кожен з

об’єктів ‘table1’, ‘table2’ і ‘table3’ містить в собі інформацію про допуски для поточного класу чистоти.

- 3й рівень вкладеності – ключем словника є назва конкретного фізичного допуску, наприклад максимальний розмір царапини, або число точок.

Щоб отримати, наприклад, обмеження максимальному допустимому розміру точки з таблиці 2 ДСТУ для класу VIIIa, потрібно звернутись до змінної dstu_classes_requirements з таким синтаксисом:

dstu_classes_requirements['VIIIa']['table2']['max_point_diameter'], де 'VIIIa' – назва класу, 'table2' – таблиця значення з якої потрібно отримати, 'max_point_diameter' – назва значення яке потрібно отримати.

Такі звернення до змінної dstu_classes_requirements використовуються в функції evaluate_dstu_class() при кожній потребі перевірки відповідності дефекту вимогам класу.

Частина оголошення змінної в коді зображено на рис.3.10.

```
dstu_classes_requirements = {
    'I': {'table2': {'max_scratch_width': 0.004, 'scratch_len_d_coef': 2.0, 'max_point_diameter': 0.02, 'point_count_d_coef': 0.5, 'cluster_d': 1.0, 'cluster_max_area': 0.004},
          'table3': None,  #для класів I і II Таблиця 3 не застосовується
          'table4': {'ignore_scratch_width': 0.001, 'ignore_point_diameter': 0.002}
    },
    'II': {'table2': {'max_scratch_width': 0.006, 'scratch_len_d_coef': 2.0, 'max_point_diameter': 0.05, 'point_count_d_coef': 0.5, 'cluster_d': 1.2, 'cluster_max_area': 0.006
           'table3': None,  #для класів I і II Таблиця 3 не застосовується
           'table4': {'ignore_scratch_width': 0.001, 'ignore_point_diameter': 0.002}
    },
    'III': {'table2': {'max_scratch_width': 0.01, 'scratch_len_d_coef': 2.0, 'max_point_diameter': 0.1, 'point_count_d_coef': 0.5, 'cluster_d': 2.0, 'cluster_max_area': 0.02},
            'table3': {'scratch_range': (0.006, 0.01), 'scratch_sum_length_d_coef': 1.0, 'point_range': (0.05, 0.1), 'point_count_d_coef': 0.2},
            'table4': {'ignore_scratch_width': 0.002, 'ignore_point_diameter': 0.004}
    },
    'IV': {'table2': {'max_scratch_width': 0.02, 'scratch_len_d_coef': 2.0, 'max_point_diameter': 0.3, 'point_count_d_coef': 0.5, 'cluster_d': 5.0, 'cluster_max_area': 0.1},
           'table3': {'scratch_range': (0.01, 0.02), 'scratch_sum_length_d_coef': 1.0, 'point_range': (0.1, 0.3), 'point_count_d_coef': 0.2},
           'table4': {'ignore_scratch_width': 0.004, 'ignore_point_diameter': 0.01}
    },
    'V': {'table2': {'max_scratch_width': 0.04, 'scratch_len_d_coef': 2.0, 'max_point_diameter': 0.5, 'point_count_d_coef': 0.5, 'cluster_d': 10.0, 'cluster_max_area': 0.4},
          'table3': {'scratch_range': (0.02, 0.04), 'scratch_sum_length_d_coef': 1.0, 'point_range': (0.3, 0.5), 'point_count_d_coef': 0.2},
          'table4': {'ignore_scratch_width': 0.006, 'ignore_point_diameter': 0.02}
    },
    'VI': {'table2': {'max_scratch_width': 0.06, 'scratch_len_d_coef': 2.0, 'max_point_diameter': 0.7, 'point_count_d_coef': 0.5, 'cluster_d':25.0, 'cluster_max_area': 3.0},
           'table3': {'scratch_range': (0.04, 0.06), 'scratch_sum_length_d_coef': 1.0, 'point_range': (0.5, 0.7), 'point_count_d_coef': 0.2},
           'table4': {'ignore_scratch_width': 0.008, 'ignore_point_diameter': 0.04}
    },
    'VII': {'table2': {'max_scratch_width': 0.1, 'scratch_len_d_coef': 2.0, 'max_point_diameter': 1.0, 'point_count_d_coef': 0.5, 'cluster_d': 50.0, 'cluster_max_area': 10.0},
            'table3': {'scratch_range': (0.06, 0.1), 'scratch_sum_length_d_coef': 1.0, 'point_range': (0.7, 1.0), 'point_count_d_coef': 0.2},
            'table4': {'ignore_scratch_width': 0.01, 'ignore_point_diameter': 0.1}
    },
    'VIII': {'table2': {'max_scratch_width': 0.2, 'scratch_len_d_coef': 2.0, 'max_point_diameter': 2.0, 'point_count_d_coef': 0.4, 'cluster_d': None, 'cluster_max_area': None},
             'table3': {'scratch_range': (0.1, 0.2), 'scratch_sum_length_d_coef': 1.0, 'point_range': (1.0, 2.0), 'point_count_d_coef': 0.2},
             'table4': {'ignore_scratch_width': 0.014, 'ignore_point_diameter': 0.14}
    }
```

Рис. 3.10 Частина оголошення змінної dstu_classes_requirements в коді

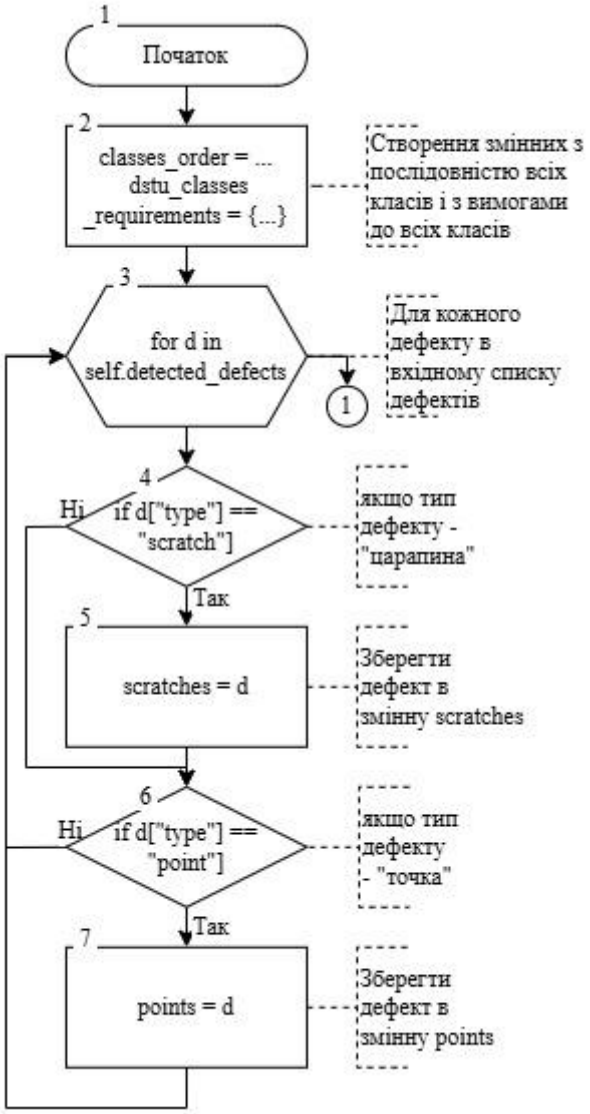


Рис. 3.11 Блок-схема алгоритму функції evaluate_dstu_class(self, light_diameter_mm) частина 1

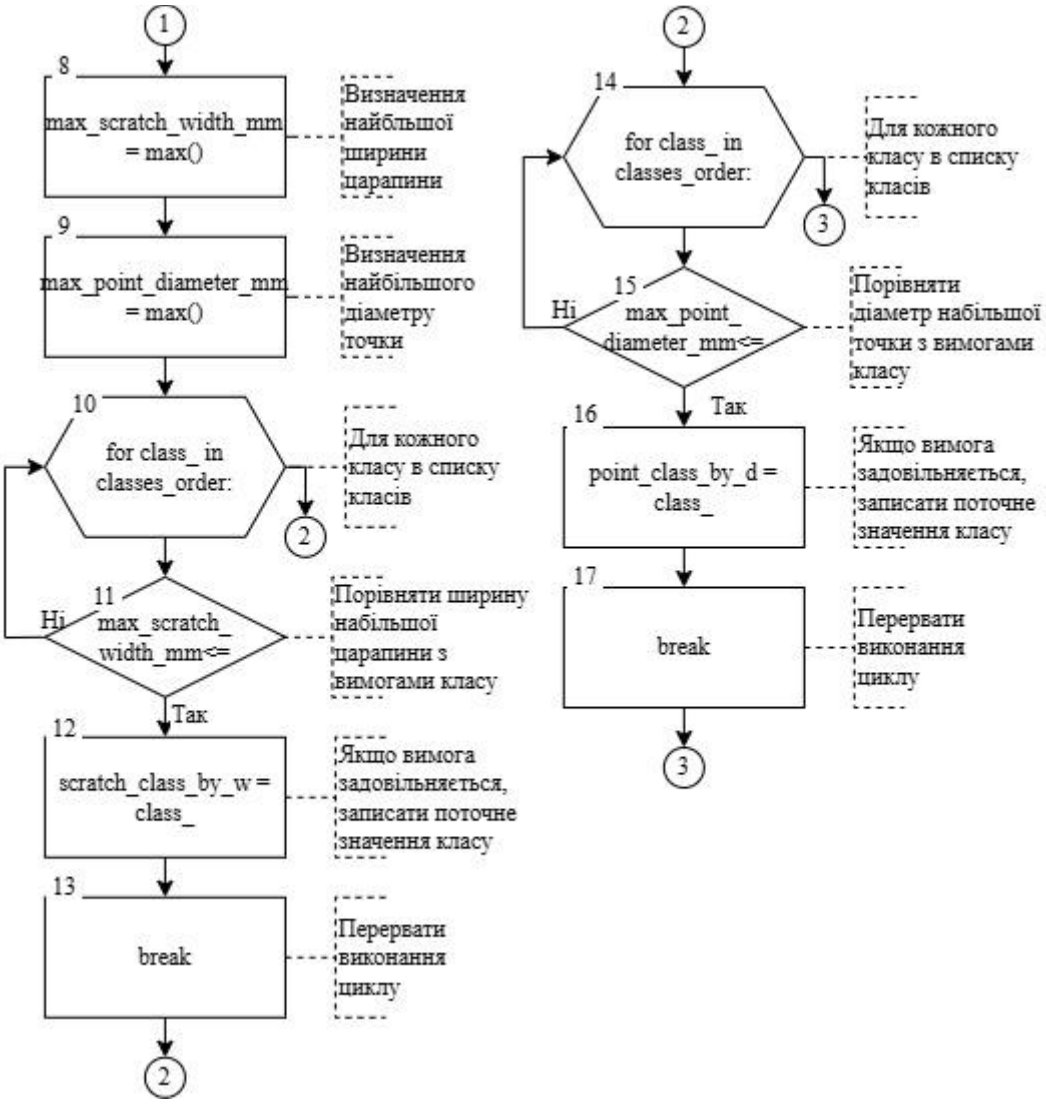


Рис. 3.12 Блок-схема алгоритму функції evaluate_dstu_class(self, light_diameter_mm) частина 2

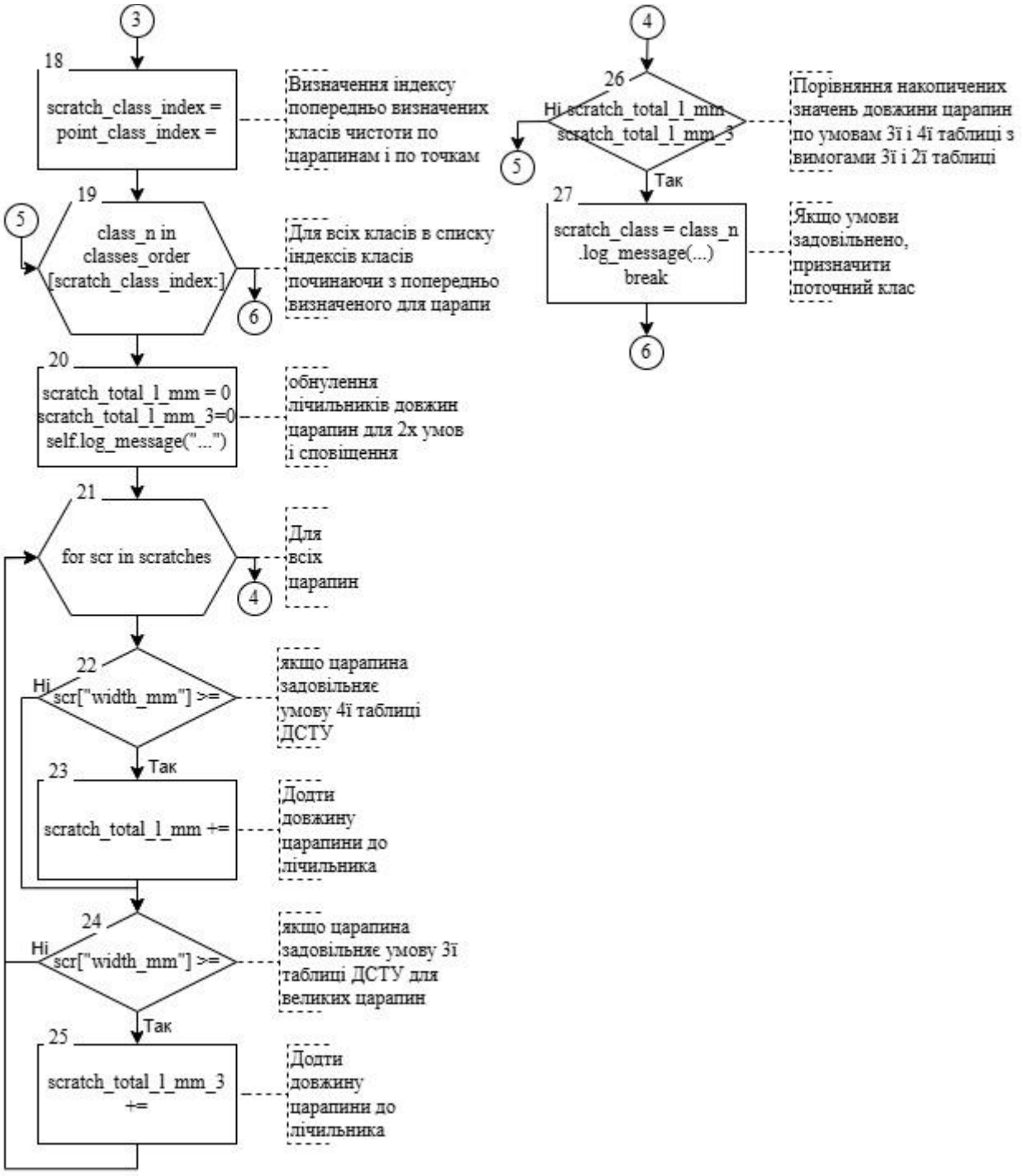


Рис. 3.13 Блок-схема алгоритму функції evaluate_dstu_class(self, light_diameter_mm) частина 3

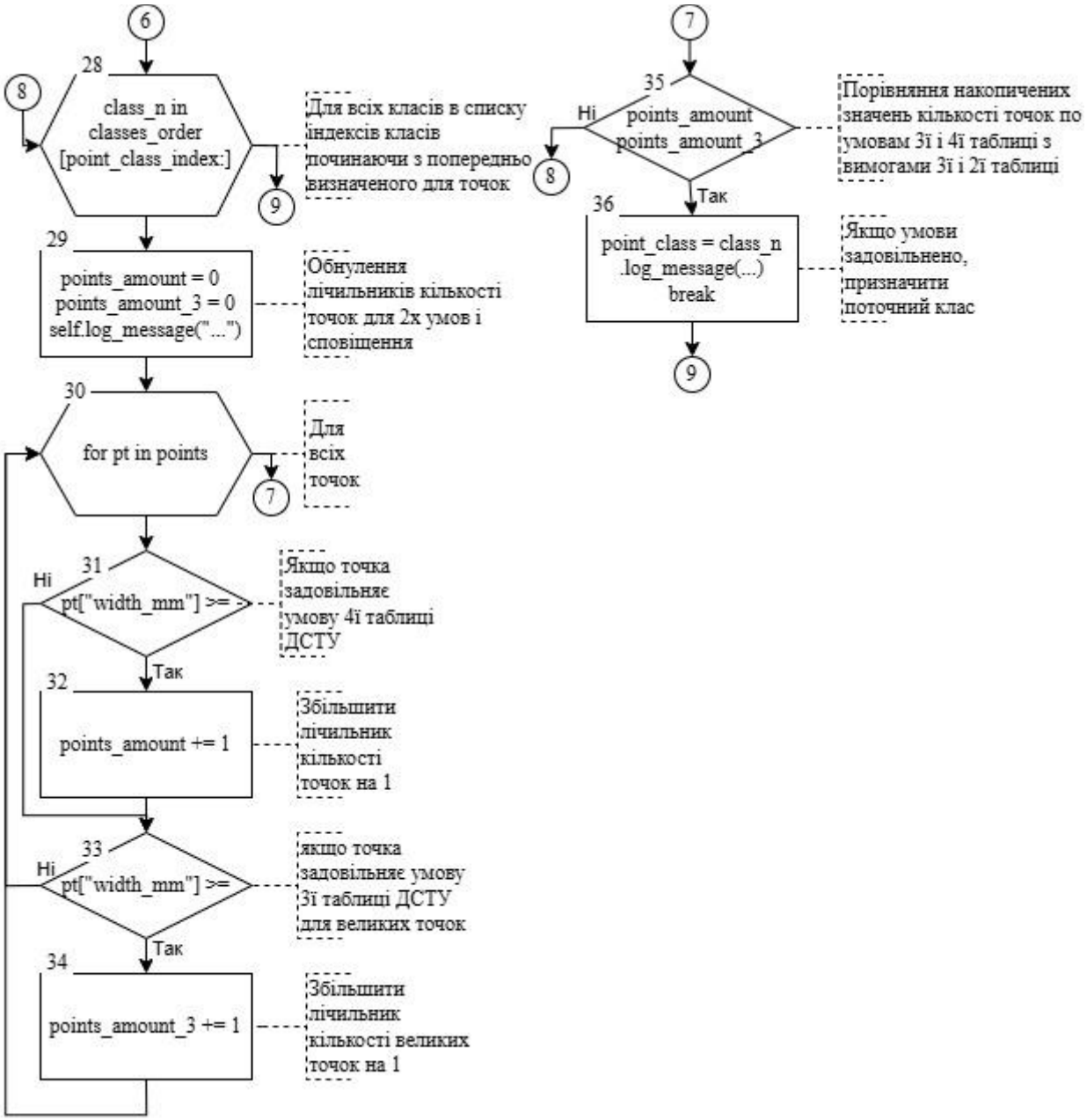


Рис. 3.14 Блок-схема алгоритму функції evaluate_dstu_class(self, light_diameter_mm) частина 4

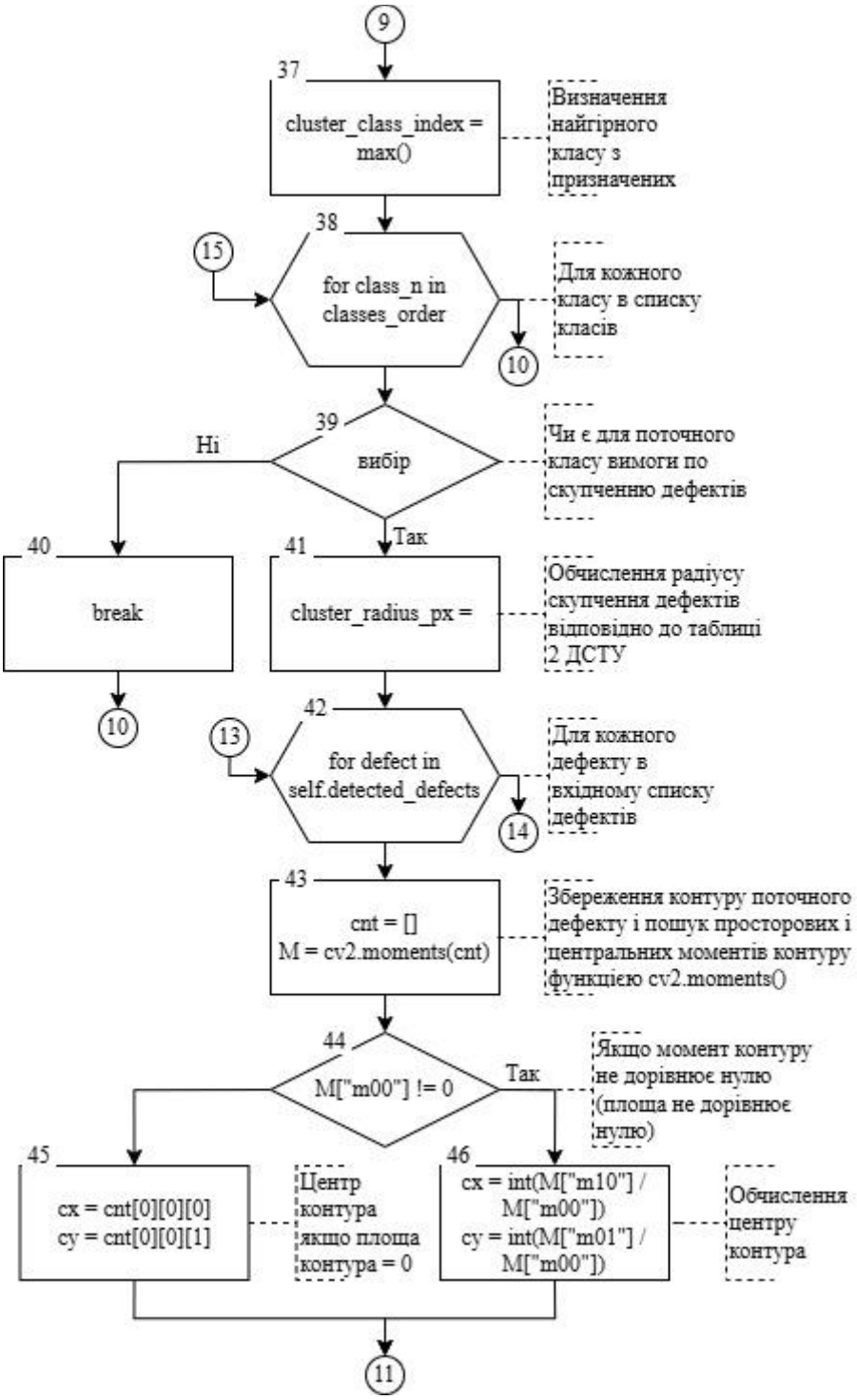


Рис. 3.15 Блок-схема алгоритму функції evaluate_dstu_class(self, light_diameter_mm) частина 5

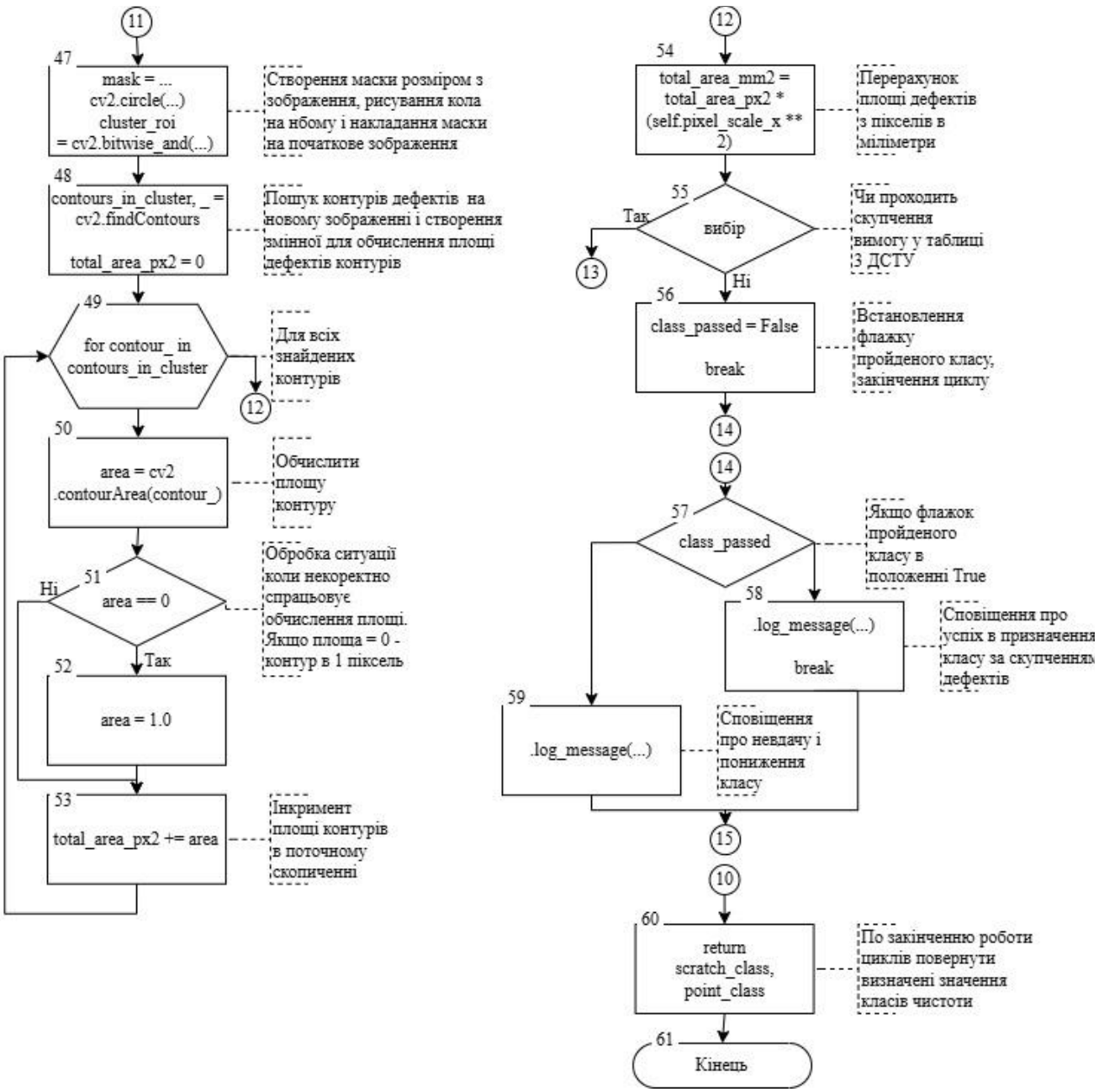


Рис. 3.16 Блок-схема алгоритму функції evaluate_dstu_class(self, light_diameter_mm) частина 6

Пояснення блоків блок-схеми алгоритму функції evaluate_dstu_class(self, light_diameter_mm) (див. позначення на рис 3.11, 3.12, 3.13, 3.14, 3.15, 3.16) за їх номерами:

1. Початок виконання функції.

2. Створення змінних classes_order і dstu_classes_requirements що зберігають в собі порядок класів чистоти і вимоги до всі параметрів для всіх класів чистоти.

3. Цикл що проходиться по всім об’єктам дефектів зі писку дефектів self.detected_defects, що були передані в функцію.

4. Якщо тип поточного дефекту зі списку дефектів self.detected_defects – царапина.

5. Додати об’єкт поточного дефекту до списку царапин scratches.

6. Якщо тип поточного дефекту зі списку дефектів self.detected_defects – точка.

7. Додати об’єкт поточного дефекту до списку точок points.

8. Визначення найбільшої ширини царапини зі списку об’єктів царапин scratches.

9. Визначення найбільшого діаметру точки зі списку об’єктів точок points.

10. Цикл всередині якого реалізоване визначення класу чистоти за ознакою максимально допустимої ширини царапини. ДСТУ11141-84 в таблиці 2 регулює максимально допустиму ширину царапини для кожного класу чистоти. Цю ознаку перевірити найлегше з усіх. Перевірка цього параметру в першу чергу дозволяє суттєво зекономити час обробки зображення шляхом уникнення виконання зайвих перевірок, бо якщо найбільша ширина царапини відповідає вимогам, на приклад, классу VII і нижче, то немає сенсу перевіряти відповідність іншим вимогам по царапинам для класів вищої якості, бо вже зрозуміло що деталь не буде відповідати всім вимогам вищих класів чистоти. Отже спочатку проводиться перевірка за максимальною шириною царапин, призначається попередній клас чистоти за царапинами, і потім починаючи з цього класу чистоти будуть проводитись перевірки на інші вимоги.[7]

11. Порівняння найбільшої ширини царапини з вимогами класу, якщо вимоги не виконано то клас чистоти понижається і вимоги перевіряються знову.

12. Якщо максимальна ширина царапини відповідає вимогам, призначити попередньо визначений за ознакою максимальної ширини царапин клас чистоти.

13. Якщо максимальна ширина відповідає вимогам, перервати виконання циклу в блоці 10.

14. Цикл з реалізацією пошуку класу чистоти за ознакою точки з найбільшим діаметром. Схоже до циклу в блоці 10, але тут перевіряється вимога не на максимальну ширину црапини, а на максимальний діаметр точки. Це також суттєво допомагає прискорити виконання аналізу, уникаючи нераціональних перевірок.

15. Порівняння найбільшого діаметру точки з вимогою в таблиці 2 ДСТУ для поточного класу чистоти, якщо вимогу не виконано то понижаємо клас чистоти.

16. Якщо вимогу по максимальному діаметру точок виконано, попередньо призначаємо поточний клас як клас чистоти по точкам. [7]

17. Переривання виконання циклу в блоці 14 за умови що знайдено клас чистоти вимогам якого відповідає діаметр найбільшої знайденої точки.

18. Пошук індексу значень попередньо визначених класів чистоти за ознаками царапин і точок в змінній classes_order() і запис їх в змінні scratch_class_index = classes_order.index(scratch_class_by_w) і point_class_index = classes_order.index(point_class_by_d). Ця операція є частиною оптимізації алгоритму пошуку класів чистоти, бо індекси потрібні для визначення того з вимог для кого класу починати інші перевірки. Наприклад попередньо було визначено клас чистоти по царапинм V, в змінній scratch_class_index після виконання scratch_class_index = classes_order.index(scratch_class_by_w) буде записана цифра 4 (бо індекси масивів починаються з 0), тоді цикл що оцінює царапини по іншим вимогам почне оцінювати по вимогам описаним в об’єкті з 4м індексом в словнику dstu_classes_requirements.

19. Цикл для перевірки умов по царапинам з таблиць 2, 3 і 4. Проходиться по всім вимогам до класів чистоти, починаючи з попередньо визначеного класу чистоти по максимальній ширині царапин, і перевіряє відповідність всім вимогам. [7]

20. Обнулення змінних scratch_total_l_mm і scratch_total_l_mm_3, що є лічильниками для сумарної довжини царапин з шириною за умовами таблиці 4 ДСТУ і сумарної довжини великих царапин. [7]

21. Внутрішній цикл що проходиться по всім царапинам з масиву scratches всередині якого обчислюється сумарна довжина царапин за розмірами в межах описаних 4ю таблицею і розмірами для великих царапин описаними 3ю таблицею.

22. Якщо ширина царапини відповідає межам описаним в 4й таблиці ДСТУ. [7]

23. Додати довжину царапини до лічильника.

24. Якщо ширина царапини відповідає межам для великих царапин описаним в таблиці 3 ДСТУ. [7]

25. Додати її довжину до лічильника.

26. Порівняти обчислені сумарні довжини царапин з вимогами ДСТУ, якщо вимоги не виконані, понизити якість класу чистоти.

27. Якщо умови виконані, призначити деталі поточний клас чистоти по царапинам, перервати виконання циклу перевірок.

28. Цикл для перевірки умов по точкам з таблиць 2, 3 і 4. Проходиться по всім вимогам до класів чистоти, починаючи з попередньо визначеного класу чистоти по максимальному діаметру точки, і перевіряє відповідність всім вимогам.

29. Скидання змінних points_amount і points_amount_3 до нуля, що є лічильниками для кількості точок діаметру за умовами таблиці 4 ДСТУ і кількості великих точок за умовами таблиці 3 ДСТУ. [7]

30. Внутрішній цикл що проходиться по всім точкам з масиву points всередині якого обчислюється сумарна кількість точок з діаметром в межах описаних 4ю таблицею і діаметром для великих точок описаним 3ю таблицею.

31. Якщо діаметр точки виконує умови в 4й таблиці ДСТУ. [7]

32. Збільшити лічильник підходящих за вимогою 4ї таблиці точок на 1.

33. Якщо діаметр точки відповідає межам для великих точок описаним в таблиці 3 ДСТУ. [7]

34. Збільшити лічильник великих точок на 1.

35. Порівняти обчислені кількості точок з вимогами ДСТУ, якщо вимоги не виконані, понизити якість класу чистоти. [7]

36. Якщо умови виконані, призначити деталі поточний клас чистоти по точкам, перервати виконання циклу перевірок.

37. Визначення гіршого класу чистоти з призначених за ознаками по точкам і по царапинам. Потрібно для аналізу по скупченню дефектів. По схожій логіці оптимізації алгоритму аналізу, перевірки по скупченню дефектів почнуть виконуватись з найгіршого з попередньо визначених класів, бо вже зрозуміло що деталь не пройде по всім вимогам кращих класів.

38. Цикл перевірок класу чистоти за ознакою скупчення дефектів у відповідності до вимог з таблиці 2 ДСТУ. Проходиться по всім вимогам до класів чистоти, починаючи з найгіршого визначеного класу чистоти. [7]

39. Перевірка на те чи є для поточного найгіршого класу чистоти вимоги на скупчення дефектів. Це важливо перевірити, бо ДСТУ висуває вимоги на скупчення дефектів тільки для класів до VII включно, а класи вище не мають вимог, і не має сенсу робити цю перевірку для них. [7]

40. Якщо вимог до поточного класу немає – перервати виконання циклу з блоку 38.

41. Якщо ДСТУ висуває вимоги по скупченню дефектів для поточного класу чистоти, обчислити діаметр кола в якому має контролюватись скупчення дефектів.

42. Запуск циклу що проходиться по кожному дефекту в визначених на зображення дефектів, щоб перевірити скупчення дефектів навколо нього.

43. Отримання інформації про контур поточного дефекту з структури даних дефекту і визначення його моментів і запис їх в змінну M = cv2.moments(cnt). Момент – термін з математики що означає геометричну характеристику. Обчислення моментів контуру дає можливість визначити різні чисельні характеристики контуру, такі як його площа і центр по 2м осям, як раз ці характеристики контуру і потрібні для його аналізу.[46]

44. Якщо обчислена площа контуру не дорівнює нулю.

45. Знайти центр контуру для випадків коли його площа дорівнює нулю.

46. Обчислити центр контуру для випадків коли площа контуру не дорівнює нулю.

47. Створення чорної маски розміром з вхідне зображення, рисування кола з діаметром рівним діаметру зони для контролю скупчення дефектів і накладання її на зображення з дефектами.

48. Пошук контурів дефектів в зоні діаметром контролю за ознакою скупчення дефектів і запис результату в змінну contours_in_cluster, _ = cv2.findContours(cluster_roi, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE). Скидання до нуля значення лічильника площі дефектів в пікселях квадратних.

49. Цикл що проходиться по всім виявленим в зоні контролю скупчення дефектів контурам і рахує сумарну площу дефектів.

50. Обчислити площу контуру.

51. Обробка ситуації коли дефект розміром в 1 піксель і його площа некоректно обчислюється.

52. Якщо площа дефекту = 0, призначити їй значення 1.

53. Збільшити лічильник загальної площі дефектів виявлених в зоні контролю скупчення дефектів.

54. Обчислення загальної площі дефектів в зоні контролю скупчення дефектів у міліметрах і запис результату в змінну total_area_mm2 = total_area_px2 * (self.pixel_scale_x ** 2).

55. Перевірка чи відповідає сумарна площа дефектів в зоні скупчення дефектів вимогам ДСТУ з таблиці 2. Якщо відповідає то повернутись до циклу в блоці 42 для перевірки скупчення навколо наступного дефекту.

56. Якщо не виконані вимоги класу то встановити бінарну змінну class_passed = False і перервати виконання циклу в блоці 42, бо немає сенсу далі перевіряти контури, бо вже зрозуміло що вимоги не виконані.

57. Перевірка стану бінарної змінної class_passed.

58. Якщо вимоги до класу не пройдені – вивести відповідне сповіщення і почати нову ітерацію циклу в блоці 38, понизивши вимоги до класу чистоти.

59. Якщо вимоги до поточного класу чистоти з циклу 38 було виконано, вивести відповідне сповіщення і перервати виконання циклу 38, бо вже не потребується пониження вимог.

60. Коли всі перевірки було виконано і визначено класи чистоти за ознаками по царапинам, по точкам і по скупченням дефектів – повернути значення визначених класів return scratch_class, point_class в батьківську функцію process_image_trigger(self).

61. Кінець виконання функції.

В алгоритмі перевірки умов по скупченню дефектів є недолік. Алгоритм побудований таким чином щоб малювати коло з центром в центрі кожного знайденого дефекту і обчислювати суму площі дефектів в цьому колі. Абсолютно можливо що дефекти будуть розташовані таким чином що центр кола їх скупчення не буде співпадати з центром одного із дефектів, а насправді сума площі дефектів в

якомусь колі буде виходити за задані мені. Це велике алгоритмічне питання, яке не розглядається, бо його вирішення статистично не вплине на результат роботи програми. Такий висновок заснований на результатах роботи програми і перевірках на більше ніж 30ти різних фото поверхонь лінз, в яких не було жодного випадку коли лінза проходила вимоги по царапинам і точкам і не проходила вимоги по скупченню дефектів. Це не виключає ймовірність ситуацій коли через особливості алгоритму не буде виявлено завеликого скупчення.

3.7. Демонстрація роботи програми

З метою перевірки і демонстраці роботоздатності розробленого програмного забезпечення, перевірки коректності математичних обчислень, алгоритму прийняття рішень було проведено декілька запусків програми і представлено результати в цьому розділі.

Зображення поверхонь лінз, використані для демонстрації роботи програмного забезпечення в цьому розділі, були отримані за допомогою стереоскопічного мікроскопа МБС-10 та цифрової камери SIGETA MDC-500 5.0 MP. Камера і мікроскоп були підключені до ПК відповідно до схеми на рис. 2.1.

Далі приведено серію скріншотів головного вікна програми, вхідних і оброблених програмою зображень і результатів обробки. Для кожного зображення окремо налаштовувалось співвідношення мм / піксель.

На рис. 3.17 зображено вікно програми після обробки тестового зображення №1. На ньому вижно вхідне і оброблене зображення, параметри обробки, лог з усіма сповіщеннями про роботу програми під час обробки, і результати обробки. На рис. 3.18 зображення до і після обробки, під ними написані результати обробки.

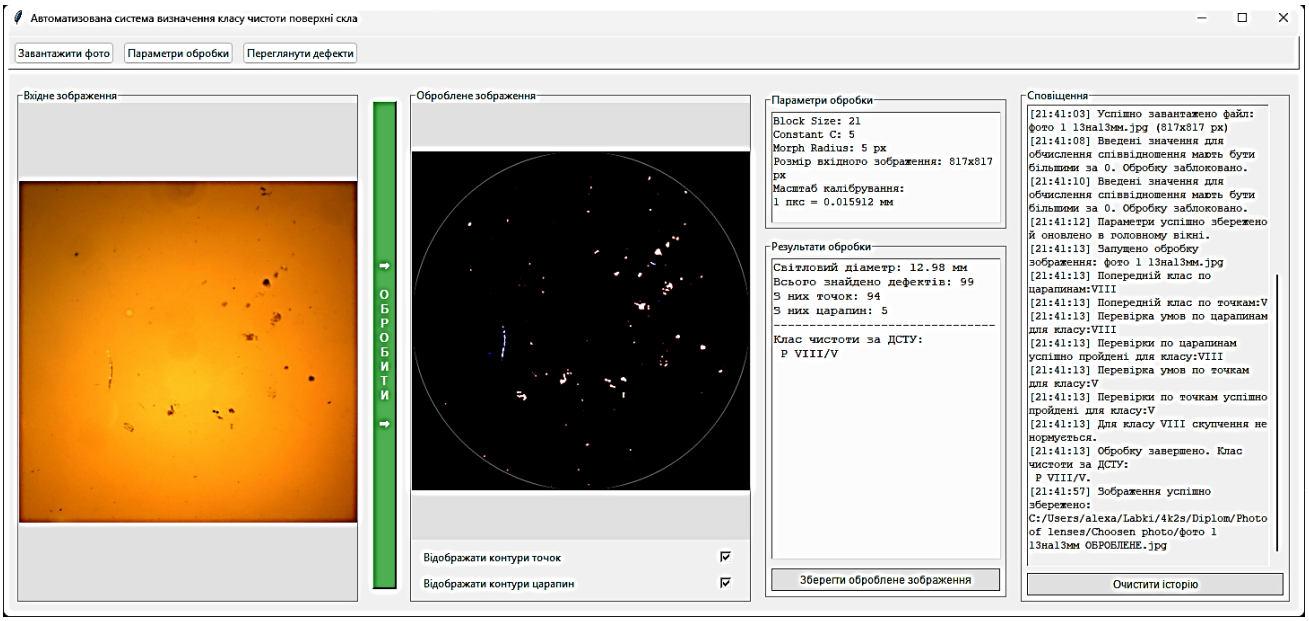


Рис. 3.17 Скріншот головного вікна програми після обробки зображення 1

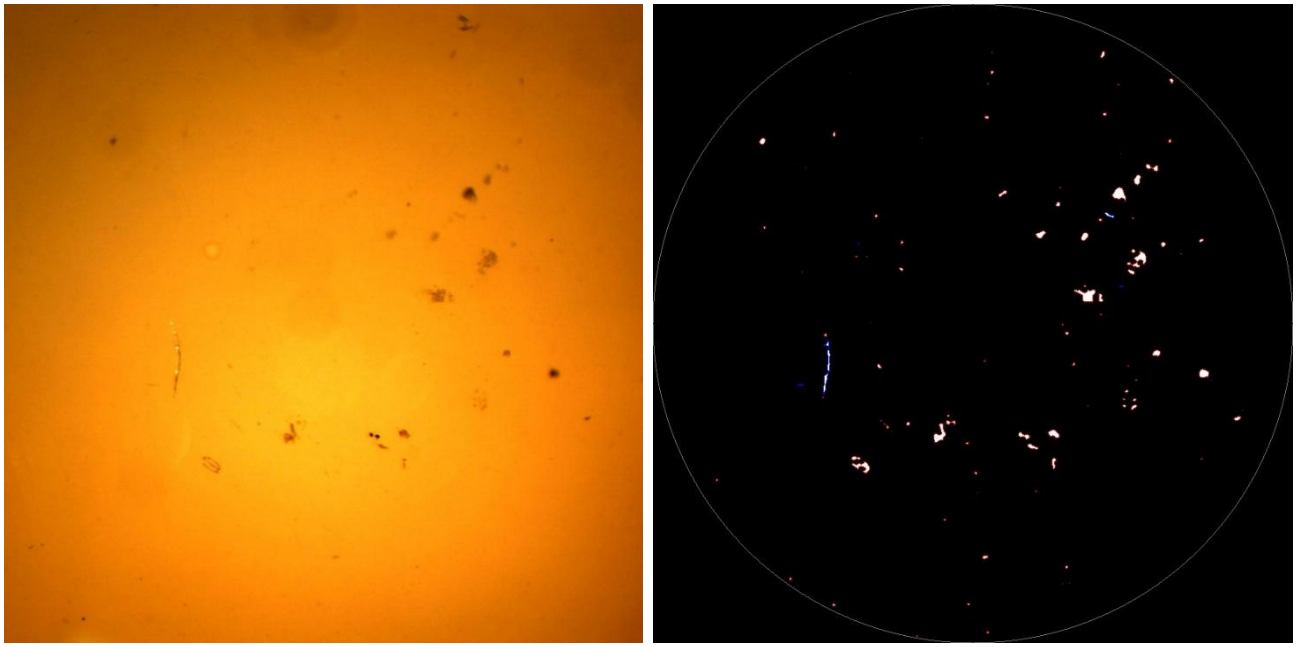


Рис. 3.18 Зображення 1 до обробки(зліва), зображення 1 після обробки з наведеними контурами дефектів (справа)

Результати обробки зображення №1:

- Світловий діаметр: 12,98мм
- Всього знайдено дефектів: 99
- З них точок: 94
- З них царапин: 5
- Клас чистоти за ДСТУ: P VIII/V

На рисунках 3.19 і 3.20 зображені головне вікно програми, зображення до і після обробки і нижче результати обробки.

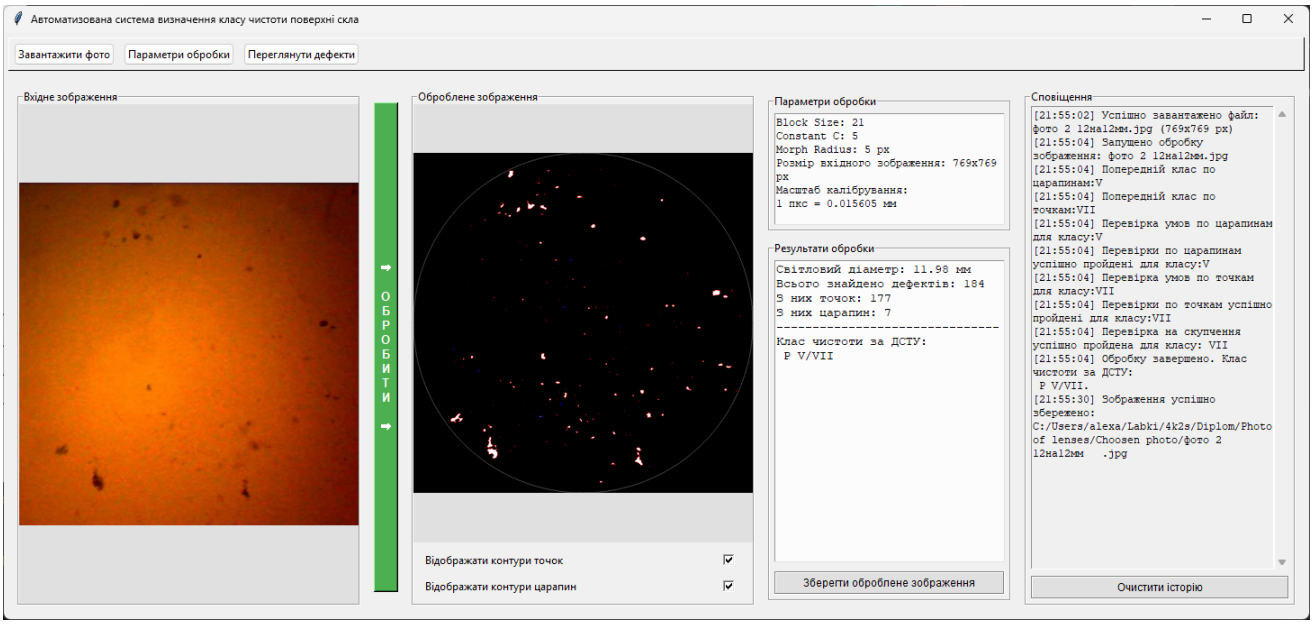


Рис. 3.19 Скріншот головного вікна програми після обробки зображення 2

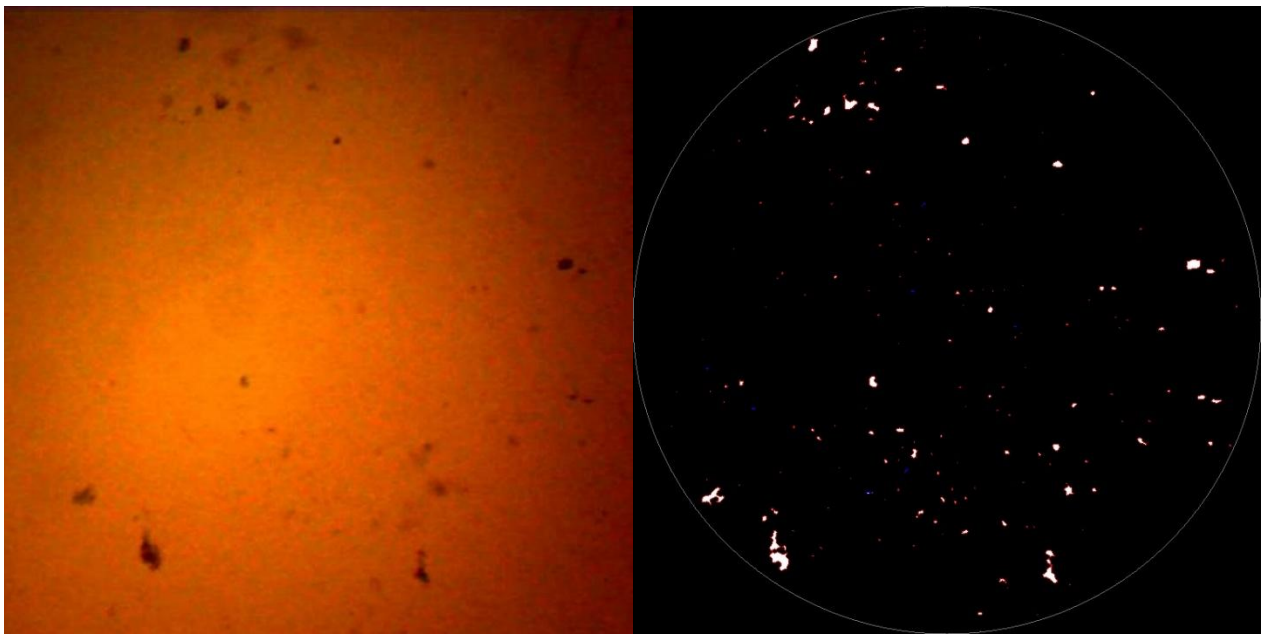


Рис. 3.20 Зображення 2 до обробки(зліва), зображення 2 після обробки з наведеними контурами дефектів (справа)

Результати обробки зображення №2:

- Світловий діаметр: 11,98мм
- Всього знайдено дефектів: 184
- З них точок: 177

- З них царапин: 7
- Клас чистоти за ДСТУ: P V/VII

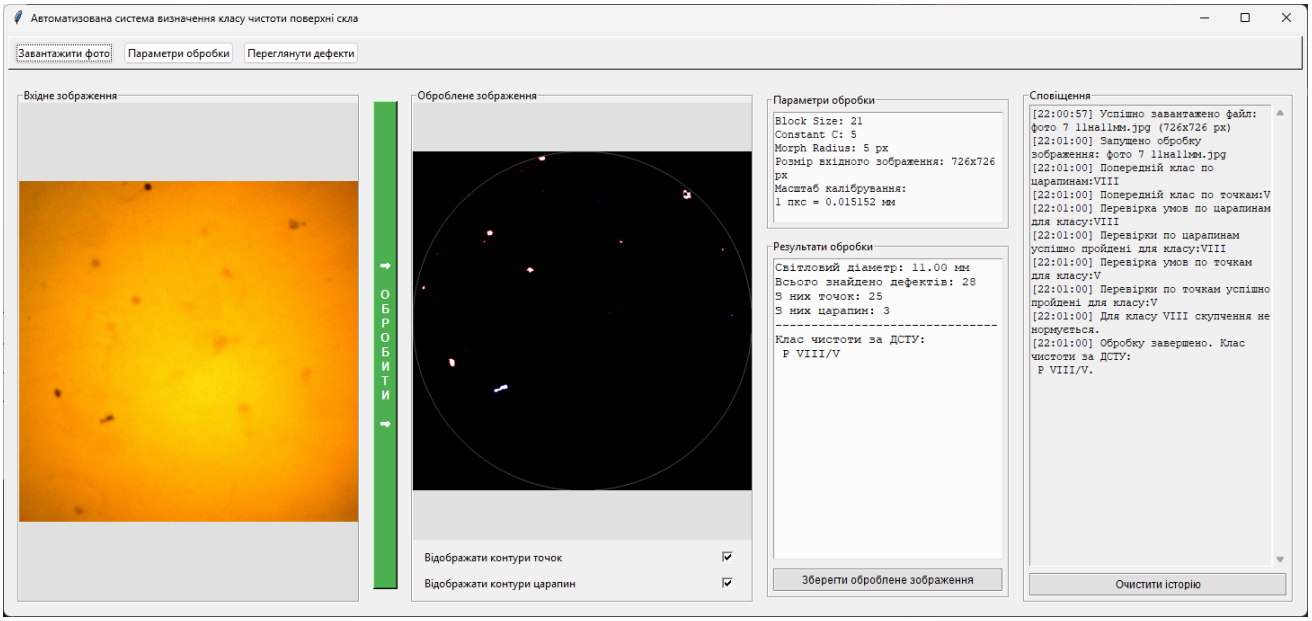


Рис. 3.21 Скріншот головного вікна програми після обробки зображення 3

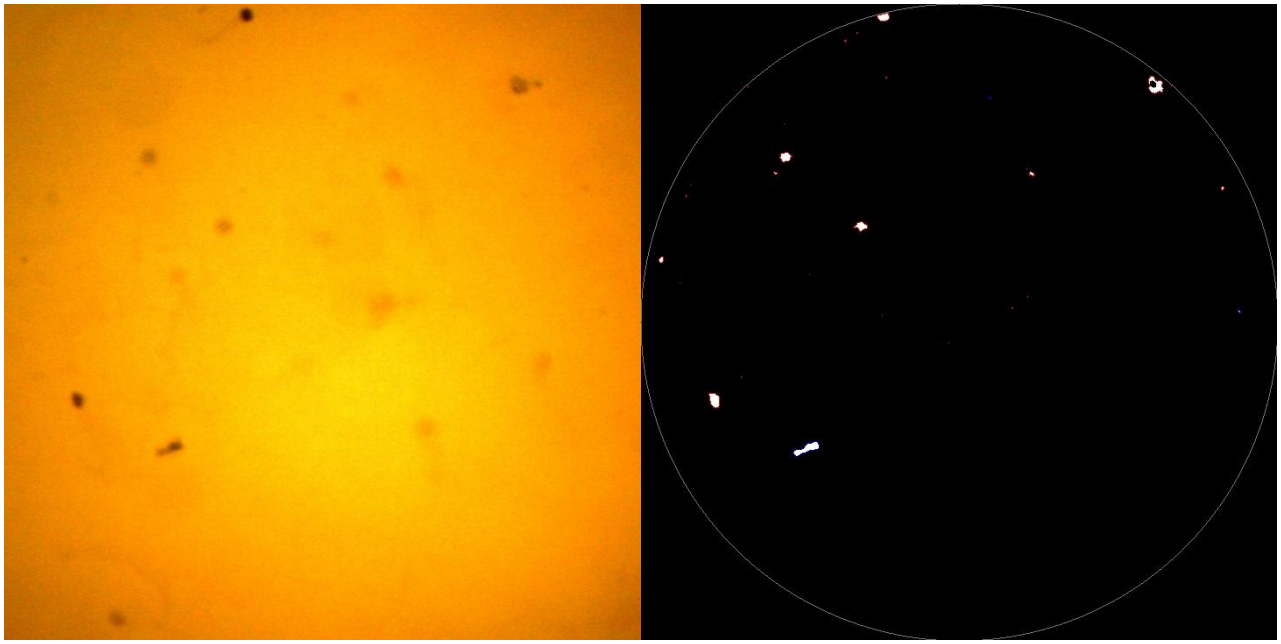


Рис. 3.22 Зображення 3 до обробки(зліва), зображення 3 після обробки з наведеними контурами дефектів (справа)

Результати обробки зображення №3:

- Світловий діаметр: 11,00мм
- Всього знайдено дефектів: 28

- З них точок: 25

- З них царапин: 3

- Клас чистоти за ДСТУ: P VIII/V

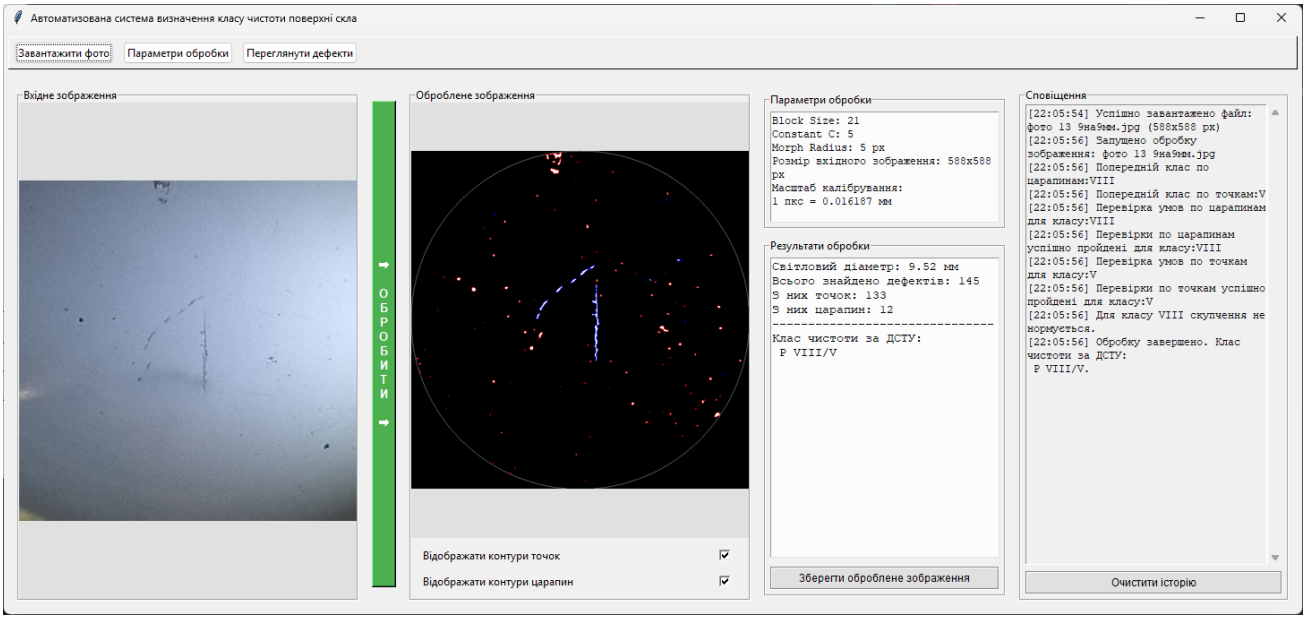


Рис. 3.23 Скріншот головного вікна програми після обробки зображення 4

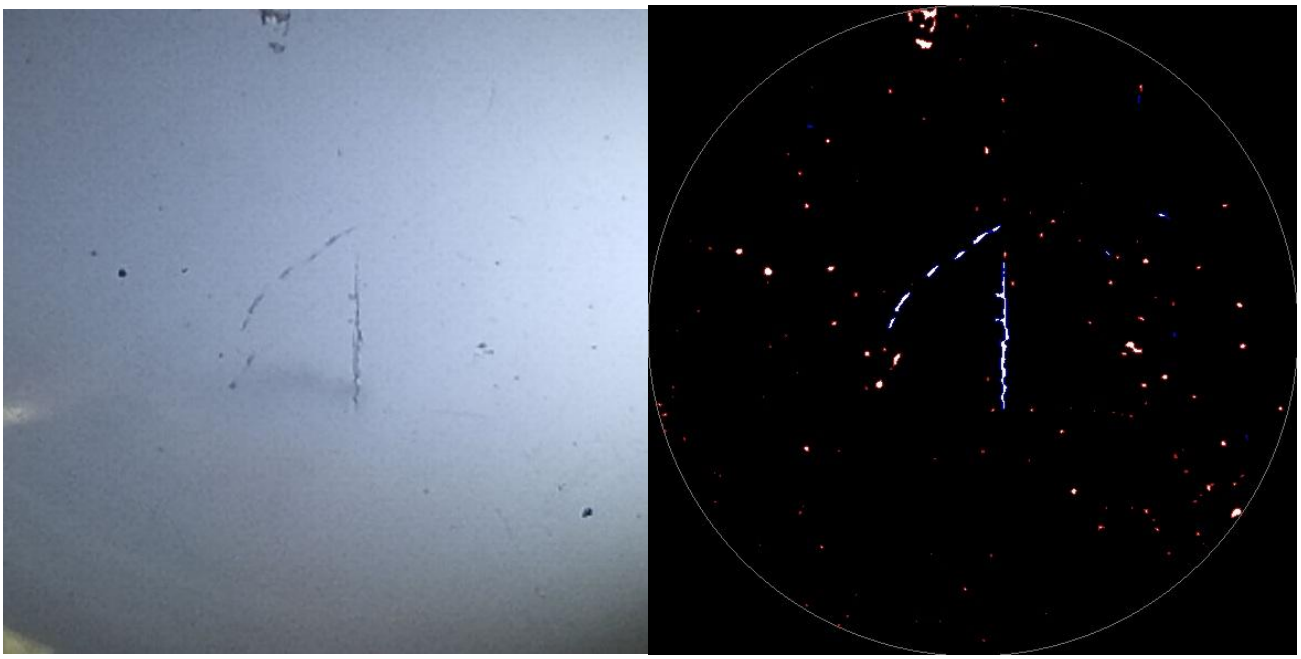


Рис. 3.24 Зображення 4 до обробки(зліва), зображення 4 після обробки з наведеними контурами дефектів (справа)

Результати обробки зображення №4:

- Світловий діаметр: 9,52мм

- Всього знайдено дефектів: 145

- З них точок: 133

- З них царапин: 12

- Клас чистоти за ДСТУ: P VIII/V

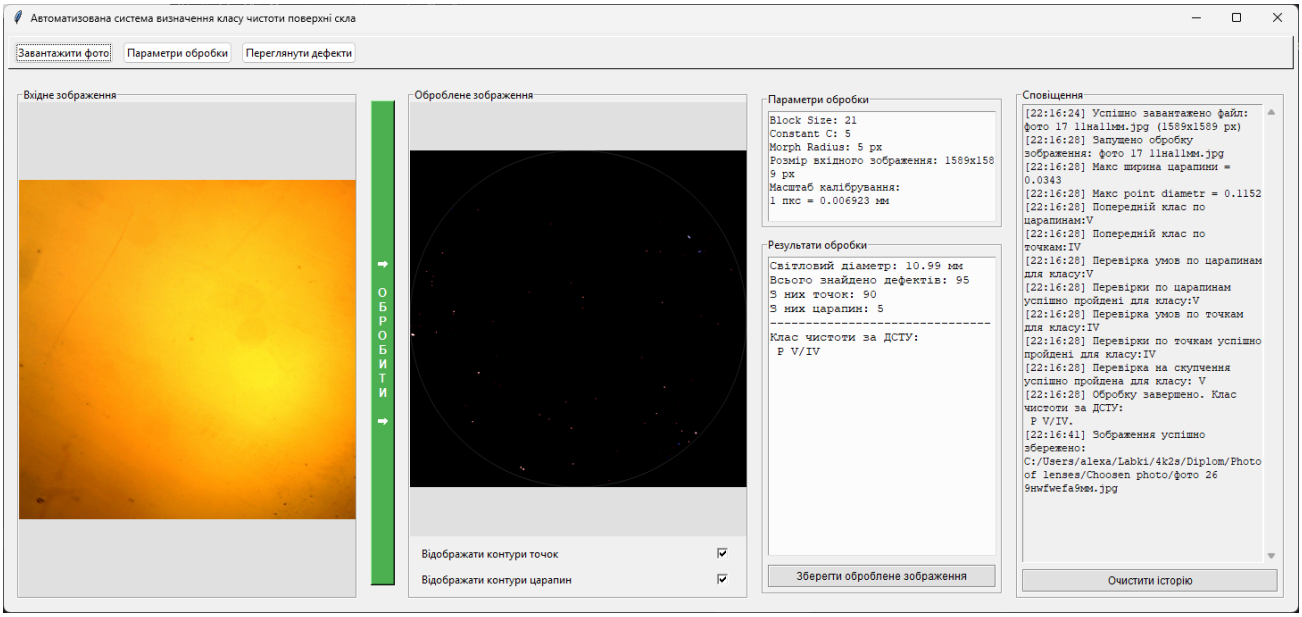


Рис. 3.26 Скріншот головного вікна програми після обробки зображення 5

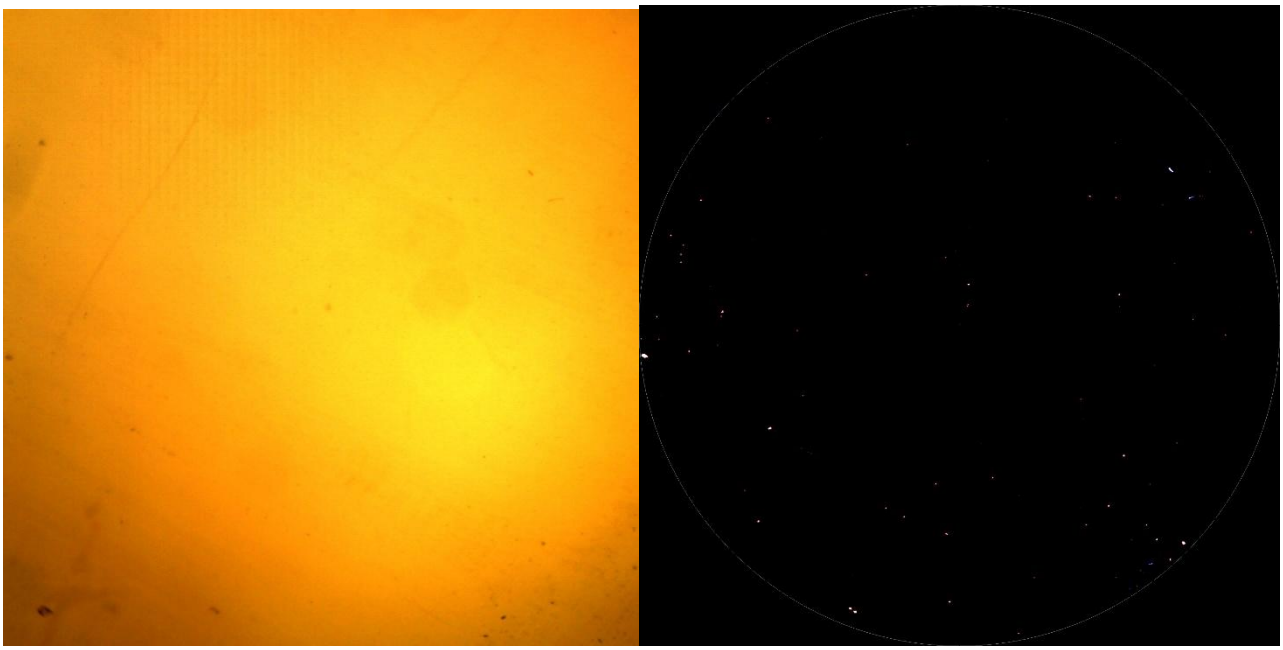


Рис. 3.26 Зображення 5 до обробки(зліва), зображення 5 після обробки з наведеними контурами дефектів (справа)

Результати обробки:

- Світловий діаметр: 10,99мм

- Всього знайдено дефектів: 95
- З них точок: 90
- З них царапин: 5
- Клас чистоти за ДСТУ: P V/IV

По результатам роботи програми видно що найвищим класом чистоти по царапинах є V і найвищим класом по точках є IV. Більш якісні класи не було продемонстровано через відсутність відповідних зображень, а їх в свою чергу не було через відсутність деталей з витриманими кращими класами чистоти, поверхню яких можна було би проаналізувати.

Також детально ознайомившись з результатами обробки можна побачити що в усіх представлених випадках попередньо визначені класи чистоти по максимальній ширині царапини і максимальному діаметру точки завжди тільки підтверджувались і ставали фінальними, без пониження якості класу при подальшій перевірці відповідності деталі вимогам ширини і сумарної довжини царапин, максимального розміру і кількості точок і скупченню дефектів з таблиць 2, 3 і 4 ДСТУ. Це свідчить про ефективність алгоритму визначення класу чистоти і про те що рішення про попередню оцінку за цими ознаками було правильне. Однак навіть якщо провести перевірку на десятках тисяч зображень поверхонь лінз, і при всіх цих перевірках буде повторюватись та сама закономірність підтвердження попередньо визначених за ознаками максимальної ширини царапини і максимального діаметру точок, не можна прибирати ці перевірки, бо в такому разі критерії перевірок не будуть відповідати стандарту ДСТУ 11141-84 і система не буде відповідати своїм початковим вимогам.[7]

ВИСНОВКИ ДО РОЗДІЛУ 3

3.1 Обґрунтовано вибір мови програмування і бібліотек для розробки програмного забезпечення. Було вибрано мову Python, для обробки зображень і взаємодії з масивами чисел було обрано бібліотеки комп’ютерного зору і матричних обчислень OpenCV і NumPy, для реалізації інтерфесу програми було вибрано Tkinter і Pillow для відображення зображень в інтерфейсі.[27][28][29][30][31]

3.2. Розроблено та впроваджено програмно алгоритм попередньої обробки кадрів, що складається з операцій фільтрації Гаусса cv2.GaussianBlur(), що прибирає зайві шуми на зображенні, адаптивної бінаризації, що виділяєдефекти cv2.adaptiveThreshold() і морфологічного закриття cv2.morphologyEx(), і забезпечує стійку сигментацію контурів дефектів та об’єднання малих фрагментованих дефектів.[37][38]

3.3. Розроблено алгоритм і реалізовано програмно математичний апарат виявлення параметрів дефектів таких як довжина, ширина і площа за допомогою використання функцій cv2.minAreaRect() що визначає мінамальний описаний навколо контуру прямокутник і cv2.contourArea() що визначає площу контуру. Реалізовано класифікацію знайдених дефектів царапина / точка по критерію відношення сторін дефекту 3:1, встановленого ДСТУ 11141-84. Також використовувалась функція cv2.moments() для пошуку двовимірних просторових дефектів для визначення центрів дефектів, що було потрібно для реалізації алгоритму перевірки класу чистоти за критерієм площі дефектів в зоні їх скупчення.[7][11][43][44][45][46]

3.4. Було створену оптимізовану логіку прийняття рішень по призначенню класу чистоти поверхні. Замість того щоб для всіх дефектів проводити перевірки за критеріями кожного класу по порядку, спочатку проводиться найпростіші перевірки – за максимальною шириною царапини і максимальним діаметром точки, за результатом яких попередньо призначаються класи чистоти і вже потім проводяться всі перевірки починаючи з критеріїв попередньо визначених класів. Обраний алгоритм дослідним шляхом довів свою ефективність. Зараз аналіз зображення 1500 на 500 пікселів займає менше 1 секунди, без впровадження оптимізованого алгоритму

перевірка тривала би до 5ти секунд. Це не звучить як багато, але в умовах масового виробництва ця економія часу була би суттєво відчутна.

3.5.Спроєктовано візуальний інтерфейс програмного забезпечення засобами бібліотеки Tkinter. Він містить поля для відображення зображень до і після обробки, поле з поточними налаштуваннями і меню де ці налаштування можна змінити, при цьому зміни налаштувань обмежені здоровим глуздом. Також є зона в якій відображаються результати аналізу і розроблений інтерфейс для перегляду детального переліку всіх знайдених на зображенні дефектів і інформації про них, з можливістю зміни порядку відображення фільтрацією по різним параметрам. Є поле з системними сповіщеннями в яке виводиться важлива інформація і сповіщення щодо етапів обробки. Є функція збереження обробленого зображення.[30]

3.6.Проведено демонстраційні обробки тестових зображень що показали дієздатність, адекватність і швидкодію роботи програмного модулю системи.

РОЗДІЛ 4 ПРОЄКТУВАННЯ МОДУЛЮ ТРАНСПОРТУВАННЯ І СОРТУВАННЯ ДЕТАЛЕЙ

4.1. **Розробка структури модуля транспортування і сортування деталей**

Відповідно до розробленої в параграфі 2.1. структурної схеми автоматизованої системи, в системі є апаратний транспортний модуль (рис. 2.1. позначення 6), задачами якого є забезпечення автоматичного переміщення деталей всередині автоматизованої системи. Без цього модулю система просто не є автоматизованою.

На структурній схемі на рисунку 2.1 зображено що модуль автоматичного транспортування і сортування деталей складається з таких елементів:

- Позначення 7 – касетний накопичувач з деталями на вхід.
- Позначення 8 – промисловий робот
- Позначення 9 – касетний накопичувач з деталями на вихід, що відповідають вимогам заданого класу чистоти
- Позначення 10 – касетний накопичувач з деталями на вихід що не відповідають вимогам заданого класу чистоти.

Касетний накопичувач це пристосування що має індивідуальні місця для лінз, використовується для накопичення і транспортування деталей під час виробництва. Приклади того як виглядають касетні накопичувачі на рисунках 4.1, 4.2.

Рис. 4.1. Приклад вигляду касетного накопичувача для лінз

Рис. 4.2. Приклад вигляду касетного накопичувача для лінз

Предметом дослідження в цьому розділі є саме промисловий робот що позначений номером 8 на рис. 2.1.

У промисловій автоматизації для транспортування сировини і деталей використовують такі засоби:

1. Стрічкові та роликові конвеєри. Представляють з себе довгу прямокутну доріжку яка рухається або за рахунок обертання стрічки, або за рахунок обертання роликів. Такі системи забезпечують високу продуктивність безперервного руху, проте мають низьку точність позиціонування деталі, що в об’єднанні з фактом наявності постійного тертя і інших фізичних чинників що призводять до пошкодження деталі, виключає можливе використання конвеєра для системи автоматичного визначення класу чистоти поверхні скла.

2. Лоткові і гравітаційні живильники. Характеризуються простотою конструкції та низькою вартістю, використовуються в основному для сипучих матеріалів для транспортування яких фактор відсутності пошкоджень не є важливим. Спричиняють неконтрольоване ковзання та зіткнення деталей між собою, що є

неприпустимим для прецизійної оптики, отже і не підходить для даної системи.

3. Промислові роботи що складаються з маніпулятора і захоплюючого пристрою. Забезпечують циклічне переміщення деталей. Маніпуляційні системи дозволяють повністю ізолювати деталь від контакту з іншими елементами лінії під час переносу, гарантуючи фіксовану траєкторію, точне позиціонування і відсутність додаткових пошкоджень.

Після оцінки існуючих засобів транспортування деталей, для розроблюваної системи сортування лінз було обрано промисловий робот-маніпулятор настільного типу. Даний вибір об ґрунтовано такими технічними і економічними факторами:

1. Висока повторюваність рухів. Відносно маленький промисловий робот має високу повторюваність рухів, що дозоляє забезпечити повторюване позиціонування деталі в касетних накопичувачах і модулі отримання зображення, що є важливим для коректної роботи алгоритмів комп’ютерного зору, бо суттєве зміщення оптичної осі сферичної лінзи може призвести до некоректного фокусування камери на поверхні лінзи, що негативно вплине на якість отриманого зображення і результат подальшого аналізу.

2. Невелика ціна. Використання настільного роботу легкого класу дозволяє уникнути суттєвих капіталовкладень в автоматизовану систему, притаманних важким промисловим роботам, що робить систему доступно для використання на виробництвах і в лабораторіях.

3. Проста інтеграція. Настільні роботи легкого класу керуються за допомогою відкритих протоколів і мають готові програмні бібліотеки для мови Python. Це дозволяє безпосередньо інтегрувати логіку управління фізичними рухами робота в розроблене у розділі 3 програмне забезпечення.

Найважливішим елементом маніпулятора є його робочий орган – захоплюючий пристрій. Захоплюючі пристрої бувають різних типів, класичним і найбільш розповсюдженим в приладобудуванні є пальцевий захоплювач. Такий робочий орган не підійде для транспортування вимогливих до відсутності пошкоджень скляних лінз діаметром 5мм через складності центрування, обмеженість простору і високу механічну напруженість в точках контакту металу зі склом.

З огляду на це, обґрунтованим є застосування вакуумного захоплюючого пристрою – «вакуумного присоска». Воно є підходящим з таких причин:

1. Відсутність нових дефектів від утримання. Силіконовий або фторопластовий еластичний присосок рівномірно розподіляє зусилля по поверхні лінзи, не залишаючи слідів своєї присутності на склі.

2. Можливість роботи з малими лінзами. Присосок з діаметром робочої зони 2-3мм здатен надійно фіксувати лінзи діаметром 5 мм за їхню сферичну, плоску чи асферичну поверхню, забезпечуючи стабільне перенесення без ризику випадіння деталі під дією сил інерції.

Отже – промисловий робот-маніпулятор з вакуумним присоском в якості робочого органу.

4.2. Підбір апаратних засобів для реалізації модуля транспортування і сортування деталей

Для практичної реалізації модуля транспортування і сортування деталей було підібрано обладнання, що задовольняє критеріям прецизійності, надійності та економічної доцільності. Ключовою перевагою обраного промислового робота і інших приладів є можливість написання керуючого програмного забезпечення для них на Python, що означає просту інтеграцію і можливість реалізації як функцій обробки і аналізу зображення так і функцій управління промисловим роботом в одній програмі.

В якості промислового робота для автоматизованої системи визначення класу чистоти поверхні скла було обрано UFactory Lite 6 від виробника Blue Sky Robotics. Зображення промислового робота на рис. 4.3. Вибір обґрунтовано такими причинами:[47]

1. Довжина досяжності або ж радіус робочої зони 440 мм, що повністю перекриває потреби у переміщенні між вхідним і вихідним потоками деталей і модулем отримання зображення.

2. Повторюваність позиціонування що становить ±0,1мм, що забезпечує достатньо стабільне позиціонування лінзи по центру оптичної осі.

3. Ціна в 3500$, що є значно дешевше за інші моделі інших виробників зі схожими характеристиками.[48]

4. Можливість налаштування керування роботом мовою Python, що надає можливість побудови програмного забезпечення що буде включати в себе керування всіма функціями автоматизованої системи.

Рис. 4.3 Промисловий робот UFactory Lite 6[47]

В якості вакуумного робочого органу для безпосереднього контакту з прецизійним склом лінз діаметром від 5мм було обрано присосок виробника SMC серії ZPT. Японська компанія SMC є світовим лідером у сфері промислової

пневматики, яка відома впровадженням найновітніших технологій і передових матеріалів. Зображення присоску на рис. 4.4.[49]

Рис. 4.4 Присосок від виробника SMC[49]

Присосок виготовлений із еластичного бездефектного силікону, що повністю виключає ризик пошкодження поверхні скляних деталей та не залишає на полірованій поверхні жирових чи пилових слідів, які могли б спотворити результати подальшого контролю чистоти.[49]

Діаметр присоску – 2мм, для деталей діаметром 5 мм. Для перевірки деталей більшого розміру потрібно змінювати присосок на більший. Важливо забезпечити достатню силу притискання, щоб при різких рухах робота деталь не відлипала від присоска. Сила притискання квадратично пропорційно росте зі збільшенням діаметру присоску, бо кругла площа притискання має квадратичну залежність від радіусу кола.[49]

Подивившись на рисунок 4.4. можна чітко побачити що присосок сам по собі не здатен тримати деталь, для цього до нього потрібно додати апаратне забезпечення, а саме:

- Вакуумний модуль для промислового робота.

- Компресор повітря що здатен забезпечити необхідний для роботи вакуумного модулю об’єм повітря.

- Поліуретанова пневматична трубка по якій буде рухатись повітря під тиском.

- Фільтр регулятор що буде контролювати тиск вхідного на вакуумний модуль повітря і фільтрувати його від мікрочастинок для запобігання пошкодженню поверхні деталей.

В якості вакуумного модуля було обрано прилад моделі ZXP7*01 від виробника присоска, SMC. Вакуумний модуль зображений на рис. 4.5. Обґрунтування вибору:[50]

- Вакуумний присосок і вакуумний модуль виробляється один виробником, що означає легку інтеграцію.

- Даний модуль призначений для використання з невеликими промисловими роботами і керуюча програма для цього може бути написана на Python, що надає можливості інтеграції з основною програмою оцінки зображення і управління промисловим роботом.

*Рис. 4.5 Вакуумний модуль ZXP7*01 від виробника SMC для промислового робота[50]*

Навіть з додаванням вакуумного модуля система все ще не може забезпечити утримання деталі. Щоб утримання деталі стало можливим потрібно додати пневматичну систему забезпечення тиску що складається з компресора, фільтру-регулятора і пневматичної трубки.

У якості компресора було обрано GTM KCW1500-50L. Зображення компресора на рисунку 4.6. Обґрунтування вибору:[51]

- Безмасляне виконання компресора є жорсткою вимогою технологічного процесу, оскільки звичайні компресори викидають у повітря мікрочастинки мастила, що неприпустимо для чистої оптичної ділянки.

- Компресор здатен стабільно забезпечувати продуктивність 210л/хв, що з запасом перекриває потребу генератора вакууму в 57л/год.

Рис. 4.6 Компресор GTM KCW1500-50L[51]

У якості фільтра і регулятора тиску біло обрано SMC AW20-F02CE-B. Зображення фільтра-регулятора на рисунку 4.7. Цей пристрій виконує дві функції:[52]

- здійснює тонке очищення стисненого повітря від вологи та мікрочастинок пилу, захищаючи внутрішні клапани вакуумного модуля ZXP7*01 та деталь від

забруднення.

- знижує і стабілізує тиск, що надходить від компресора, до номінальних робочих для генератора вакууму 0,5 МПа

Рис. 4.7 Фільтр-регулятор тиску SMC AW20-F02CE-B[52]

У якості пневматичної трубки було обрано прозору поліуретанову трубку TBPU з зовнішнім діаметром 6мм і внутрішнім діаметром 4мм. Поліуретан забезпечує високу гнучкість, стійкість до зламів а механічного зносу під час постійних поворотів осей маніпулятора.[53]

ВИСНОВКИ ДО РОЗДІЛУ 4

4.1. Було проведено аналіз існуючих засобів і методів автоматизованого переміщення деталей, виробів і сировини. Встановлено, що для прецизійних скляних лінз діаметром від 5 мм традиційні конвеєрні та гравітаційні системи є неефективними через високий ризик механічного пошкодження або забруднення оптичних поверхонь. Обґрунтовано доцільність застосування настільного промислового робота з вакуумним робочим органом, що є єдиним бездефектним способом транспортування скляних деталей.

4.2. В якості промислового робота було обрано 6-осьовий промисловий робот UFactory Lite 6. Визначено, що його ринкова вартість в 3500$ є суттєво нижчою за існуючі світові аналоги, що забезпечує високу економічну ефективність впровадження. При цьому так технічні параметри маніпулятора як радіус досяжності 440 мм та повторюваність позиціонування 0,1 мм повністю задовольняють вимогам високої точності укладання деталей в оптичну вісь цифрової камери модуля отримання зображення.

4.3. Обрано робочий орган для промислового робота – вакуумний присосок виробника SMC діаметром 2мм, і вибрано до нього генератор вакууму від того ж виробника, компресор GTM KCW1500-50L, фільтр-регулятор SMC AW20-F02CE-B та гнучку поліуретанову трубку. Ця конфігурація робочого органу може забезпечити необхідне утримання деталей під час переміщення.

ВИСНОВКИ

1. На основі аналізу предметної області та нормативної документації, зокрема ДСТУ 11141-84, обґрунтовано необхідність автоматизації контролю класів чистоти скляних поверхонь. Визначено, що традиційний візуальний контроль вручну оператором характеризується високою суб’єктивністю, низькою повторюваністю результатів та недостатньою продуктивністю в умовах сучасного дрібносерійного та масового виробництва оптики для приладів. На основі порівняльного техніко-економічного аналізу існуючих оптичних та інтерференційних методів доведено перевагу використання цифрових систем комп'ютерного зору.

2. Спроектовано загальну модульну архітектуру та структурну схему системи, яка розділена на 4 взаємопов'язані функціональні блоки. Здійснено технічний підбір матеріального забезпечення для модуля отримання зображень поверхні деталі на базі цифрової камери SIGETA MDC-500 5МП і бінокулярного мікроскопа МБС-10, що дозволило забезпечити необхідну оптичну роздільну здатність та кадрування робочої зони у фокальній площині.

3. Розроблено та практично реалізовано програмне забезпечення на мові програмування Python з використанням бібліотек OpenCV і NumPy для обробки та сегментації зображень та Pillow і Tkinter для побудови візуального інтерфейсу програми. Створено комплекс логіко-математичних алгоритмів для автоматичного виявлення, бінаризації, контурного аналізу та геометричної класифікації дефектів за трьома ознаками: точки, царапини та їх локальні скупчення. Алгоритм забезпечує автоматичний розрахунок і фінальне присвоєння класу чистоти деталі у суворій відповідності до табличних критеріїв ДСТУ 11141-84, мінімізуючи вплив людського фактора.

4. Проведено проєктування контуру фізичного переміщення й сортування деталей. Обґрунтовано безальтернативність застосування еластичного вакуумного робочого органу для транспортування лінз діаметром від 5мм. Обрано 6ти осьовий настільний промисловий робот UFactory Lite 6, який за бюджетної вартості 3500$ забезпечує високу точність позиціонування 0,1мм та довжину досяжності 440 мм, що задовільняє потреби системи.

5. Сформовано вакуумну підсистему на базі приладів компанії SMC силіконової присоски серії ZPT та вакуумного блоку ZXP7-01, роботу якої забезпечено поршневим безмасляним компресором GTM KCW1500-50L та блоком підготовки повітря SMC AW20-F02CE-B. Зазначене компресорне рішення повністю виключає потрапляння мікрочастинок мастила чи вологи в зону контролю, що є критично важливим для збереження чистоти скла на оптичній дільниці.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Apollo Optical Components. "Quality Control in Optical Lens Manufacturing: An Engineering Application Note" Apollo Optical. Дата звернення: 12 трав. 2025. [Електронний ресурс]. Доступно: https://www.apollooptical.com/feeds/blog/optical-lenses-quality-control

2. R. E. Parks, “Book Rvw: Fabrication Methods for Precision Optics. By Hank H. Karow”, *Opt. Eng.*, т. 32, № 10, с. 2624, жовт. 1993. Дата звернення: 1 черв. 2026. [Електронний ресурс]. Доступно: https://doi.org/10.1117/1.oe.32.10.bkrvw1.

3. Laramy-K Optical. *How They're Made (Lenses and Coatings): Inside The Optical Lab*. (19 квіт. 2018). Дата звернення: 7 черв. 2026. [Онлайн-відео]. Доступно: https://www.youtube.com/watch?v=yL_02Cxkmxk

4. The Wall Street Journal. *China’s Dark Factories: So Automated, They Don't Need Lights | WSJ*. (18 лип. 2025). Дата звернення: 7 черв. 2026. [Онлайн-відео]. Доступно: https://www.youtube.com/watch?v=MCBdcNA_FsI

5. J. Pessan. "Inside China’s Dark Factories Revolution," Medium. (30 квітня. 2024) Дата звернення: 18 травня. 2026. [Електронний ресурс]. Доступно: https://medium.com/@julio.pessan.pessan/inside-chinas-dark-factories-revolution-50934748789f

6. Flexible Vision. "Flexible Vision - AI Edge Vision for Automation," Flexible Vision. Дата звернення: 22 груд. 2025. [Електронний ресурс]. Доступно: https://www.flexiblevision.com

7. Держстандарт СРСР, ДСТУ 11141-84. Деталі оптичні. Класи чистоти поверхні. Методи контролю. Москва, РФ: Видавництво стандартів, 1985.

8. International Organization for Standardization, ISO 10110-7:2017 Optics and photonics — Preparation of drawings for optical elements and systems — Part 7: Surface imperfections. Geneva, Switzerland: ISO, 2017.

9. DIOPTIC GmbH. "Automated surface quality inspection in optical manufacturing," DIOPTIC. Дата звернення: 3 черв. 2026. [Електронний ресурс].

Доступно: https://www.dioptic.de/en/automated-surface-quality-inspection-in-optical-manufacturing/

10. Omniscan. "Interferometry in Surface Measurement," Omniscan. Дата звернення: 3 черв. 2026. [Електронний ресурс]. Доступно: https://omniscan.co.uk/interferometry-in-surface-measurement/

11. DNAOP ГОСТ 11141-84. Деталі оптичні, класи чистоти поверхонь, методи контролю (71563) Дата звернення: 1 черв. 2026. [Електронний ресурс]. Доступно: https://dnaop.com/html/71563/doc-ГОСТ_11141-84

12. Evident Scientific. "Cleanliness Inspection System Can Now Be Used as a Digital Microscope," Evident Olympus. Дата звернення: 2 черв. 2026. [Електронний ресурс]. Доступно: https://evidentscientific.com/en/news/cleanliness-inspection-system-can-now-be-used-as-a-digital-microscope

13. N.S. Rosli, M.H.F.M. Fauadi, N.F. Awang, A.Z.M. Noor"Vision-Based Defects Detection for Glass Production based on Improved Image Processing Method," Journal of Advanced Manufacturing Technology (JAMT), vol. 12, no. 1, pp.2018. [Електронний ресурс]. Доступно: https://jamt.utem.edu.my/jamt/article/view/3935

14. Chuanxia Jian, Jian Gao, Yinhui Ao, "Automatic surface defect detection for mobile phone screen glass based on machine vision," Applied Soft Computing, vol. 52, Pages 348-358, March. 2017. [Електронний ресурс]. Доступно: https://www.sciencedirect.com/science/article/abs/pii/S1568494616305531

15. Francesco Adamo, Filippo Attivissimo, Attilio Di Nisio and Mario Savino, "An automated visual inspection system for the glass industry" in Proceedings of 16th IMEKO TC4 Symposium, sept 22-24 Florence, Italy, 2008. [Електронний ресурс]. Доступно: https://www.imeko.info/publications/tc4-2008/IMEKO-TC4-2008-141.pdf

16. DIOPTIC, "ARGOS – Surface Defect Inspection ". Дата звернення: 2 черв. 2026. [Електронний ресурс]. Доступно: https://www.dioptic.de/en/argos/

17. DIOPTIC. "ARGOS Matrix Systems. Дата звернення: 4 черв. 2026. [Електронний ресурс]. Доступно: https://www.dioptic.de/en/products/argos/argos-matrix-systems/

18. DIOPTIC "DIOPTIC ARGOS Matrix 200/300/400 Technical Specification" DIOPTIC. Дата звернення: 4 черв. 2026. [Електронний ресурс]. Доступно: https://www.dioptic.de/wp-content/uploads/2024/06/DIOPTIC_ARGOS-matrix-200-300-400_specifications_EN.pdf

19. Zygo Corporation, "ZeGage Pro 3D Optical Profiler". Дата звернення: 4 черв. 2026. [Електронний ресурс]. Доступно: https://www.zygo.com/products/metrology-systems/3d-optical-profilers/zegage-pro

20. Lambda Photometrics. "ZeGage Pro Metrology System Brochure," Lambda Photo. Дата звернення: 4 черв. 2026. [Електронний ресурс]. Доступно: https://www.lambdaphoto.co.uk/pdfs/Zygo/LAMBDA_ZeGagePro-Brochure.pdf

21. Zygo Corporation. "Industrial Metrology and Optical Profiling Solutions," LinkedIn. Дата звернення: 4 черв. 2026. [Електронний ресурс]. Доступно: https://www.linkedin.com/posts/zygo-corporation_metrology-optics-opticalprofilers-activity-7414999488005566465-tyaS/

22. Саблук Р. П., Денисенко А. А. «Проблеми підприємництва в Україні під час воєнного стану» Економіка та управління, 2024. [Електронний ресурс]. Доступно: https://cms.economics-management.e-u.edu.ua/uploads/4_10_af8fd88994.pdf

23. Volkan okbay, "Automated image processing for scratch detection on specular surfaces" a thesis submitted to the graduate school of natural and applied sciences of middle east technical university, Ankara, Turkey, September 2018. [Електронний ресурс]. Доступно: https://etd.lib.metu.edu.tr/upload/12622717/index.pdf

24. Tristan Salomon. *Inspection system - scratch on glass*. (11 верес. 2020). Дата звернення: 4 черв. 2026. [Онлайн-відео]. Доступно: https://www.youtube.com/watch?v=9669iR_DxWY

25. АТ ЛЗОС, Мікроскоп стереоскопічний МБС-10. Керівництво з експлуатації. Литкаріно, РФ: ЛЗОС.

26. SIGETA, "Цифрова камера для мікроскопа SIGETA MDC-500 5.0 MP," Sigeta.com.ua. Дата звернення: 6 черв. 2026. [Електронний ресурс]. Доступно: https://sigeta.com.ua/products/tsyfrova-kamera-dlya-mikroskopa-sigeta-mdc-500-5-0mp.html

27. Python Software Foundation, "The Python Language Reference," Python.org. Дата звернення: 7 черв. 2026. [Електронний ресурс]. Доступно: https://docs.python.org/3/reference/

28. G. Bradski, "The OpenCV Library," Dr. Dobb's Journal of Software Tools, 2000. Дата звернення: 7 черв. 2026. [Електронний ресурс]. Доступно: https://opencv.org/

29. Harris, C.R., Millman, K.J., van der Walt, S.J. *et al.* Array programming with NumPy. *Nature* **585**, 357–362 (2020). Дата звернення: 7 черв. 2026. [Електронний ресурс]. Доступно: https://doi.org/10.1038/s41586-020-2649-2

30. “Tkinter-GUI-Application-Development-Blueprints-Second-Edition”, *PacktPublishing*, б. д. Дата звернення: 4 черв. 2026. [Електронний ресурс]. Доступно: https://github.com/PacktPublishing/Tkinter-GUI-Application-Development-Blueprints-Second-Edition/tree/master

31. "Pillow (PIL Fork) Documentation," Read the Docs. Дата звернення: 7 черв. 2026. [Електронний ресурс]. Доступно: https://pillow.readthedocs.io/en/stable/reference/Image.html

32. Rafael C. Gonzalez, Richard E. Woods, “Digital Image Processing FOURTH EDITION”. New York, NY, USA: Pearson, 2018.

33. “Image thresholding guide”. Дата звернення: 7 черв. 2026. [Електронний ресурс]. Доступно: https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html

34. destroy() method. Дата звернення: 7 черв. 2026. [Електронний ресурс]. Доступно: https://www.geeksforgeeks.org/python/destroy-method-in-tkinter-python/

35. numpy.array(). Дата звернення: 6 черв. 2026. [Електронний ресурс]. Доступно: https://numpy.org/doc/stable/reference/generated/numpy.array.html

36. Колрні схеми зображень в NumPy [Електронний ресурс]. Доступно: https://www.scivision.dev/numpy-image-bgr-to-rgb/

37. Image bluring. Дата звернення: 6 черв. 2026. [Електронний ресурс]. Доступно: https://www.geeksforgeeks.org/python/python-image-blurring-using-opencv/

38. Morphological operations. Дата звернення: 6 черв. 2026. [Електронний ресурс]. Доступно: https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html

39. Numpy.shape(). Дата звернення: 7 черв. 2026. function [Електронний ресурс]. Доступно: https://numpy.org/doc/stable/reference/generated/numpy.shape.html

40. Numpy.zeros_like(). Дата звернення: 7 черв. 2026. [Електронний ресурс]. Доступно: https://numpy.org/doc/stable/reference/generated/numpy.zeros_like.html

41. Cv2.circle(). Дата звернення: 6 черв. 2026. [Електронний ресурс]. Доступно: https://www.geeksforgeeks.org/python/python-opencv-cv2-circle-method/

42. Bitwise operations. Дата звернення: 7 черв. 2026. [Електронний ресурс]. Доступно: https://www.geeksforgeeks.org/python/arithmetic-operations-on-images-using-opencv-set-2-bitwise-operations-on-binary-images/

43. Cv2.contours(). Дата звернення: 6 черв. 2026. [Електронний ресурс]. Доступно: https://www.geeksforgeeks.org/python/find-and-draw-contours-using-opencv-python/

44. Cv2. minAreaRect() Дата звернення: 7 черв. 2026. [Електронний ресурс]. Доступно: https://www.geeksforgeeks.org/python/finding-minimum-enclosing-rectangle-in-opencv-python/

45. Computer Vision Lab. OPENCV & C++ TUTORIALS - 55 | contourArea() | Contour Area. (21 верес. 2022). Дата звернення: 6 черв. 2026. [Онлайн-відео]. Доступно: https://www.youtube.com/watch?v=Ar7Gyq8S8W4

46. Cv2.moments() Дата звернення: 7 черв. 2026. [Електронний ресурс]. Доступно: https://www.tutorialspoint.com/article/how-to-compute-image-moments-in-opencv-python

47. Дата звернення: 7 черв. 2026. [Електронний ресурс]. Доступно: https://www.blueskyrobotics.ai/product-page/ufactory-lite-6

48. Дата звернення: 7 черв. 2026. [Електронний ресурс]. Доступно: https://www.blueskyrobotics.ai/post/collaborative-robotic-arm-cost-breakdown-by-payload-class-2026

49. Дата звернення: 7 черв. 2026. [Електронний ресурс]. Доступно: https://www.smcusa.com/products/zpt-u-c-d-b-ut-ct-suction-cups-vertical-entry~87419

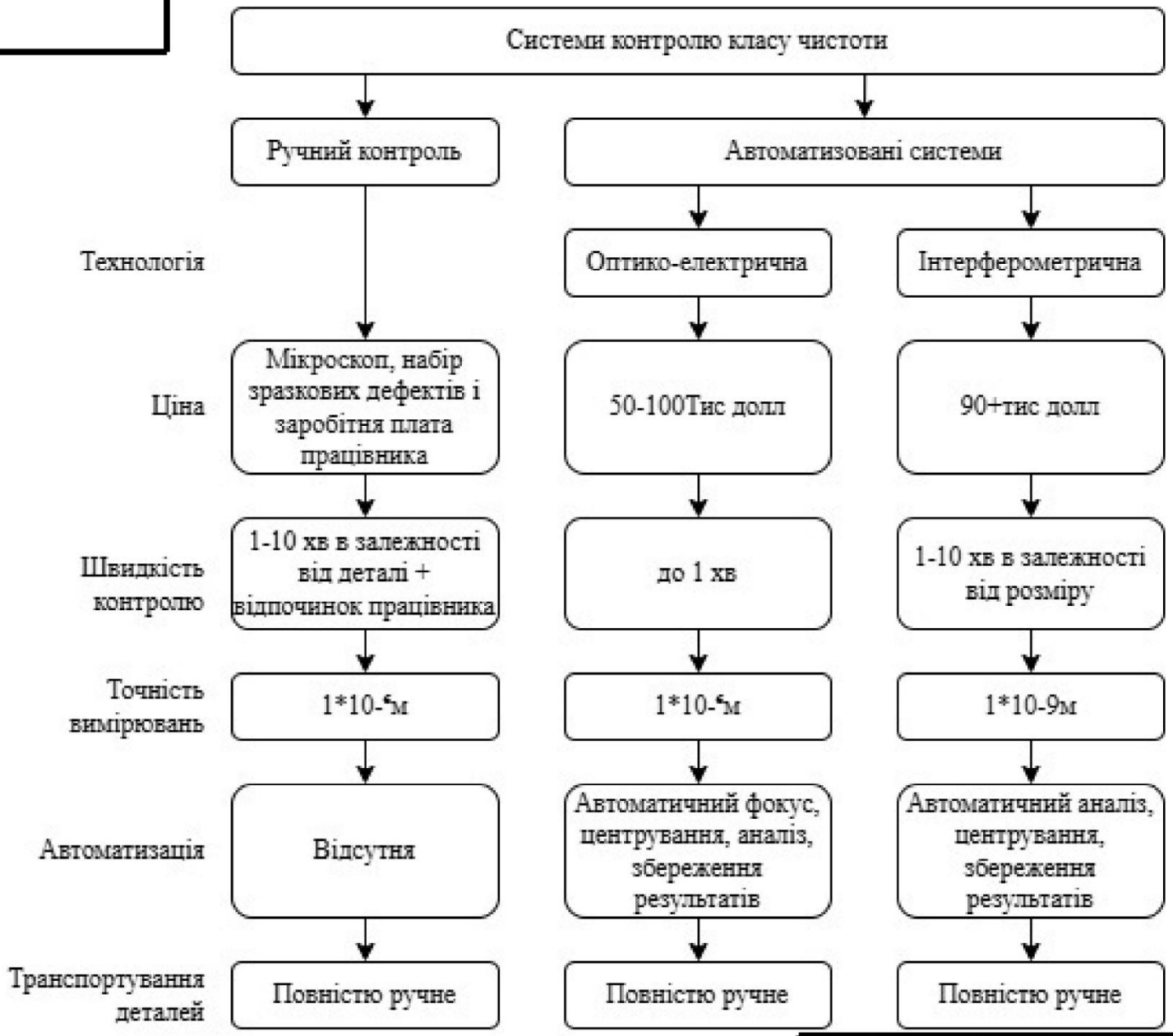
50. Дата звернення: 7 черв. 2026. [Електронний ресурс]. Доступно: https://www.smcusa.com/products/zxp7-01-vacuum-unit-for-universal-robots~167213

51. Дата звернення: 7 черв. 2026. [Електронний ресурс]. Доступно: https://gtm-ukraine.com.ua/porshneviy-pov-tryaniy-kompresor-50-l-gtm-kcw1500-50l-bezmaslyaniy.html

52. Дата звернення: 7 черв. 2026. [Електронний ресурс]. Доступно: https://trade-control.com.ua/ua/smc-aw20-f02ce-b

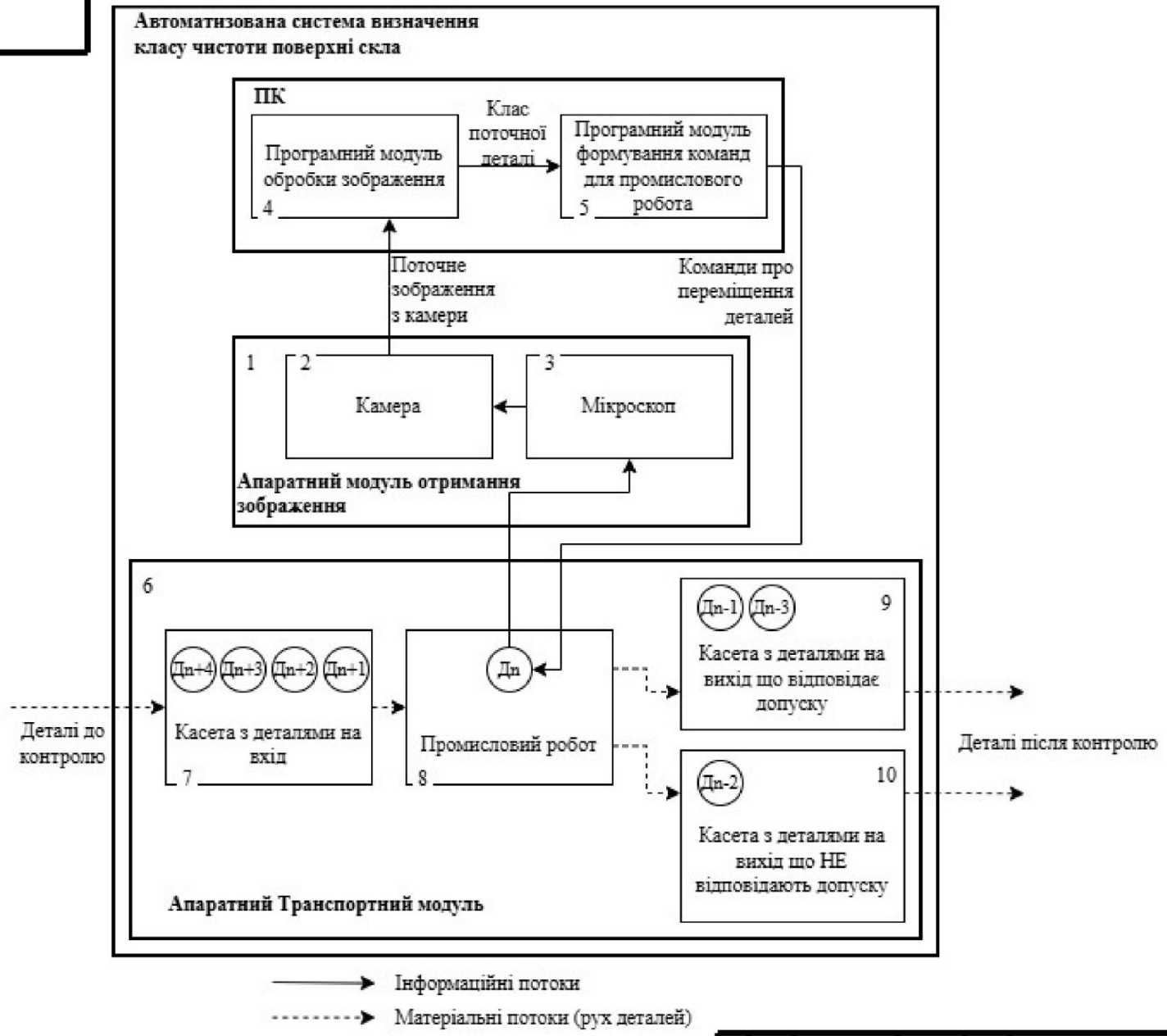
53. Дата звернення: 7 черв. 2026. [Електронний ресурс]. Доступно: https://pnevmosvit.com.ua/trubka-poliuretanova-tbpu-64-mm-prozora-tbpu

ДОДАТОК А



Зм.	Арк.	№ докум.	Підпис	Дата	ДР ПБ–з21.01.1720.001.СХ		
Розроб.		Александров О.О.			Порівняльня характеристика методів контролю поверхні	Літ.	Формат: А1
Перев.		Александров О.О.				Аркуш 1	Аркушів 1
Т. контр.		Александров О.О.				КПІ, ФРП, 4к	
Н. контр.		Александров О.О.					
Затв.		Тимчик Г.С.					

Перв. засл.
Справ. №
Підп. і дата
Інв. № дубл.
Взам. інв. №
Підп. і дата
Інв. № підл.



Зм.	Арк.	№ докум.	Підпис	Дата
Розроб.		Александров О.О.		
Перев.		Александров О.О.		
Т. контр.		Александров О.О.		
Н. контр.		Александров О.О.		
Затв.		Тимчик Г.С.		

ДР ПБ-з21.01.1720.002.СхС

Структурна схема системи

Літ.	Формат	Масштаб
	А1	
Аркуш 1	Аркушів 1	

КПІ, ФРП, 4к

Перв. заст.

Справ. №

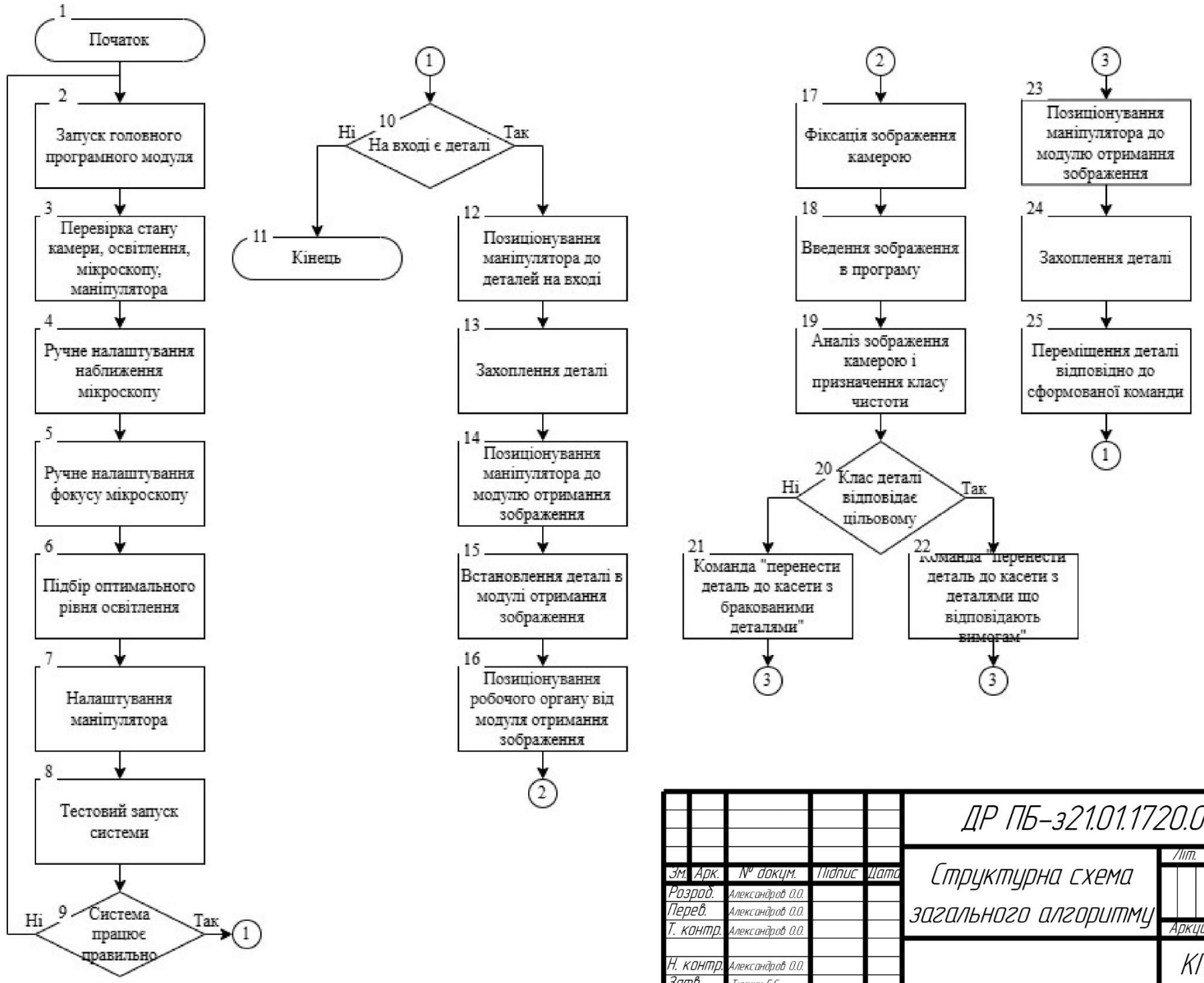
Підп. і дата

Інв. № дубл.

Взам. інв. №

Підп. і дата

Інв. № підл.



Зм.	Арк.	№ докум.	Підпис	Дата
Розроб.		Александров О.О.		
Перев.		Александров О.О.		
Т. контр.		Александров О.О.		
Н. контр.		Александров О.О.		
Затв.		Тимчик Г.С.		

ДР ПБ–з21.01.1720.003.СхА

Структурна схема загального алгоритму

Літ.	Формат	Масштаб
	А1	

Аркуш 1 | Аркушів 1

КПІ, ФРП, 4к

Перв. заст. | Справ. № | Підп. і дата | Інв. № дубл. | Взам. інв. № | Підп. і дата | Інв. № підл.

Зм.	Арк.	№ докум.	Підпис	Дата
Розроб.		Александров О.О.		
Перев.		Александров О.О.		
Т. контр.		Александров О.О.		
Н. контр.		Александров О.О.		
Затв.		Тимчик Г.С.		

ДР ПБ-з21.01.1720.004.СхА

Узагальнена блок-схема програмного забезпечення

Літ.	Формат	Масштаб
	А1	
Аркуш 1	Аркушів 1	

КПІ, ФРП, 4к

Перв. заст.

Справ. №

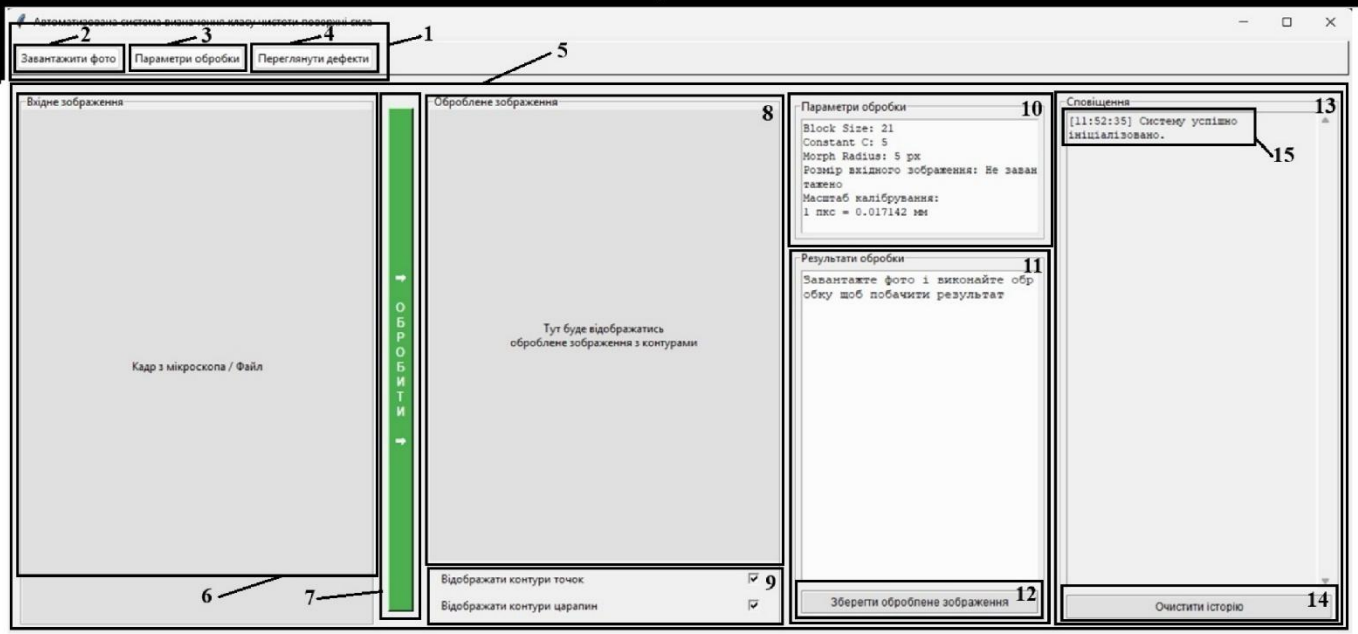
Підп. і дата

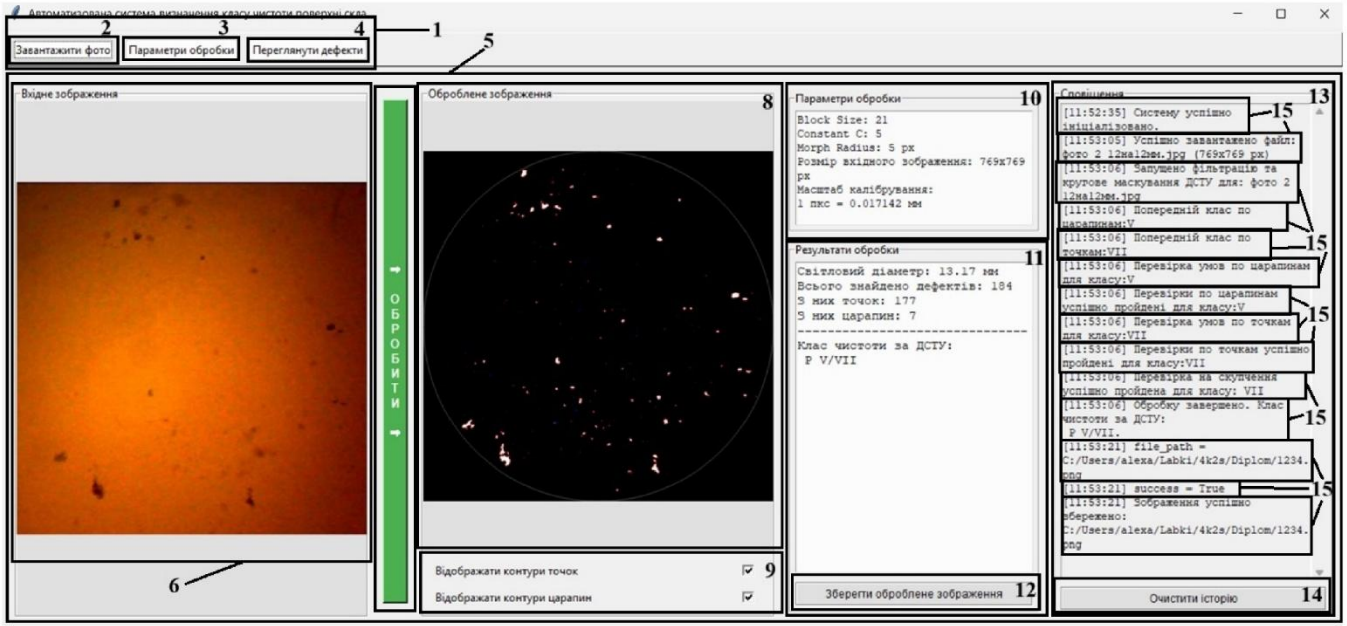
Інв. № дубл.

Взам. інв. №

Підп. і дата

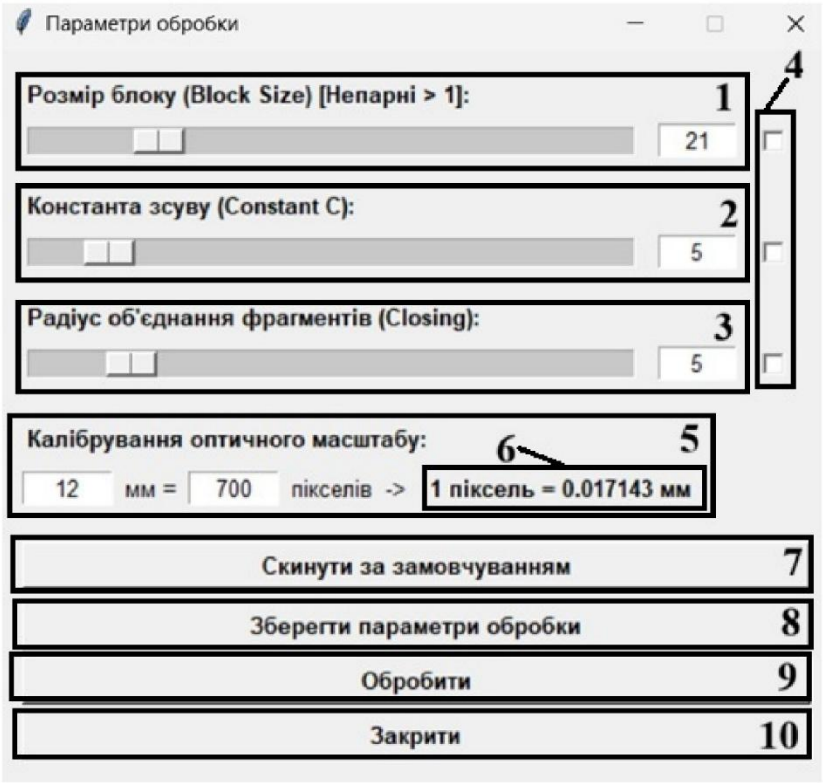
Інв. № підл.

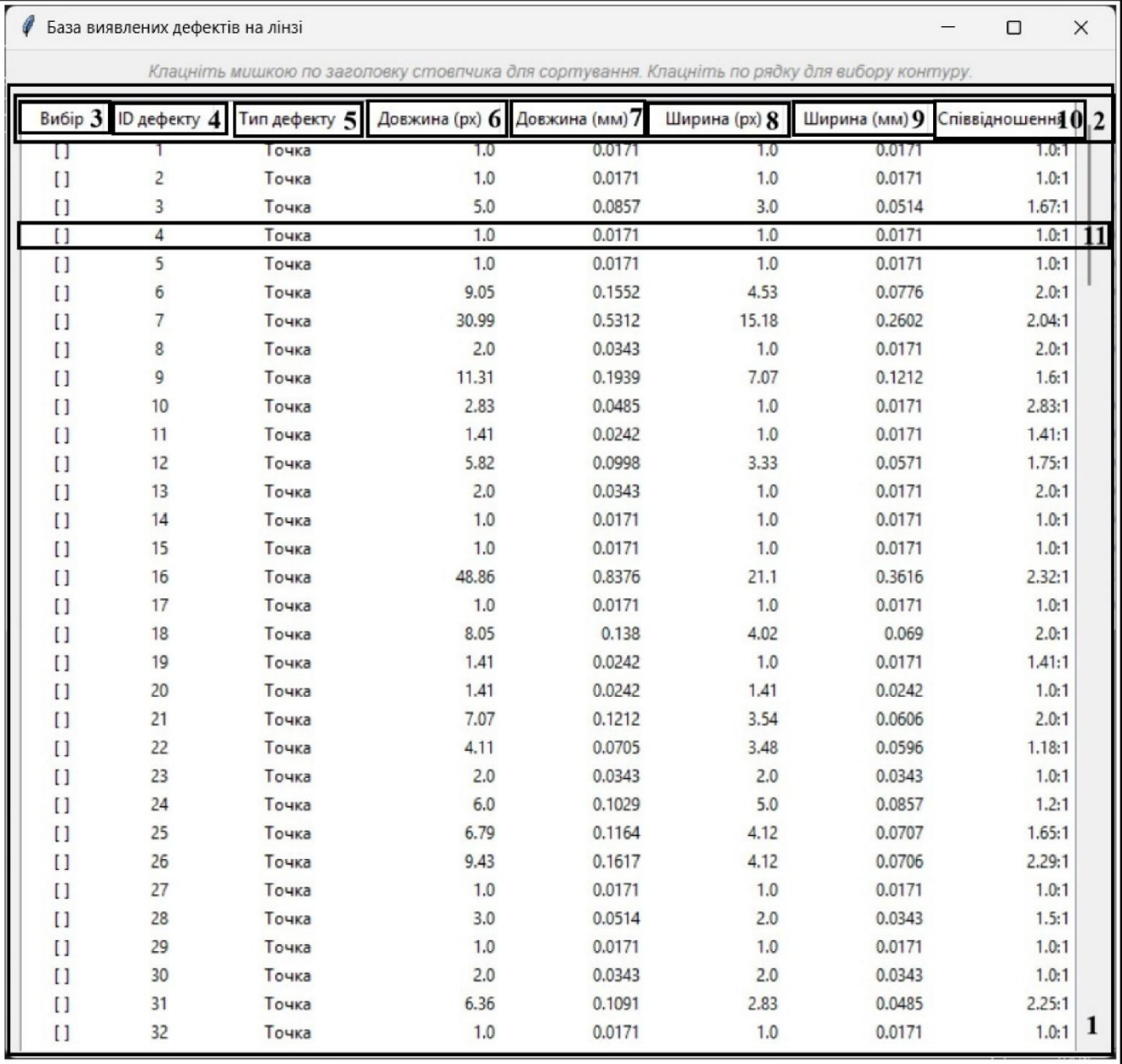




Зм.	Арк.	№ докум.	Підпис	Дата	ДР ПБ-з21.01.1720.005.01.СхА		
Розроб.		Александров О.О.			Інтерфейс головного вікна програми	Літ.	Формат / Масштаб
Перев.		Александров О.О.					А1
Т. контр.		Александров О.О.				Аркуш 1	Аркушів 1
Н. контр.		Александров О.О.				КПІ, ФРП, 4к	
Затв.		Тимчик Г.С.					

Перв. застос.
Справ. №
Підп. і дата
Інв. № дубл.
Взам. інв. №
Підп. і дата
Інв. № підл.





Вибір	ID дефекту	Тип дефекту	Довжина (px)	Довжина (мм)	Ширина (px)	Ширина (мм)	Співвідношення
[ ]	1	Точка	1.0	0.0171	1.0	0.0171	1.0:1
[ ]	2	Точка	1.0	0.0171	1.0	0.0171	1.0:1
[ ]	3	Точка	5.0	0.0857	3.0	0.0514	1.67:1
[ ]	4	Точка	1.0	0.0171	1.0	0.0171	1.0:1
[ ]	5	Точка	1.0	0.0171	1.0	0.0171	1.0:1
[ ]	6	Точка	9.05	0.1552	4.53	0.0776	2.0:1
[ ]	7	Точка	30.99	0.5312	15.18	0.2602	2.04:1
[ ]	8	Точка	2.0	0.0343	1.0	0.0171	2.0:1
[ ]	9	Точка	11.31	0.1939	7.07	0.1212	1.6:1
[ ]	10	Точка	2.83	0.0485	1.0	0.0171	2.83:1
[ ]	11	Точка	1.41	0.0242	1.0	0.0171	1.41:1
[ ]	12	Точка	5.82	0.0998	3.33	0.0571	1.75:1
[ ]	13	Точка	2.0	0.0343	1.0	0.0171	2.0:1
[ ]	14	Точка	1.0	0.0171	1.0	0.0171	1.0:1
[ ]	15	Точка	1.0	0.0171	1.0	0.0171	1.0:1
[ ]	16	Точка	48.86	0.8376	21.1	0.3616	2.32:1
[ ]	17	Точка	1.0	0.0171	1.0	0.0171	1.0:1
[ ]	18	Точка	8.05	0.138	4.02	0.069	2.0:1
[ ]	19	Точка	1.41	0.0242	1.0	0.0171	1.41:1
[ ]	20	Точка	1.41	0.0242	1.41	0.0242	1.0:1
[ ]	21	Точка	7.07	0.1212	3.54	0.0606	2.0:1
[ ]	22	Точка	4.11	0.0705	3.48	0.0596	1.18:1
[ ]	23	Точка	2.0	0.0343	2.0	0.0343	1.0:1
[ ]	24	Точка	6.0	0.1029	5.0	0.0857	1.2:1
[ ]	25	Точка	6.79	0.1164	4.12	0.0707	1.65:1
[ ]	26	Точка	9.43	0.1617	4.12	0.0706	2.29:1
[ ]	27	Точка	1.0	0.0171	1.0	0.0171	1.0:1
[ ]	28	Точка	3.0	0.0514	2.0	0.0343	1.5:1
[ ]	29	Точка	1.0	0.0171	1.0	0.0171	1.0:1
[ ]	30	Точка	2.0	0.0343	2.0	0.0343	1.0:1
[ ]	31	Точка	6.36	0.1091	2.83	0.0485	2.25:1
[ ]	32	Точка	1.0	0.0171	1.0	0.0171	1.0:1

Перв. заст.
Справ. №
Підп. і дата
Інв. № дубл.
Взам. інв. №
Підп. і дата
Інв. № підл.

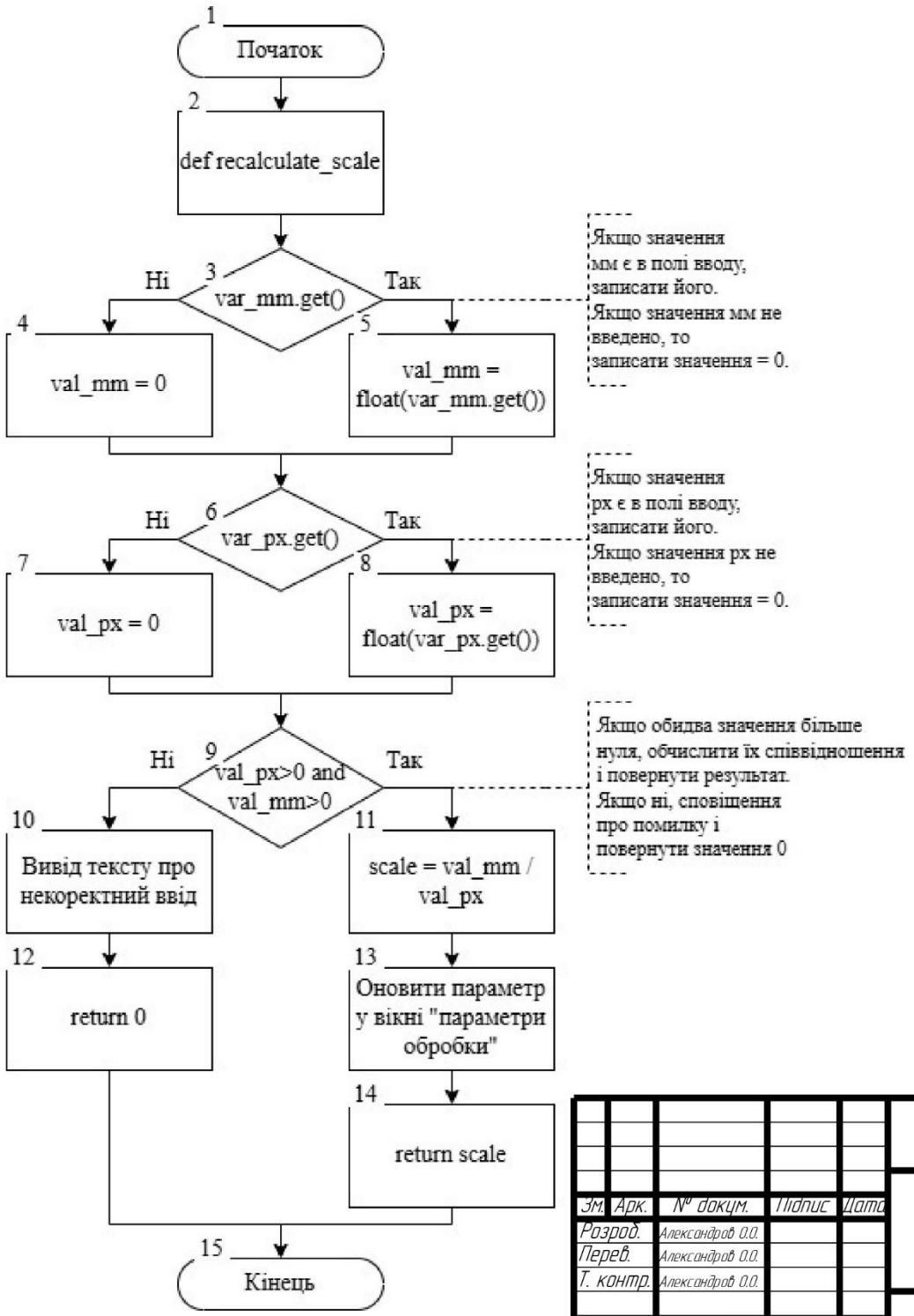
Зм.	Арк.	№ докум.	Підпис	Дата
Розроб.		Александров О.О.		
Перев.		Александров О.О.		
Т. контр.		Александров О.О.		
Н. контр.		Александров О.О.		
Затв.		Тимчик Г.С.		

ДР ПБ-з21.01.1720.005.02.СхА

Інтерфейс допоміжних вікон програми

Літ.	Формат	Масштаб
	А1	
Аркуш 1	Аркушів 1	

КПІ, ФРП, 4к



Зм.	Арк.	№ докум.	Підпис	Дата
Розроб.		Александров О.О.		
Перев.		Александров О.О.		
Т. контр.		Александров О.О.		
Н. контр.		Александров О.О.		
Затв.		Тимчик Г.С.		

ДР ПБ-з21.01.1720.006.СхА

Блок-схема алгоритму функції recalculate_scale()

Літ.	Формат	Масштаб
	А1	

Аркуш 1	Аркушів 1

КПІ, ФРП, 4к

Перв. застос.

Справ. №

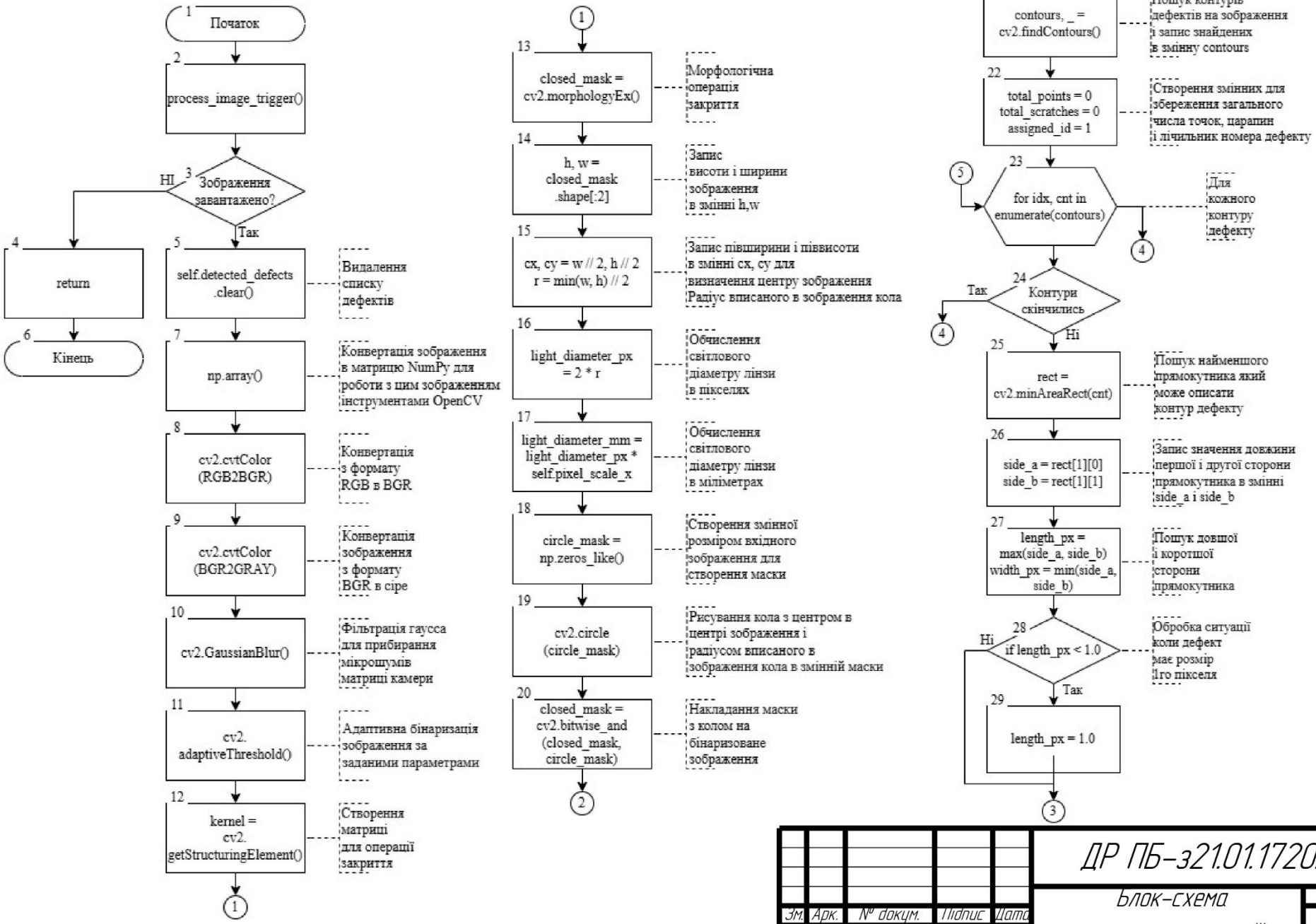
Підп. і дата

Інв. № дубл.

Взам. інв. №

Підп. і дата

Інв. № підл.



Зм.	Арк.	№ докум.	Підпис	Дата
Розроб.		Александров О.О.		
Перев.		Александров О.О.		
Т. контр.		Александров О.О.		
Н. контр.		Александров О.О.		
Затв.		Тимчик Г.С.		

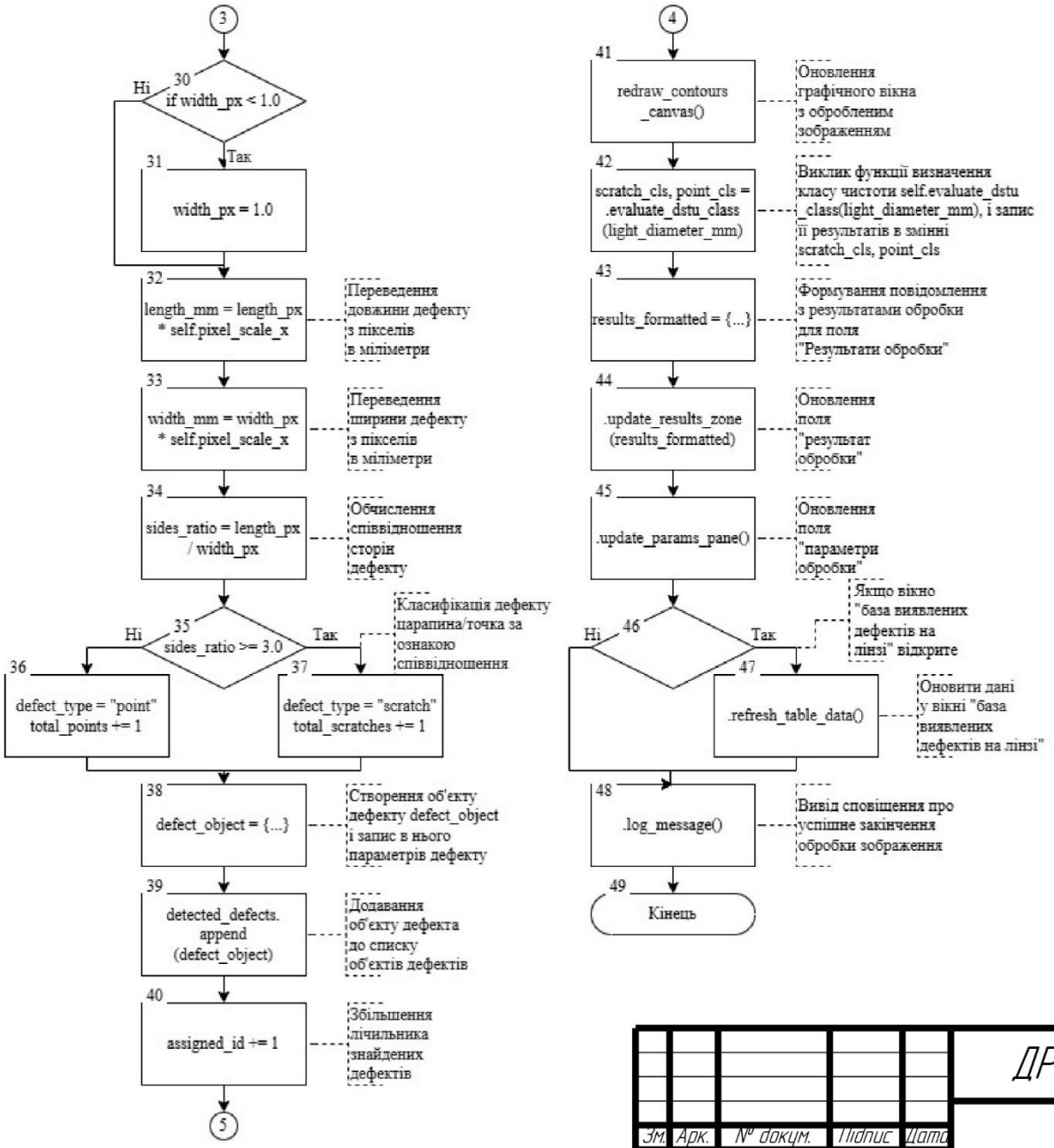
ДР ПБ-з21.01.1720.007.01.СхА

Блок-схема алгоритму функції process_image_trigger()

Літ.	Формат	Масштаб
	А1	
Аркуш 1	Аркушів 1	

КПІ, ФРП, 4к

Перв. застос.
Справ. №
Підп. і дата
Інв. № дубл.
Взам. інв. №
Підп. і дата
Інв. № підл.



Зм.	Арк.	№ докум.	Підпис	Дата
Розроб.		Александров О.О.		
Перев.		Александров О.О.		
Т. контр.		Александров О.О.		
Н. контр.		Александров О.О.		
Затв.		Тимчик Г.С.		

ДР ПБ-з21.01.1720.007.02.СхА

Блок-схема алгоритму функції process_image_trigger()

Літ.	Формат	Масштаб
	А1	
Аркуш 1	Аркушів 1	

КПІ, ФРП, 4к

Перв. заст.

Справ. №

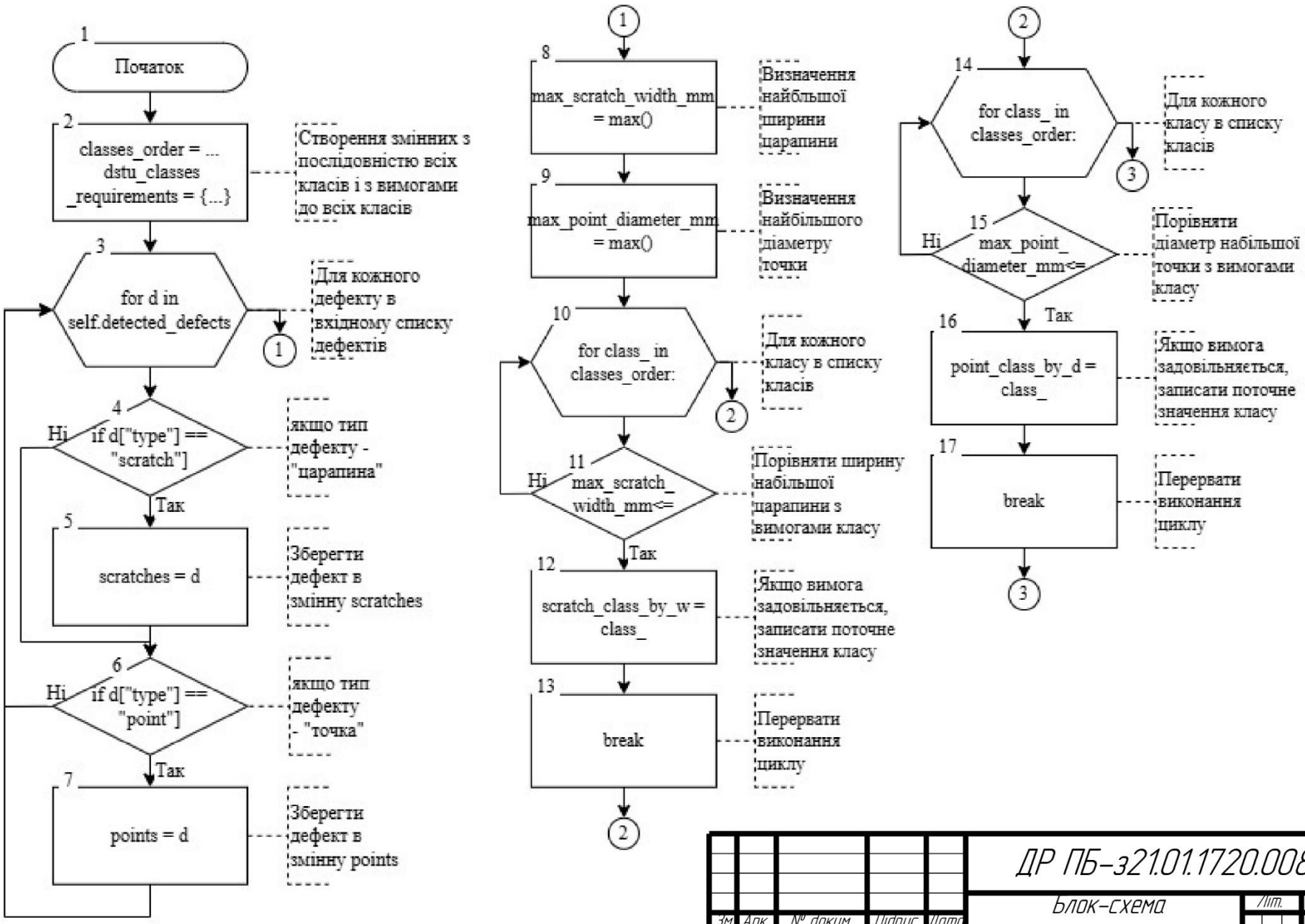
Підп. і дата

Інв. № дубл.

Взам. інв. №

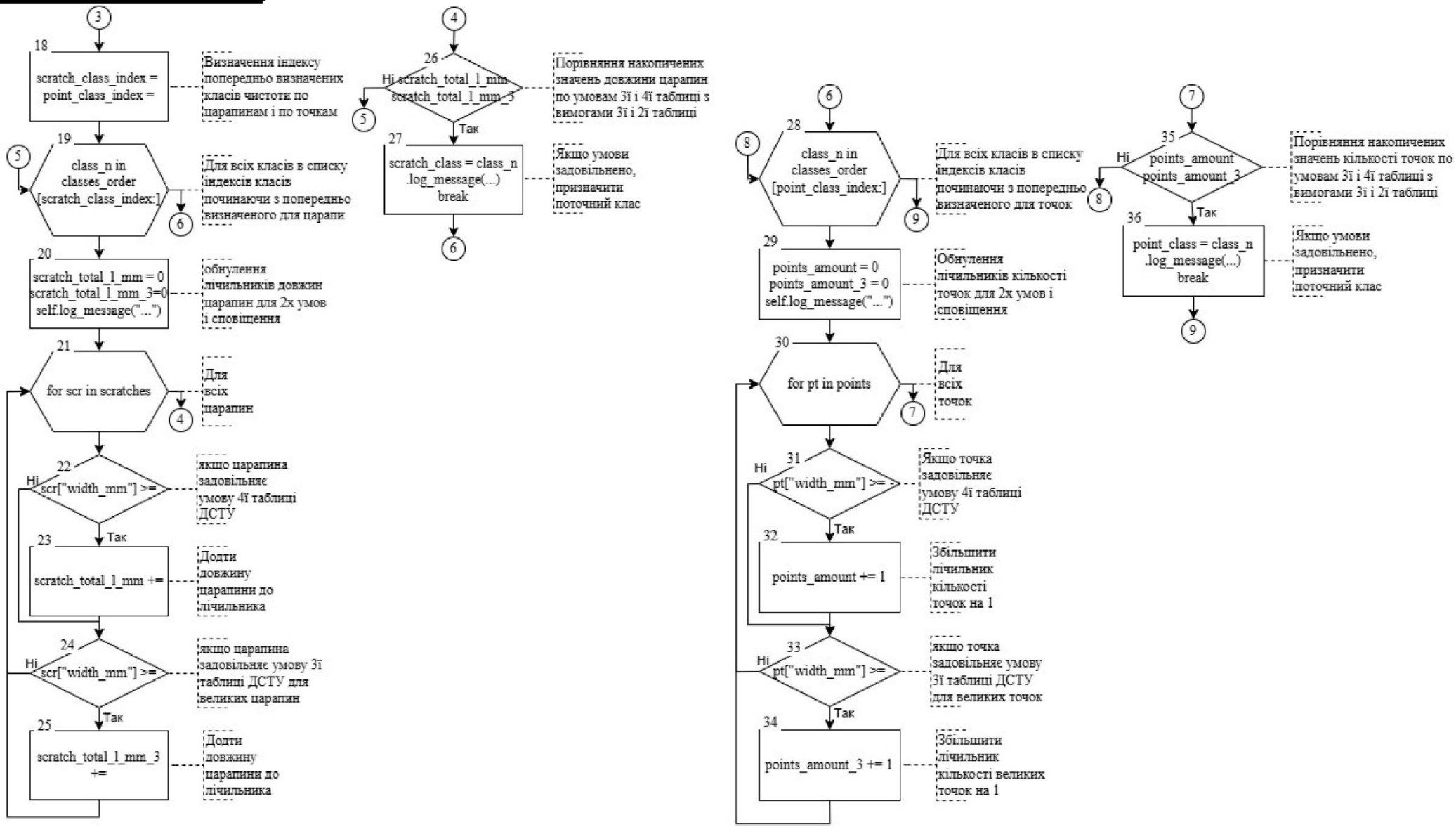
Підп. і дата

Інв. № підл.



Зм.	Арк.	№ докум.	Підпис	Дата	ДР ПБ-з21.01.1720.008.01.СхА		
Розроб.		Александров О.О.			Блок-схема алгоритму функції evaluate_dstu_class()	Літ.	Формат A1 / Масштаб
Перев.		Александров О.О.				Аркуш 1	Аркушів 1
Т. контр.		Александров О.О.				КПІ, ФРП, 4к	
Н. контр.		Александров О.О.					
Затв.		Тимчик Г.С.					

Перв. застос. | Довід. № | Підп. і дата | Інв. № дубл. | Взам. інв. № | Підп. і дата | Інв. № підл.



Зм.	Арк.	№ докум.	Підпис	Дата
Розроб.		Александров О.О.		
Перев.		Александров О.О.		
Т. контр.		Александров О.О.		
Н. контр.		Александров О.О.		
Затв.		Тимчик Г.С.		

ДР ПБ-з21.01.1720.008.02.СхА

Блок-схема алгоритму функції evaluate_dstu_class()

Літ.	Формат	Масштаб
	А1	

Аркуш 1 | Аркушів 1

КПІ, ФРП, 4к

Перв. заст.

Справ. №

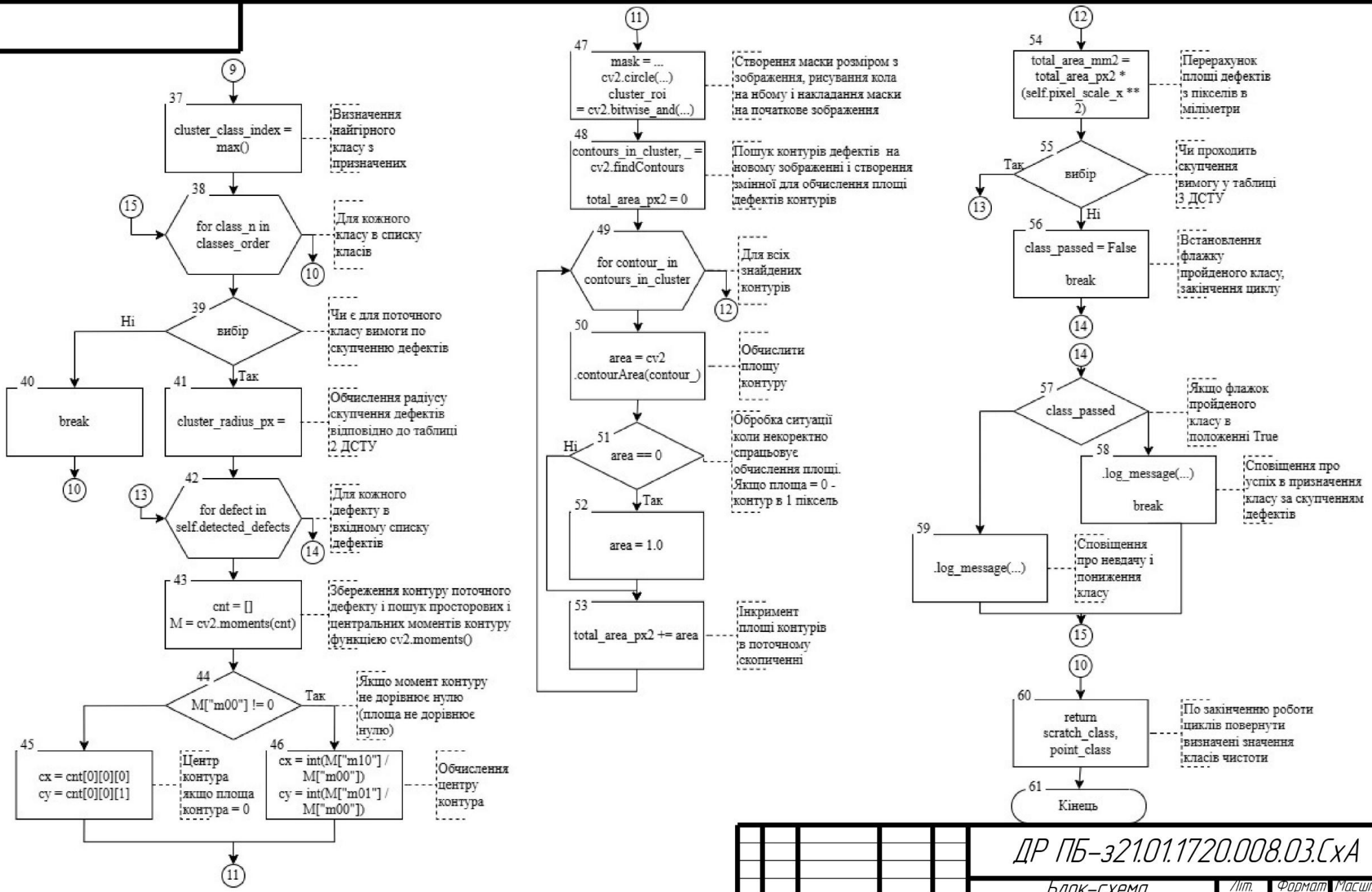
Підп. і дата

Інв. № дубл.

Взам. інв. №

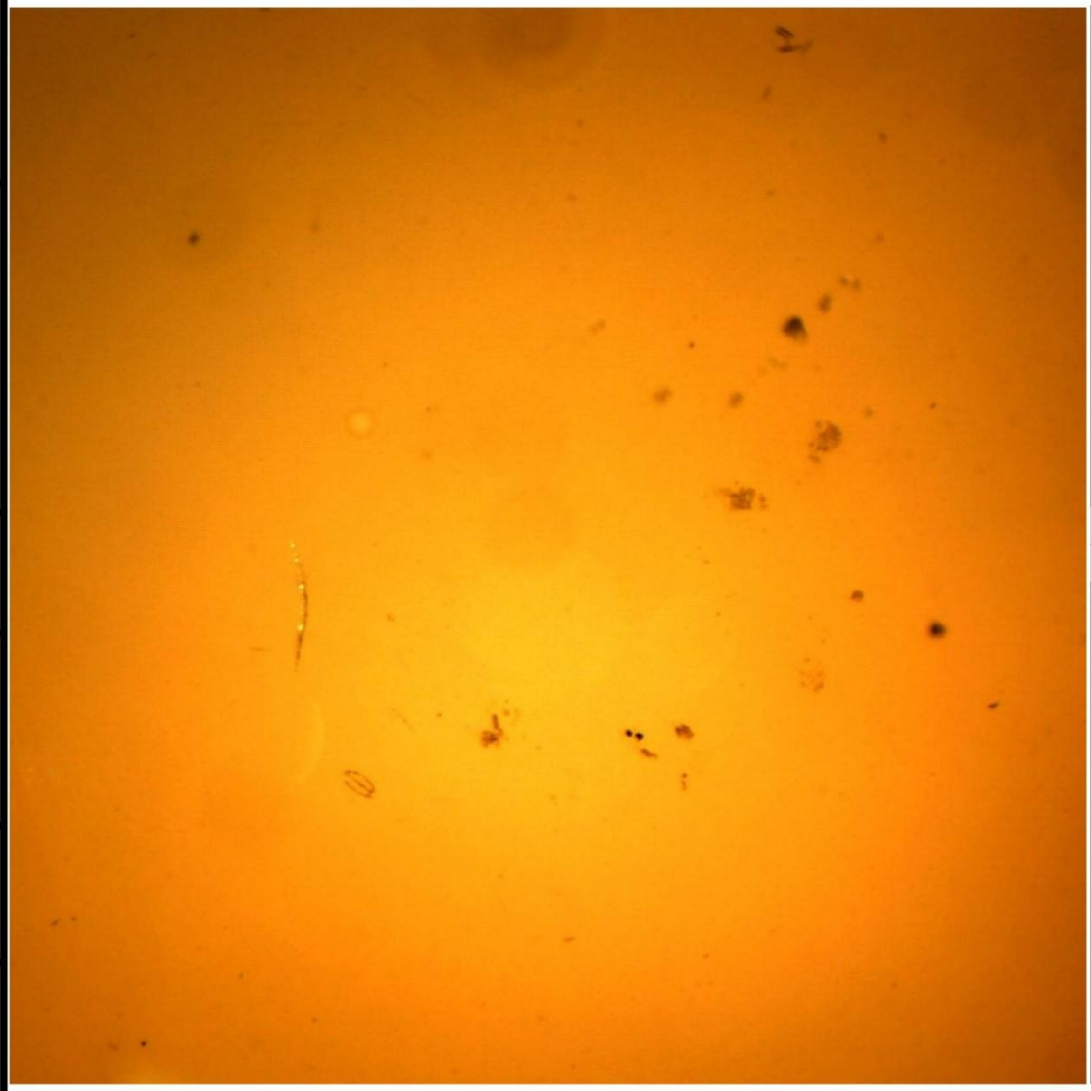
Підп. і дата

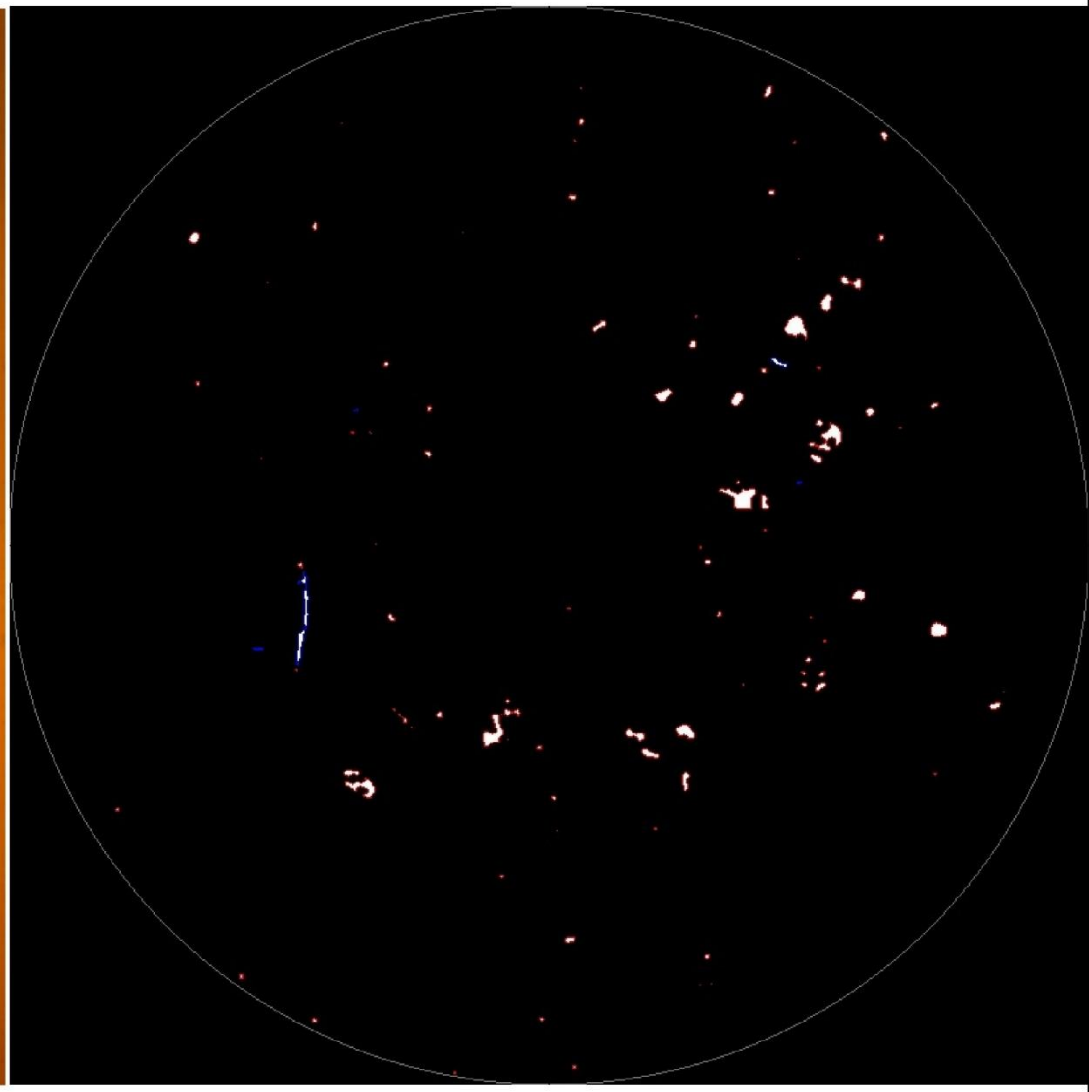
Інв. № підл.



Зм.	Арк.	№ докум.	Підпис	Дата	ДР ПБ-з21.01.1720.008.03.СхА		
Розроб.		Александров О.О.			Блок-схема алгоритму функції evaluate_dstu_class()	Літ.	Формат A1 / Масштаб
Перев.		Александров О.О.				Аркуш 1	Аркушів 1
Т. контр.		Александров О.О.				КПІ, ФРП, 4к	
Н. контр.		Александров О.О.					
Затв.		Тимчик Г.С.					

Перв. застос. | Справ. № | Підп. і дата | Інв. № дубл. | Взам. інв. № | Підп. і дата | Інв. № підл.





Зм.	Арк.	№ докум.	Підпис	Дата
Розроб.		Александров О.О.		
Перев.		Александров О.О.		
Т. контр.		Александров О.О.		
Н. контр.		Александров О.О.		
Затв.		Тимчик Г.С.		

ДР ПБ-з21.01.1720.009.01.СХ

Зображення 1 до/після обробки

Літ.	Формат	Масштаб
	А1	
Аркуш 1	Аркушів 1	

КПІ, ФРП, 4к

Перв. застос.

Справ. №

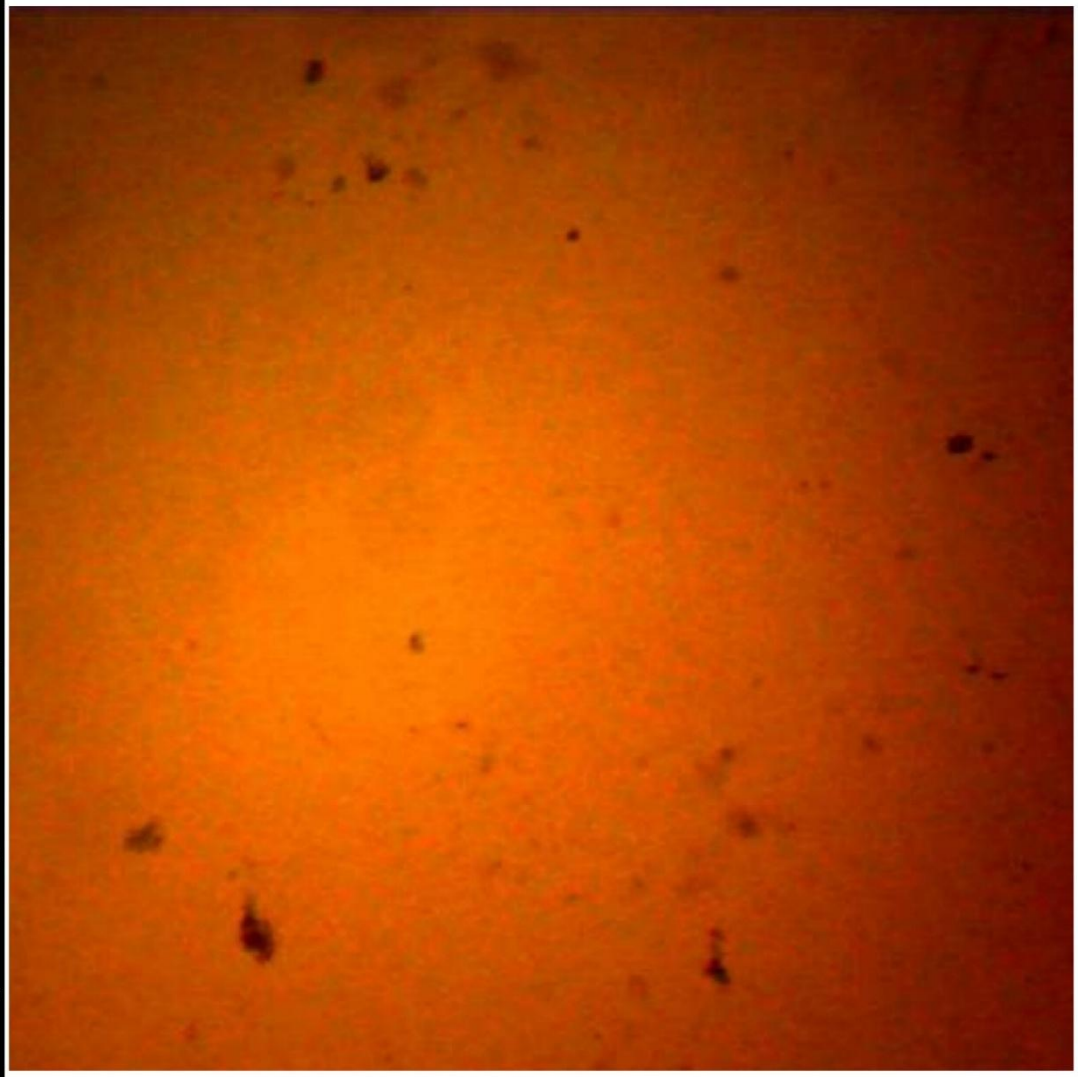
Підп. і дата

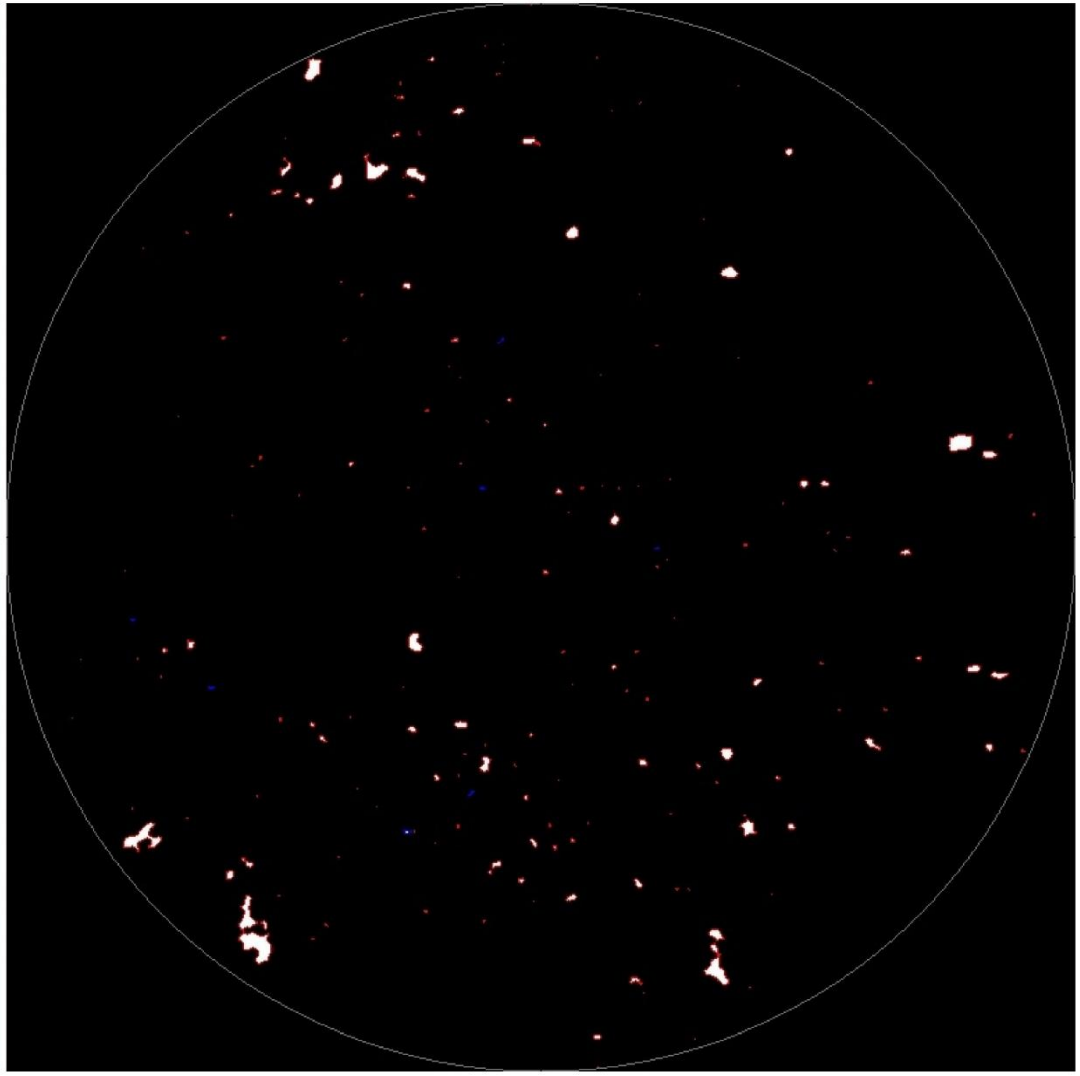
Інв. № дубл.

Взам. інв. №

Підп. і дата

Інв. № підл.





					ДР ПБ-з21.01.1720.009.02.СХ		
Зм.	Арк.	№ докум.	Підпис	Дата	Зображення 2 до/після обробки	Літ.	Формат / Масштаб
Розроб.		Александров О.О.					А1
Перев.		Александров О.О.				Аркуш 1	Аркушів 1
Т. контр.		Александров О.О.					
Н. контр.		Александров О.О.				КПІ, ФРП, 4к	
Затв.		Тимчик Г.С.					

Перев. засп.

Справ. №

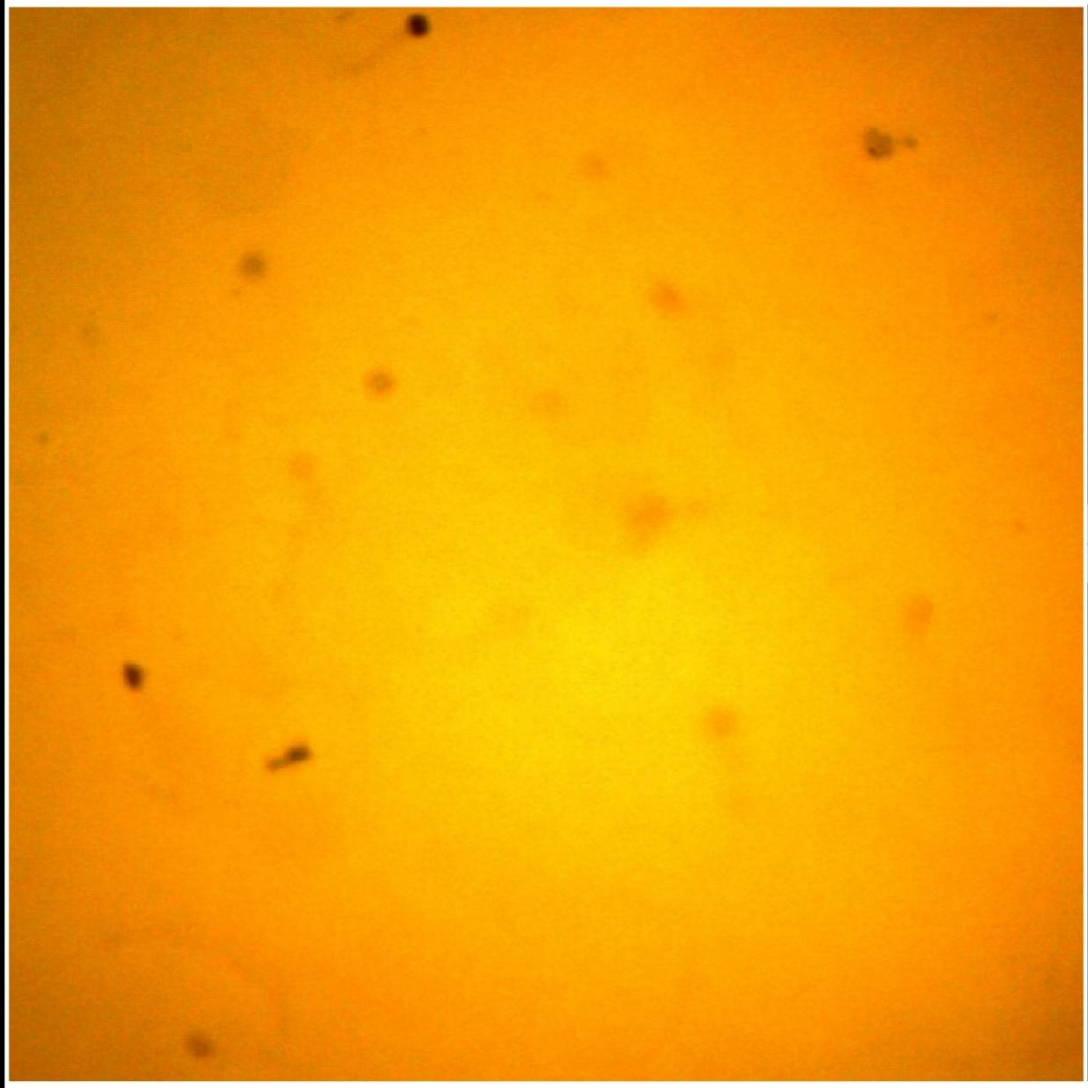
Підп. і дата

Інв. № дубл.

Взам. інв. №

Підп. і дата

Інв. № підл.



Зм.	Арк.	№ докум.	Підпис	Дата
Розроб.		Александров О.О.		
Перев.		Александров О.О.		
Т. контр.		Александров О.О.		
Н. контр.		Александров О.О.		
Затв.		Тимчик Г.С.		

ДР ПБ-з21.01.1720.009.03.СХ

Зображення 3 до/після обробки

Літ.	Формат	Масштаб
	А1	
Аркуш 1	Аркушів 1	

КПІ, ФРП, 4к

Перед. засл.

Справ. №

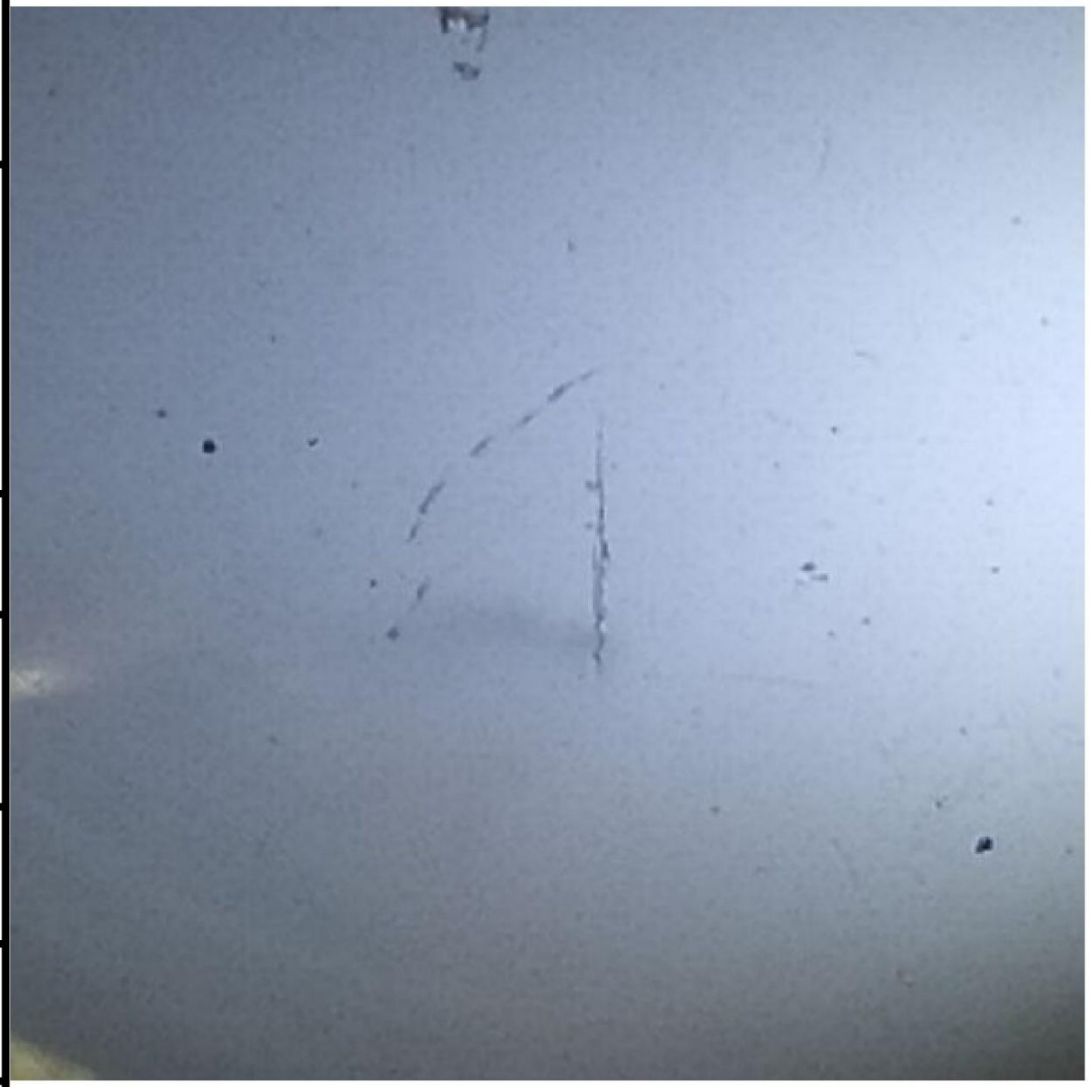
Підп. і дата

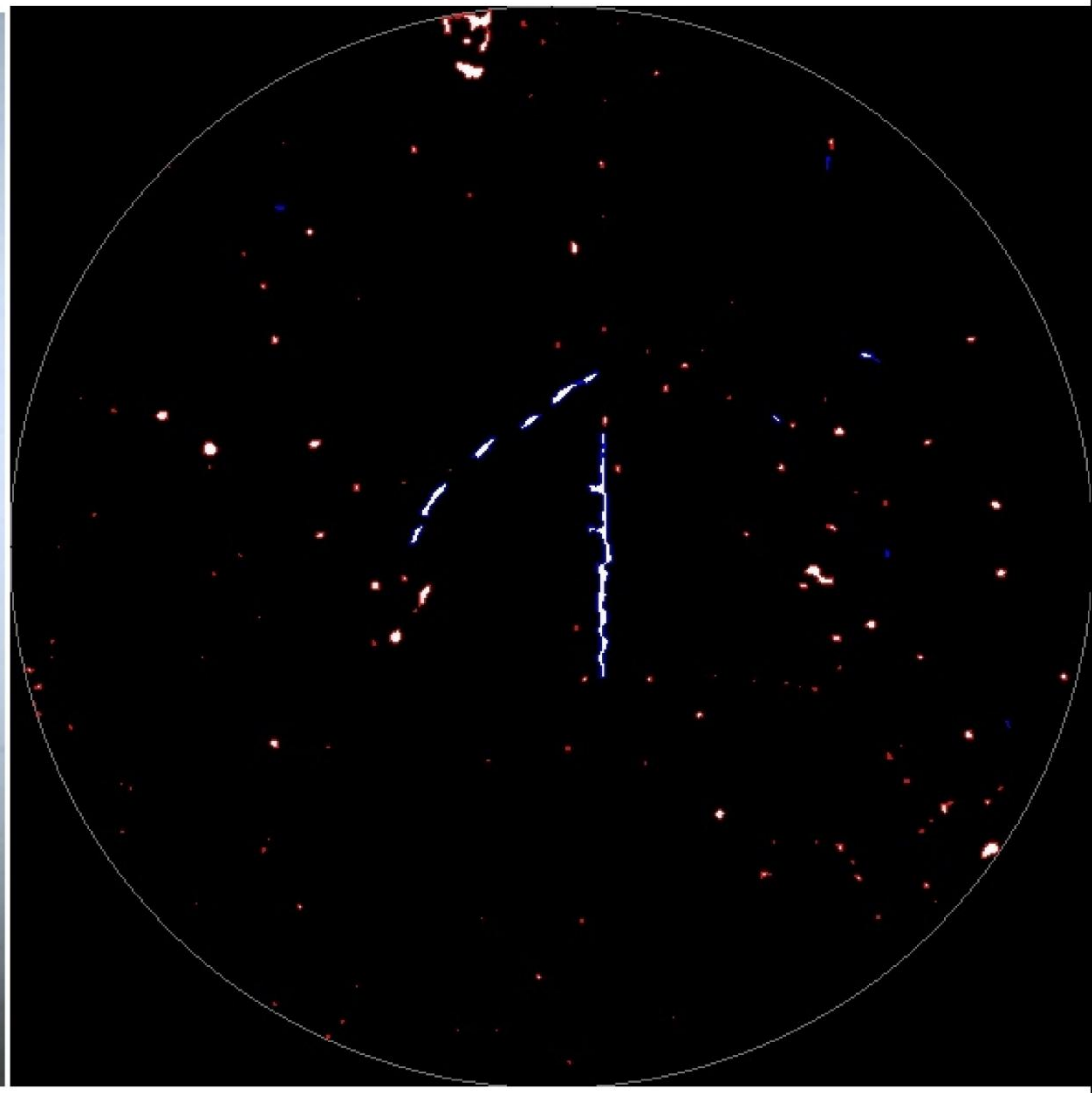
Інв. № дубл.

Взам. інв. №

Підп. і дата

Інв. № підл.





					ДР ПБ-з21.01.1720.009.04.СХ		
Зм.	Арк.	№ докум.	Підпис	Дата	Зображення 4 до/після обробки	Літ.	Формат / Масштаб
Розроб.		Александров О.О.					А1
Перев.		Александров О.О.				Аркуш 1	Аркушів 1
Т. контр.		Александров О.О.					
Н. контр.		Александров О.О.				КПІ, ФРП, 4к	
Затв.		Тимчик Г.С.					

Перв. застос.

Справ. №

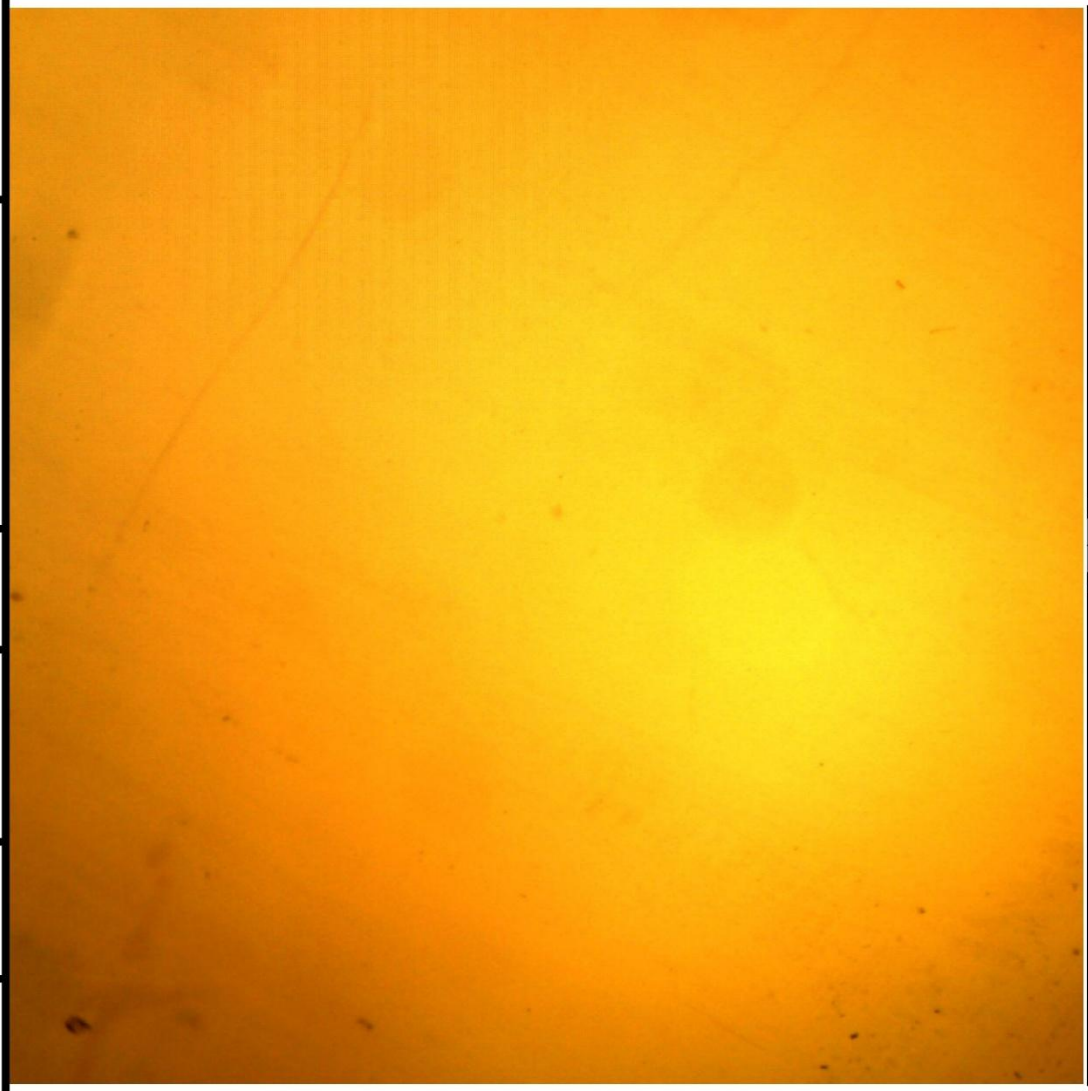
Підп. і дата

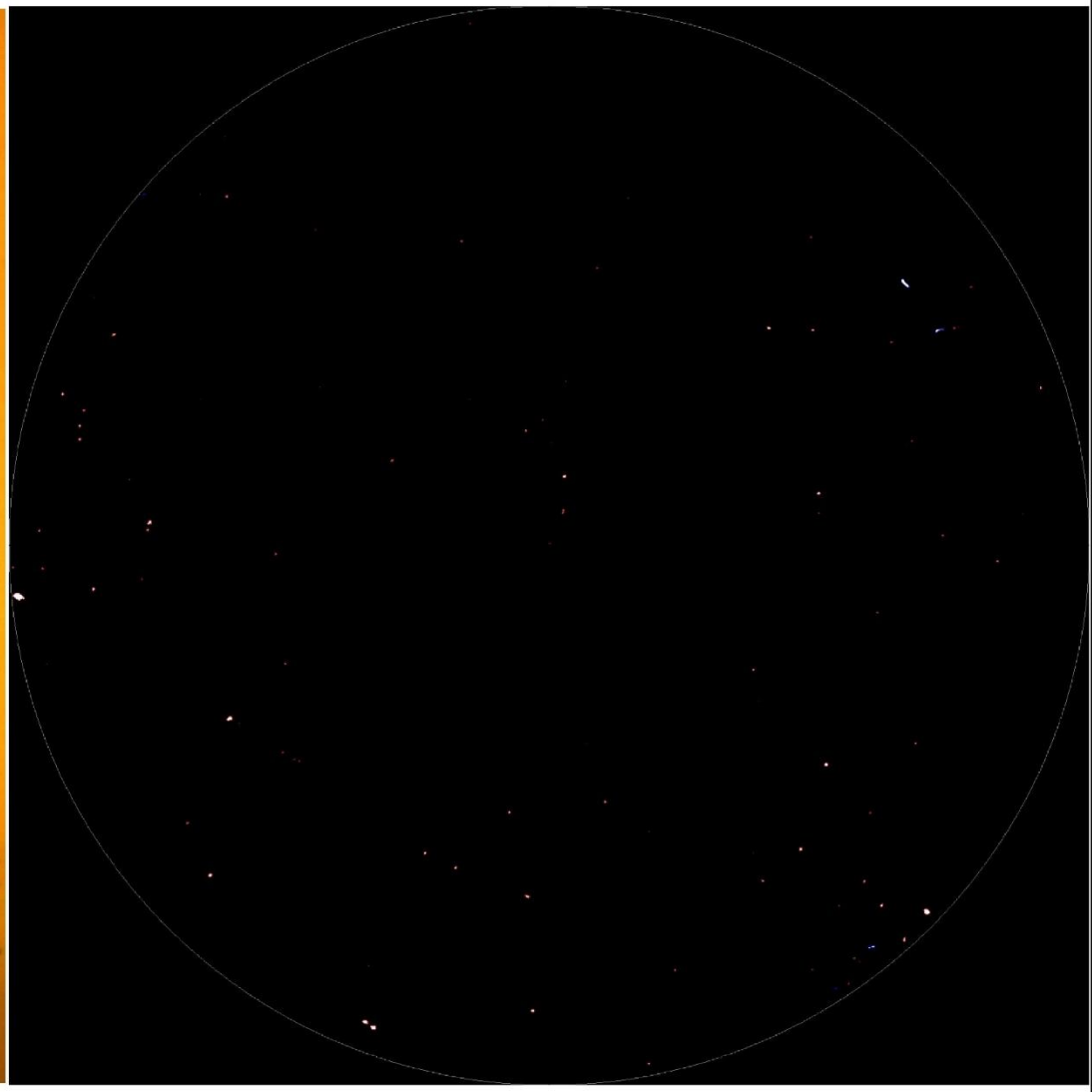
Інв. № дубл.

Взам. інв. №

Підп. і дата

Інв. № підл.





					ДР ПБ-з21.01.1720.009.05.СХ		
Зм.	Арк.	№ докум.	Підпис	Дата	Зображення 5 до/після обробки	Літ.	Формат / Масштаб
Розроб.		Александров О.О.					А1
Перев.		Александров О.О.				Аркуш 1	Аркушів 1
Т. контр.		Александров О.О.				КПІ, ФРП, 4к	
Н. контр.		Александров О.О.					
Затв.		Тимчик Г.С.					

Перв. застос.

Довід. №

Підп. і дата

Інв. № дубл.

Взам. інв. №

Підп. і дата

Інв. № підл.

ДОДАТОК Б

```
import cv2                          #імпорт бібліотеки OpenCV для комп'ютерного зору
та обробки зображень
import numpy as np                  #імпорт бібліотеки numpy з ім'ям np для реалізації
операцій з зображеннями
from PIL import Image, ImageTk      #імпорт бібліотеки Pillow для роботи із
зображеннями
import tkinter as tk                #імпорт бібліотеки tkinter з ім'ям tk для
створення графічного інтерфейсу
from datetime import datetime       #модуль для отримання поточного часу
from tkinter import ttk             #модуль для елементів інтерфейсу (таблиця
дефектів)
from tkinter import filedialog      #модуль для реалізації вибору файлу фото


#головний клас програми що описує всю роботу програми, тобто графічний інтерфейс і
математику за ним
class GlassPurityApp:

    #ініціалізація головного вікна програми
    def __init__(self, root):                #системний метод __init__ створює
об'єкт додатку класу
        self.root = root                     #призначення змінній класу
GlassPurityApp.root значення самого себе, щоб можна було звертатись до себе
        self.root.title("Автоматизована система визначення класу чистоти поверхні
скла") #назва вікна програми
        self.root.geometry("1450x650")
      #початковий розмір вікна програми
        self.root.minsize(1350,
600)                                                  #мінімально допустимий
розмір вікна програми

        self.settings_win = None    #змінна для збереження посилання на додаткове
вікно налаштувань, щоб уникнути дублювання
        self.table_win = None       #змінна задля збереження посилання на
додаткове вікно списку виявлених дефектів, щоб уникнути дублювання

        #елементи інтерфейсу таблиці списку виявлених дефектів
        self.tree = None
        self.table_container = None
        self.lbl_table_info = None
        self.lbl_empty_table = None

        #значення параметрів обробки за замовчуванням
        self.default_block_size = 21    #розмір сусідства пікселів для розрахунку
локального порогу бінаризації
        self.default_constant_c = 5     #константа, що віднімається від середнього
зваженого значення
        self.default_morph_size = 5     #розмір матриці для ерозій і дилатації

        #поточні значення параметрів обробки що можуть змінюватися користувачем і
повертатись до параметрів за замовчуванням
        self.current_block_size = self.default_block_size   #розмір блоку для
бінаризації
        self.current_constant_c = self.default_constant_c   #константа для
бінаризації
        self.current_morph_size = self.default_morph_size   #розмір матриці для
ерозій і дилатації

        #прапорці для блокування скидання параметрів обробки
        self.lock_block = tk.BooleanVar(value=False)
        self.lock_c = tk.BooleanVar(value=False)
        self.lock_morph = tk.BooleanVar(value=False)
```

```
        #прапорці для вимкнення/ввімкнення відображення точок/царапин на
обробленому зображенні
        self.show_points = tk.BooleanVar(value=True)
        self.show_scratches = tk.BooleanVar(value=True)

        #параметри калібрування оптичного масштабу
        self.calib_mm_val = "12"        #мм які дорівнюють пікселям
        self.calib_px_val = "700"       #пікселі які дорівнюють міліметрам
        self.pixel_scale_x = 0.017142   #обчислений коеф масштабу, а саме скільки
міліметрів міститься в 1 пікселі

        #змінні що стосуються зображень
        self.image_path = None          #шлях до файлу завантаженого зображення
        self.original_image = None      #об'єкт для оригінального зображення в
форматі PIL
        self.tk_input_image = None      #об'єкт вхідного зображення, адаптований
для відображення в Tkinter
        self.tk_output_image = None     #об'єкт обробленого зображення для
відображення в Tkinter
        self.img_w = 0                  #ширина завантаженого зображення
        self.img_h = 0                  #висота заватаженого зображення

        self.cached_thresh = None       #бінарна маска після адаптивного порогу та
фільтрації
        self.cached_contours = []       #список знайдених контурів дефектів
        self.detected_defects = []      #структурований список словників із
параметрами кожного дефекту зі списку знайдених контурів дефектів

        #розділ створення графічного інтерфейсу
        self.top_bar = tk.Frame(root, bg="#f0f0f0", bd=1, relief=tk.RAISED)
#виклик методу tk.Frame() для створення головного вікна програми з сірим кльором
фону #f0f0f0, товщиною рамки 1 і в стилі tk.RAISED
        self.top_bar.pack(side=tk.TOP, fill=tk.X, padx=5,
pady=5)           #менеджер геометрії .pack() малює панель для кнопок в головному
вікні програми по всій ширині з відступами по 5 пікселів по бокам і зверху

        #створення кнопки для відкриття файлу на верхній панелі з написом
"завантажити фото" і викликом методу self.load_image при натисканні
        self.btn_load = ttk.Button(self.top_bar, text="Завантажити фото",
command=self.load_image)
        self.btn_load.pack(side=tk.LEFT, padx=5,
pady=5)                     #розмістити кнопку з самого ліва

        #створення кнопки для відкриття вікна з параметрами обробки на верхній
панелі з написом "Параметри обробки" і викликом методу
self.open_processing_settings_menu при натисканні
        self.btn_proc_settings = ttk.Button(self.top_bar, text="Параметри
обробки", command=self.open_processing_settings_menu)
        self.btn_proc_settings.pack(side=tk.LEFT, padx=5,
pady=5)           #розмістити кнопку з самого ліва

        #створення кнопки для відкриття вікна з таблицею виявлених дефектів на
верхній панелі з написом "Параметри обробки" і викликом методу
self.open_defects_table_window при натисканні
        self.btn_view_defects = ttk.Button(self.top_bar, text="Переглянути
дефекти", command=self.open_defects_table_window)
        self.btn_view_defects.pack(side=tk.LEFT, padx=5,
pady=5)            #розмістити кнопку з самого ліва

        #створення контейнера робочої зони програми
        self.main_container =
tk.Frame(root)                                  #виклик методу tk.Frame() для
створення об'єкту основної робочої області що зберігається в змінну
self.main_container
```

```
        self.main_container.pack(fill=tk.BOTH, expand=True, padx=10,
pady=10)   #менеджер геометрії .pack() малює основну область з центруванням по
центру(fill=tk.BOTH) і ввімкненою можливістю розтягування(expand=True) з
відсутпами по 10 пікселів з усіх сторін

        #створення рамки за написом(tk.LabelFrame()) що розміщена в
self.main_container з написом "Вхідне зображення" і заданими розмірами
        self.frame_input = tk.LabelFrame(self.main_container, text="Вхідне
зображення", width=380, height=400)
        self.frame_input.pack_propagate(False)
         #вимкнення зміни розміру рамки
        self.frame_input.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=5,
pady=5)      #менеджер геометрії .pack() розміщує рамку зліва з центруванням
елементів в ній, з можливістю розтягування і відступами по 5 пікселів
        self.lbl_input_image = tk.Label(self.frame_input, text="Кадр з мікроскопа
/ Файл", bg="#e0e0e0")    #напис "Кадр з мікроскопа / Файл" з сірим воном що
відображається допоки не завантажене фото
        self.lbl_input_image.pack(fill=tk.BOTH,
expand=True)                                  #центрування напису і його фону з
можливістю розтяжки

        #створення кнопки "ОБРОБИТИ" з заданим шрифтом, кольором фону і колором
шрифта, стилем внопки (relief=tk.RAISED) і викликом функції
self.process_image_trigger при натисканні
        self.btn_process = tk.Button(self.main_container,
text="➡\n\nO\nБ\nP\nO\nБ\nИ\nT\nИ\n\n➡", font=("Arial", 10, "bold"),
bg="#4CAF50", fg="white", relief=tk.RAISED, command=self.process_image_trigger)
        self.btn_process.pack(side=tk.LEFT, padx=10, fill=tk.Y,
pady=20)                    #розміщення кнопки

        #поле для відображення обробленого зображення, аргументи і логіка схожа на
поле для завантаженого зображення
        self.frame_output = tk.LabelFrame(self.main_container, text="Оброблене
зображення", width=380, height=400)
        self.frame_output.pack_propagate(False)
          #вимкнення зміни розміру рамки
        self.frame_output.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=5,
pady=5)     #менеджер геометрії .pack() розміщує рамку зліва з центруванням
елементів в ній, з можливістю розтягування і відступами по 5 пікселів
        self.lbl_output_image = tk.Label(self.frame_output, text="Тут буде
відображатись\nоброблене зображення з контурами", bg="#e0e0e0") #напис з сірим
фоном що відображається допоки не завантажене фото
        self.lbl_output_image.pack(side=tk.TOP, fill=tk.BOTH,
expand=True)                  #центрування напису і його фону з можливістю
розтяжки

        #поле для керування відображенням контурів дефектів
        self.frame_view_filters =
tk.Frame(self.frame_output)                                  #створення підфрейму
self.frame_view_filters в фреймі self.frame_output
        self.frame_view_filters.pack(side=tk.BOTTOM, fill=tk.X, padx=10,
pady=5)            #розміщення підфрейму

        #чекбокс для вимкнення/ввімкнення функції підсвічування контурів знайдених
точок
        f_row_pts =
tk.Frame(self.frame_view_filters)                                          #створення
підфрейму f_row_pts у фреймі self.frame_view_filters
        f_row_pts.pack(fill=tk.X,
pady=2)                                                    #розміщення підфрейму
f_row_pts
```

```
        lbl_pts_filter = tk.Label(f_row_pts, text="Відображати контури точок",
anchor=tk.W) #створення напису з текстом "Відображати контури точок" з привіязкою
до west грані
        lbl_pts_filter.pack(side=tk.LEFT, fill=tk.X,
expand=True)                              #розміщення напису "Відображати контури
точок"
        chk_pts_filter = tk.Checkbutton(f_row_pts, variable=self.show_points,
command=self.redraw_contours_canvas)  #створення чекбоксу результат вибору якого
буде зберігатись в змінній self.show_points і при натисканні на який буде
викликатись метод self.redraw_contours_canvas
        chk_pts_filter.pack(side=tk.RIGHT)
         #розміщення чекбоксу справа

        #чекбокс для вимкнення/ввімкнення функції підсвічування контурів знайдених
царапин
        f_row_scr =
tk.Frame(self.frame_view_filters)                                          #створення
підфрейму f_row_scr у фреймі self.frame_view_filters
        f_row_scr.pack(fill=tk.X,
pady=2)                                                  #розміщення підфрейму
f_row_scr
        lbl_scr_filter = tk.Label(f_row_scr, text="Відображати контури царапин",
anchor=tk.W)   #створення напису з текстом "Відображати контури царапин" з
прив'язкою до west грані
        lbl_scr_filter.pack(side=tk.LEFT, fill=tk.X,
expand=True)                              #розміщення напису "Відображати контури
царапин"
        chk_scr_filter = tk.Checkbutton(f_row_scr, variable=self.show_scratches,
command=self.redraw_contours_canvas)   #створення чекбоксу результат вибору якого
буде зберігатись в змінній self.show_scratches і при натисканні на який буде
викликатись метод self.redraw_contours_canvas
        chk_scr_filter.pack(side=tk.RIGHT)
         #розміщення чекбоксу справа

        self.frame_data = tk.Frame(self.main_container,
width=280)                             #створення фрейму в якому далі будуть
розміщені вікна "параметри обробки" і "результати обробки"
        self.frame_data.pack_propagate(False)
          #вимкнення зміни розміру фрейму
        self.frame_data.pack(side=tk.LEFT, fill=tk.Y, padx=5,
pady=5)                        #центрування фрейму в його батьківському фреймі з
відступами

        self.subframe_params = tk.LabelFrame(self.frame_data, text="Параметри
обробки")     #створення фрейму з підписом "Параметри обробки" для виведення
поточних налаштувань обробки
        self.subframe_params.pack(side=tk.TOP, fill=tk.X, padx=5,
pady=5)                    #розміщення фрейму "параметри обробки"
        #створення багаторядковорго текствого поля self.txt_params_display методом
tk.Text() і задавання параметрів тексту. В цьому полі будуть відображатись поточні
налаштування обробки
        self.txt_params_display = tk.Text(self.subframe_params, height=8,
bg="#fbfbfb", font=("Courier New", 9), state=tk.DISABLED)
        self.txt_params_display.pack(fill=tk.X, padx=5,
pady=5)                                #розміщення текстового поля
self.txt_params_display

        self.subframe_results = tk.LabelFrame(self.frame_data, text="Результати
обробки")   #створення фрейму з підписом "Результати обробки" для виведеня
інформації про результат обробки
        self.subframe_results.pack(side=tk.TOP, fill=tk.BOTH, expand=True, padx=5,
pady=5)  #розміщення фрейму "результати обробки"
        #кнопка перегляду послідовності прийняття рішень при обробці зображеняя
(вирішив виводити в сповіщення)
```

```
        self.result_img_save = tk.Button(self.subframe_results, text="Зберегти
оброблене зображення", font=("Arial", 9), bg="#e0e0e0", fg="black",
relief=tk.GROOVE, command=self.save_img)
        self.result_img_save.pack(side=tk.BOTTOM, fill=tk.X,
padx=5,pady=5)  #розміщення кнопки self.result_img_save

        self.txt_results = tk.Text(self.subframe_results, bg="#fcfcfc",
font=("Courier New", 10), state=tk.DISABLED)
        self.txt_results.pack(fill=tk.BOTH, expand=True, padx=5,
pady=5)                    #розміщення текстового поля self.subframe_results
        ##створення багаторядковорго текствого поля self.subframe_results методом
tk.Text() і задавання параметрів тексту. В цьому полі буде відображатись
інформація про оброблене зображення
        self.update_results_zone("Завантажте фото і виконайте обробку щоб побачити
результат")  #виклик функції оновлення поля з результатами обробки

        self.frame_notifications = tk.LabelFrame(self.main_container,
text="Сповіщення", width=300) #створення фрейму з підписом "Сповіщення" в якому
будуть відображатис сповіщенян і повідомлення що стосуються роботи програми
        self.frame_notifications.pack_propagate(False)
         #вимкнення зміни розміру фрейму
        self.frame_notifications.pack(side=tk.LEFT, fill=tk.BOTH, expand=True,
padx=5, pady=5)  #розміщення фрейму self.frame_notifications

        self.console_container =
tk.Frame(self.frame_notifications)                          #створення підфрейму
self.console_container у фреймі self.frame_notifications
        self.console_container.pack(fill=tk.BOTH, expand=True, padx=5,
pady=2)              #розміщення фрейму self.console_container

        self.scrollbar =
tk.Scrollbar(self.console_container)                                #створення
скролбару у фреймі self.console_container
        self.scrollbar.pack(side=tk.RIGHT,
fill=tk.Y)                                  #розміщення скролбару
        #створення багаторядкового текстового поля self.txt_console у фреймі
self.console_container і налаштування його стилю для відображення системних
повідомлень
        self.txt_console = tk.Text(self.console_container, bg="#f0f0f0",
fg="black", font=("Courier New", 9), state=tk.DISABLED,
yscrollcommand=self.scrollbar.set, wrap=tk.WORD)
        self.txt_console.pack(side=tk.LEFT, fill=tk.BOTH,
expand=True)                        #розміщення текствого поля self.txt_console
        self.scrollbar.config(command=self.txt_console.yview)
          #налаштуванян скролбару для виклику функції self.txt_console.yview

        #створення кнопки для очищення консолі зі сповіщеннями, задавання напису
кнопки, стилю напису і функції self.clear_console_history при натисканні
        self.btn_clear_console = tk.Button(self.frame_notifications,
text="Очистити історію", font=("Arial", 9), bg="#e0e0e0", fg="black",
relief=tk.GROOVE, command=self.clear_console_history)
        self.btn_clear_console.pack(side=tk.BOTTOM, fill=tk.X, padx=5,
pady=5)              #розміщенян кнопки self.btn_clear_console

        #функції для початкового оновлення інтерфейсу
        self.update_params_pane()
          #виклик функції оновлення параметрів поточних налаштувань в текствому
полі self.txt_params_display
        self.log_message("Систему успішно
ініціалізовано.")                                    #виклик функції надсилання
першого повідомлення в "консоль" зі сповіщеннями
```

```
    #функція оновлення відображення поточних налаштувань обробки. При виклику
оновлює текст у полі txt_params_display, пишучи туди актуальні значення поточних
налаштувань
    def update_params_pane(self):
        self.txt_params_display.config(state=tk.NORMAL)
          #дозвіл на зміну текстового поля  self.txt_params_display
        self.txt_params_display.delete("1.0",
tk.END)                                  #очищення тексту з встановленням поточного
кінця тексту у кординатах "1.0"
        dims_text = f"{self.img_w}x{self.img_h} px" if self.img_w > 0 else "Не
завантажено" #збереження запису розміру завантаженого зображення або повідомлення
про його відсутність в змінну dims_text
        info_text =
(                                                                        #запис в
змінну info_text усіх даних про поточні налаштування і розмір зображення
            f"Block Size:
{self.current_block_size}\n"                                          #розмір блоку
для бінаризації
            f"Constant C:
{self.current_constant_c}\n"                                          #константа для
бінаризації
            f"Morph Radius: {self.current_morph_size}
px\n"                                  #радіус для ерозії та дилатації
            f"Розмір вхідного зображення:
{dims_text}\n"                                    #розмір вхідного зображення
            f"Масштаб
калібрування:\n"                                                        #масштаб
калібрування
            f"1 пкс = {self.pixel_scale_x:.6f}
мм")                                           #масштаб калібрування з точністю 6
знаків після коми
        self.txt_params_display.insert(tk.END,
info_text)                                    #написання даних в змінній info_text
у текстове поле self.txt_params_display
        self.txt_params_display.config(state=tk.DISABLED)
          #заборона на зміну текстового поля self.txt_params_display

    #функція виводу результатів обробки зображення в поле результати обробки
    def update_results_zone(self, text):
        self.txt_results.config(state=tk.NORMAL)
         #дозвіл на зміну текстового поля self.txt_results
        self.txt_results.delete("1.0",
tk.END)                                     #очищення тексту з встановленням
поточного кінця тексту у кординатах "1.0"
        self.txt_results.insert(tk.END,
text)                                              #написання змінної text, що
передається в цю функцію як аргумент, у текстове поле self.txt_results
        self.txt_results.config(state=tk.DISABLED)
         #заборона на зміну текстового поля self.txt_results

    #функція відкриття вікна з налаштуваннями обробки
    def open_processing_settings_menu(self):
        if self.settings_win is not None and
self.settings_win.winfo_exists():              #якщо вікно з налаштуваннями не
дорівнює нічому і якщо воно відкрите
            self.settings_win.lift()
          #винести вікно на перший план
            self.settings_win.focus_set()
           #фокус клавіатури і миші на виведеному вікні
            return
          #закінчити виконання функції

          #код далі викнується тільки якщо вікна немає
```

```
        self.settings_win =
tk.Toplevel(self.root)                                          #створення
додаткового віка з назвою змінної self.settings_win
        self.settings_win.title("Параметри
обробки")                                       #назва вікна self.settings_win
        self.settings_win.geometry("490x500")
          #розмір вікна self.settings_win
        self.settings_win.resizable(False,
False)                                  #заборона на зміну розміру вікна
self.settings_win

        #локальні змінні інтерфейсу вікна налаштувань
        #block_size_var - змінна що відображається в інтерфейсі,
self.current_block_size - поточне налаштування на основі якого буде відбуватись
обробка
        block_size_var = tk.IntVar(value=self.current_block_size)
        constant_c_var = tk.IntVar(value=self.current_constant_c)
        morph_size_var = tk.IntVar(value=self.current_morph_size)

        #реалізація налаштування параметру Block Size
        tk.Label(self.settings_win, text="Розмір блоку (Block Size) [Непарні >
1]:", font=("Arial", 10, "bold"), fg="black").pack(anchor=tk.W, padx=15, pady=(15,
2)) #лейбл з загловком блок сайз в стилі
        frame_block_controls =
tk.Frame(self.settings_win)                                     #створення фрейму для
повзунка і чекбокса
        frame_block_controls.pack(fill=tk.X, padx=15,
pady=2)                                  #розміщення фрейму
        scale_block = tk.Scale(frame_block_controls, from_=3, to=99,
orient=tk.HORIZONTAL, variable=block_size_var, showvalue=False)    #створення
повзунка з значеннями від 3 до 99 в горизонтальному положенні
        scale_block.pack(side=tk.LEFT, fill=tk.X, expand=True, padx=(0,
10))                #розміщення повзунка
        tk.Entry(frame_block_controls, textvariable=block_size_var, width=6,
font=("Arial", 10), justify=tk.CENTER).pack(side=tk.LEFT)  #поле для вводу і
відображення числа
        tk.Checkbutton(frame_block_controls,
variable=self.lock_block).pack(side=tk.LEFT, padx=(10, 5)) #чекбокс для вимкнення
скидання і його розміщення

        #реалізація налаштування параметру константи С
        tk.Label(self.settings_win, text="Константа зсуву (Constant C):",
font=("Arial", 10, "bold"), fg="black").pack(anchor=tk.W, padx=15, pady=(12,
2))  #напис константа С
        frame_c_controls =
tk.Frame(self.settings_win)                                      #фрейм для
повзунка і чекбоксу
        frame_c_controls.pack(fill=tk.X, padx=15,
pady=2)                                   #розміщення фрейму
        scale_c = tk.Scale(frame_c_controls, from_=0, to=50, orient=tk.HORIZONTAL,
variable=constant_c_var, showvalue=False)    #створення повзунка
        scale_c.pack(side=tk.LEFT, fill=tk.X, expand=True, padx=(0,
10))                    #розміщення повзунка
        tk.Entry(frame_c_controls, textvariable=constant_c_var, width=6,
font=("Arial", 10), justify=tk.CENTER).pack(side=tk.LEFT)  #поле для вводу
значення
        tk.Checkbutton(frame_c_controls, variable=self.lock_c).pack(side=tk.LEFT,
padx=(10, 5)) #чекбокс для вимкнення скидання параметру

        #реалізація налаштування розміру матриці для ерозії ат делатації
        tk.Label(self.settings_win, text="Радіус об'єднання фрагментів
(Closing):", font=("Arial", 10, "bold"), fg="black").pack(anchor=tk.W, padx=15,
pady=(12, 2))    #клейбл з назвою параметру
```

```
        frame_morph_controls =
tk.Frame(self.settings_win)                                    #фрейм для повзунка і
чекбоксу
        frame_morph_controls.pack(fill=tk.X, padx=15,
pady=2)                                  #розміщення фрейму
        scale_morph = tk.Scale(frame_morph_controls, from_=1, to=30,
orient=tk.HORIZONTAL, variable=morph_size_var, showvalue=False)    #створення
повзунка
        scale_morph.pack(side=tk.LEFT, fill=tk.X, expand=True, padx=(0,
10))                #розміщення повзунка
        tk.Entry(frame_morph_controls, textvariable=morph_size_var, width=6,
font=("Arial", 10), justify=tk.CENTER).pack(side=tk.LEFT)  #поле для вводу
значення
        tk.Checkbutton(frame_morph_controls,
variable=self.lock_morph).pack(side=tk.LEFT, padx=(10, 5)) #чекбокс

        #реалізація налаштування параметру калібрування оптичного масштабу
        tk.Label(self.settings_win, text="Калібрування оптичного масштабу:",
font=("Arial", 10, "bold"), fg="black").pack(anchor=tk.W, padx=15, pady=(18,
2))   #лейбл
        frame_calib =
tk.Frame(self.settings_win)                                                #фрейм
        frame_calib.pack(fill=tk.X, padx=15,
pady=5)                                        #розміщення фрейму

        var_mm =
tk.StringVar(value=self.calib_mm_val)                                      #збереж
ення введеного значення мм у змінній var_mm, ініціалізуючи її змінною за
замовчуванням self.calib_mm_val
        entry_mm = tk.Entry(frame_calib, textvariable=var_mm, width=7,
font=("Arial", 10), justify=tk.CENTER)   #поле для вводу значення мм
        entry_mm.pack(side=tk.LEFT, padx=(0,
2))                                            #розміщення поля для вводу значення
мм

        tk.Label(frame_calib, text="мм =", font=("Arial", 10),
fg="black").pack(side=tk.LEFT, padx=2)   #лейбл з текстом

        var_px =
tk.StringVar(value=self.calib_px_val)                                      #збереж
ення введеного значення мм у змінній var_px, ініціалізуючи її змінною за
замовчуванням self.calib_px_val
        entry_px = tk.Entry(frame_calib, textvariable=var_px, width=7,
font=("Arial", 10), justify=tk.CENTER)   #поле для вводу значення px
        entry_px.pack(side=tk.LEFT,
padx=2)                                                #розміщення поля для вводу
значення px

        tk.Label(frame_calib, text="пікселів  -> ", font=("Arial", 10),
fg="black").pack(side=tk.LEFT, padx=2)  #лейбл з текстом

        lbl_ratio_res = tk.Label(frame_calib, text=f"1 піксель =
{self.pixel_scale_x:.6f} мм", font=("Arial", 10, "bold"), fg="black")  #лейбл для
обчисленого значення відношення мм/пкс
        lbl_ratio_res.pack(side=tk.LEFT,
padx=2)                                            #розміщення попереднього лейблу

        #функція перерахунку відношення мм/пікс
        def recalculate_scale(*args):
            try:
                val_mm = float(var_mm.get() if var_mm.get() else
0)                          #отримання введеного значення мм і try для перевірки
допустимості введеного значення мм
```

```
                val_px = float(var_px.get() if var_px.get() else
0)                          #отримання введеного значення пкс і try для перевірки
допустимості введеного значення пкс
                if val_px > 0 and val_mm >
0:                                                  #якщо введені значення міліметрів
та пікселів більше нуля
                    scale = val_mm /
val_px                                                  #обчислити значення
співвідношення
                    lbl_ratio_res.config(text=f"1 піксель = {scale:.6f} мм",
fg="black")    #зміна тексту в лейблі на відповідний
                    self.btn_process.config(state=tk.NORMAL)
          #розблокування кнопки обробити
                    return
scale                                                               #повернення
співвідношення
                else:
                    lbl_ratio_res.config(text="1 піксель = Помилка вводу",
fg="red")         #вивід тексту про помилку
                    self.btn_process.config(state=tk.DISABLED)
          #блокування кнопки обробити
                    self.log_message("Введені значення для обчислення
співвідношення мають бути більшими за 0. Обробку заблоковано.")    #вивід
сповіщення
                    return
0.0                                                                 #повернення нуля
            except ValueError:
                lbl_ratio_res.config(text="1 піксель = Помилка вводу",
fg="red")             #вивід тексту про помилку
                self.btn_process.config(state=tk.DISABLED)
          #блокування кнопки обробити
                self.log_message("Неправильний формат вводу в полях калібрування!
Будь ласка, використовуйте лише числа.")  #вивід сповіщення
                return
0.0                                                                     #повернення
нуля
        var_mm.trace_add("write",
recalculate_scale)                                  #функція-наглядач за зміною
значення змінної var_mm і виклику функції
        var_px.trace_add("write",
recalculate_scale)                                  #функція-наглядач за зміною
значення змінної var_px і виклику функції
        recalculate_scale()
          #виклик функції обчислення

        #функція збереження параметрів обробки
        def save_variables_state():
            try:
                b_size =
block_size_var.get()                                            #отримання
введеного значення розміру блока
                if b_size % 2 == 0 or b_size <=
1:                                          #перевірка чи значення непарне
                    b_size +=
1                                                       #якщо непарне, то
додати 1
                    block_size_var.set(b_size)
          #записати змінене значення
                    self.log_message(f"Коригування: Блок автоматично змінено на
непарне число: {b_size}")   #сповіщення в консоль про змінене значення

                self.current_block_size =
b_size                                      #запис поточного значення
величини блоку у змінну self.current_block_size
```

```
                self.current_constant_c = constant_c_var.get()
                self.current_morph_size = morph_size_var.get()
                self.calib_mm_val = var_mm.get()
                self.calib_px_val = var_px.get()
                self.pixel_scale_x = recalculate_scale()
                self.update_params_pane()
          #запуск функції оновлення налаштувань в зоні "параметри обробки"
                self.log_message("Параметри успішно збережено й оновлено в
головному вікні.")   #сповіщення в консоль
            except tk.TclError:
                self.log_message("Не вдалося зберегти параметри. Перевірте
правильність вводу.")#сповіщення в консоль при помилці

        #функція скидань налаштувань обробки до стандартних
        def reset_defaults():
            if not
self.lock_block.get():                                                  #якщо
чекбокс не натиснуто
                block_size_var.set(self.default_block_size)
          #скинути до величини за замовчуванням
            if not
self.lock_c.get():                                                      #якщо
чекбокс не натиснуто
                constant_c_var.set(self.default_constant_c)
          #скинути до величини за замовчуванням
            if not
self.lock_morph.get():                                                  #якщо
чекбокс не натиснуто
                morph_size_var.set(self.default_morph_size)
          #скинути до величини за замовчуванням
            self.log_message("Скинуто не зафіксовані параметри
обробки.")                    #сповіщення в консоль після скиду значення

        #кнопка скидань налаштувань за замовчуванням і її розміщення
        btn_reset = tk.Button(self.settings_win, text="Скинути за замовчуванням",
font=("Arial", 10, "bold"), bg="#f0f0f0", fg="black", command=reset_defaults)
        btn_reset.pack(fill=tk.X, padx=15, pady=(15, 5))
        #кнопка збереження параметрів обробки і її розміщення
        btn_save = tk.Button(self.settings_win, text="Зберегти параметри обробки",
font=("Arial", 10, "bold"), bg="#f0f0f0", fg="black",
command=save_variables_state)
        btn_save.pack(fill=tk.X, padx=15, pady=2)
        #кнопка запуску обробки і її розміщення
        btn_process_now = tk.Button(self.settings_win, text="Обробити",
font=("Arial", 10, "bold"), bg="#f0f0f0", fg="black",
command=self.process_image_trigger)
        btn_process_now.pack(fill=tk.X, padx=15, pady=2)
        #кнопка закриття вікна з налаштуваннями і її розміщення
        btn_close_win = tk.Button(self.settings_win, text="Закрити",
font=("Arial", 10, "bold"), bg="#f0f0f0", fg="black",
command=self.settings_win.destroy)
        btn_close_win.pack(fill=tk.X, padx=15, pady=2)

    #метод що описую роботу вікна з таблицею знайдених дефектів
    def open_defects_table_window(self):
        if self.table_win is not None and
self.table_win.winfo_exists():                      #якщо вікно не дорінює нічому і
не існує
            self.table_win.lift()
          #підняти вікно на передній план
            self.table_win.focus_set()
         #сфокусувати класіатуру і мишку на вікні
            self.refresh_table_data()
          #оновити дані у вікні
```

```
            return
        if not
self.detected_defects:                                                    #якщо
немає записаних дефектів
            self.log_message("УВАГА: Таблиця порожня. Спочатку виконайте ОБРОБКУ
зображення!")  #сповіщення в консоль

        self.table_win =
tk.Toplevel(self.root)                                                  #створення
вікна
        self.table_win.title("База виявлених дефектів на
лінзі")                              #назва нового вікна
        self.table_win.geometry("780x400")
         #розмір нового вікна

        self.lbl_table_info = tk.Label(self.table_win, text="Клацніть мишкою по
заголовку стовпчика для сортування. Клацніть по рядку для вибору контуру.",
font=("Arial", 9, "italic"), fg="gray") #текст - підказка вгорі вікна
        self.lbl_empty_table = tk.Label(self.table_win, text="Дані про дефекти
відсутні.\n\nБудь ласка, завантажте зображення та виконайте обробку.",
font=("Arial", 12, "bold"), fg="#7f8c8d") #текст-сповіщення для випадку якщо немає
дефектів

        self.table_container =
tk.Frame(self.table_win)                                        #створення фрейму-
контейнеру для таблиці дефектів
        scrollbar_y = tk.Scrollbar(self.table_container,
orient=tk.VERTICAL)                 #скролбар для таблиці
        scrollbar_y.pack(side=tk.RIGHT,
fill=tk.Y)                                          #розміщення скролбару

        columns = ("check", "id", "type", "length", "length_mm", "width",
"width_mm", "ratio")  #визначення стовпчиків майбутньої таблиці
        self.tree = ttk.Treeview(self.table_container, columns=columns,
show="headings", yscrollcommand=scrollbar_y.set)    #створення віджету
ttk.Treeview() для відображення даних у вигляді таблиц
        self.tree.pack(side=tk.LEFT, fill=tk.BOTH,
expand=True)                              #розміщення віджету таблиці
        scrollbar_y.config(command=self.tree.yview)
          #прив'язка скролбару до таблиці

        #назви заголовків колонок таблиці
        self.tree.heading("check", text="Вибір")
        self.tree.heading("id", text="ID дефекту")
        self.tree.heading("type", text="Тип дефекту")
        self.tree.heading("length", text="Довжина (px)")
        self.tree.heading("length_mm", text="Довжина (мм)")
        self.tree.heading("width", text="Ширина (px)")
        self.tree.heading("width_mm", text="Ширина (мм)")
        self.tree.heading("ratio", text="Співвідношення")

        #араметри ширини колонок
        self.tree.column("check", width=55, anchor=tk.CENTER)
        self.tree.column("id", width=75, anchor=tk.CENTER)
        self.tree.column("type", width=100, anchor=tk.CENTER)
        self.tree.column("length", width=95, anchor=tk.E)
        self.tree.column("length_mm", width=100, anchor=tk.E)
        self.tree.column("width", width=95, anchor=tk.E)
        self.tree.column("width_mm", width=100, anchor=tk.E)
        self.tree.column("ratio", width=100, anchor=tk.E)

        #використання heading() для впорядкування елементів в таблиці, а саме
сортування
        for col in columns:
```

```
            self.tree.heading(col, command=lambda _col=col:
self.sort_table_column(_col, False))
        #використання .bind() для прив'язки кліку по рядку таблиці для
підсвічування обраного дефекту
        self.tree.bind("<Button-1>", self.on_table_row_click)
        self.refresh_table_data()
          #оновлення даних в таблиці

    #метод сортування рядків за спаданням чи зростанням значення в вибраному
стовпці на вхід назва стовпця і прапорець напрямку сортування
    def sort_table_column(self, col_name, reverse_sort):
        if self.table_win is None or not
self.table_win.winfo_exists():                      #якщо вікна не існує
            return
         #закінчити метод
        list_items = [(self.tree.set(k, col_name), k) for k in
self.tree.get_children("")]  #список кортеджів з парами значень в клітинці і
ідентифікатором в tkinter

        #функція парсер що викликається до кожного елемента під час сортування
        def parse_sort_key(item_tuple):                                      #
            val =
item_tuple[0]                                                          #витягув
ання елемента під час сортування
            if val.endswith(":1"): val = val[:-
2]                                          #якщо елементом є співвідношення
сторін то прибирається 2 знака попереду
            if val == "[✓]": return
1                                                  #якщо стоїть галочка в
рядку з виділенням
            if val == "[ ]": return
0                                                  #якщо галочки в рядку з
виділенням не стоїть
            try:
                return
float(val)                                                        #якщо число
типу float - повертаємо його у цьому форматі
            except ValueError:
                return
val.lower()                                                       #якщо елемент
- рядок сортування за алфавітом

        list_items.sort(key=parse_sort_key,
reverse=reverse_sort)                          #сортування через ознаку надану
parse_sort_key()

        for index, (_, item_id) in
enumerate(list_items):                             #перемальовування таблиці
після сортування
            self.tree.move(item_id, "", index)

        #зміна направлення сортування для наступного натискання на заголовок
стовпця таблиці
        self.tree.heading(col_name, command=lambda:
self.sort_table_column(col_name, not reverse_sort))

    #метод для реалізації підсвічування окремого дефекту, на вхід івент-сутність з
даними координат кліку мишки
    def on_table_row_click(self, event):
        if self.table_win is None or not
self.table_win.winfo_exists():                      #перевірка на існування вікна
            return
```

```
        selected_item =
self.tree.identify_row(event.y)                                    #отримання з
івенту інформації про координату мишки і визначення того на який рядок натиснув
користувач
        if not
selected_item:                                                                #якщо
там де натиснув користувач немає рядка
            return
         #закінчити роботу методу
        row_values = self.tree.item(selected_item,
"values")                              #зчитування усіх значень в рядку
        if not
row_values:                                                                   #якщо
значень в рядку немає
            return
         #закінчити роботу методу
        defect_id =
int(row_values[1])                                                          #перетворе
ння значення номера контуру з стрінгу в інт

        #цикл для пошуку елементу в списку і інвертація стану його виділення
        for df in
self.detected_defects:                                                        #для
кожного елементу у списку знайдених дефектів self.detected_defects
            if df["id"] ==
defect_id:                                                          #якщо поле "id"
елементу дорівнює номеру виділеного дефекту defect_id
                df["is_highlighted"] = not
df["is_highlighted"]                                #інвертація стану вибору рядка
                check_marker = "[✓]" if df["is_highlighted"] else "[ ]"
                display_type = "Подряпина" if df["type"] == "scratch" else "Точка"
                self.tree.item(selected_item,
values=(                                          #зміна відображення галочки в рядку
                    check_marker, df["id"], display_type,
                    df["length_px"], df["length_mm"],
                    df["width_px"], df["width_mm"],
                    f"{df['ratio']}:1"
                ))
                break
        self.redraw_contours_canvas()
          #перемальовування контурів на зображення

    #метод оновлення таблиці з дефектами
    def refresh_table_data(self):
        if self.table_win is None or not
self.table_win.winfo_exists():                      #якщо таблиця існує
            return
         #закінчити виконання методу

        if not
self.detected_defects:                                                        #якщо
дефектів в списку дефектів немає
            self.table_container.pack_forget()
         #приховування таблиці
            self.lbl_table_info.pack_forget()
          #приховування заголовків
            self.lbl_empty_table.pack(fill=tk.BOTH, expand=True, padx=20,
pady=20)          #відображення повідомлення про відсутність дефектів
        else:
            self.lbl_empty_table.pack_forget()
         #прибирання повідомлення про відсутність дефектів
            self.lbl_table_info.pack(side=tk.TOP, fill=tk.X, padx=10,
pady=5)               #відображення заголовків таблиці
```

```
            self.table_container.pack(fill=tk.BOTH, expand=True, padx=10,
pady=5)            #відображення таблиці

            for row in
self.tree.get_children():                                                  #для кожного
рядку в списку рядків що поточно відображаються в таблиці
                self.tree.delete(row)
           #видалити поточний рядок

            for df in
self.detected_defects:                                                      #для кожного
знайденого дефекту
                check_marker = "[✓]" if df["is_highlighted"] else "[
]"                    #відобразити параметр вибору
                display_type = "Подряпина" if df["type"] == "scratch" else
"Точка"           #відобразити тип
                self.tree.insert("", tk.END,
values=(                                        #відобразити усі інші параметри
                    check_marker, df["id"], display_type,
                    df["length_px"], df["length_mm"],
                    df["width_px"], df["width_mm"],
                    f"{df['ratio']}:1"
                ))

    #метод відображення контурів дефектів в зоні "оброблене зображення"
    def redraw_contours_canvas(self):
        if self.cached_thresh is
None:                                                   #якщо ще не було
виконано бінаризацію
            return
         #зупинити виконання методу

        result_canvas = cv2.cvtColor(self.cached_thresh,
cv2.COLOR_GRAY2BGR)                 #перетворення бінарного зображення на
кольорове, щоб можна було малювати кольорові контури

        #сіре коло для відображення зони в якій проводився контроль
        h, w = result_canvas.shape[:2]
        centerx, centery = w // 2, h // 2
        r = min(w, h) // 2
        cv2.circle(result_canvas, (centerx, centery), r+1, (120, 120, 120), 1)

        #відображення контурів дефектів
        for df in
self.detected_defects:                                                    #для
кожного дефекту в списку визначених дефектів
            #фільтрація відображення за чекбоксами інтерфейсу
            if df["type"] == "point" and not self.show_points.get():
continue                #реакція на наявність галочки в чекбоксі "відображати
контури точок"
            if df["type"] == "scratch" and not self.show_scratches.get():
continue           #реакція на наявність галочки в чекбоксі "відображати контури
царапин"

            cnt =
self.cached_contours[df["raw_index"]]                                     #витягув
ання елементу масиву точок з self.cached_contours за індексом

            if
df["is_highlighted"]:                                                      #якщо
контур обрано в таблиці дефектів
                color = (0, 255,
255)                                                  #обрати жовтий як колір
```

```
            elif df["type"] ==
"scratch":                                                  #якщо тип дефекту -
царапина
                color = (255, 0,
0)                                                        #призначити синій колір
            else:
          #в інших випадках, тобто коли дефект - точка
                color = (0, 0,
255)                                                       #призначити червоний
колір

            cv2.drawContours(result_canvas, [cnt], -1, color,
1)           #нарисувати контур

        result_rgb = cv2.cvtColor(result_canvas,
cv2.COLOR_BGR2RGB)                          #конвертація кольорової карти
зображення з BGR в RGB для коректного відображення
        output_pil_image =
Image.fromarray(result_rgb)                                      #перетворення
об'єкта зображення в тип з яким може працювати Tkinter

        lbl_out_w =
self.lbl_output_image.winfo_width()                                      #масштабув
ання зображення під розмір вікна по ширині
        lbl_out_h =
self.lbl_output_image.winfo_height()                                     #масштабув
ання зображення під розмір вікна по висоті
        if lbl_out_w <= 1 or lbl_out_h <= 1: lbl_out_w, lbl_out_h = 380,
400                #перевірка дл коректного відображення вікна під час першого
запуску програми

        scale_factor_out = min(lbl_out_w / output_pil_image.size[0], lbl_out_h /
output_pil_image.size[1])  #обчислення коефіцієнту масштабування
        new_out_w = int(output_pil_image.size[0] *
scale_factor_out)                          #масштабування по ширині
        new_out_h = int(output_pil_image.size[1] *
scale_factor_out)                          #масштабування по висоті

        self.numpy_output_image = cv2.cvtColor(np.array(output_pil_image),
cv2.COLOR_RGB2BGR)   #збереження зображення в форматі масиву для можливості його
завантаження з проміжною зміною кольорової схеми
        resized_output = output_pil_image.resize((new_out_w, new_out_h),
Image.Resampling.LANCZOS)  #відображення масштабованого зображення за допомогою
якісного алгоритму інтерполяції Ланцоша
        self.tk_output_image =
ImageTk.PhotoImage(resized_output)                                 #конвертація
зображення з PIL об'єкту у підходящий для Tkinter об'єкт
        self.lbl_output_image.config(image=self.tk_output_image,
text="")                    #відображення зображення

    #метод додавання сповіщень в зону "сповіщення"
    def log_message(self, message):
        current_time =
datetime.now().strftime("%H:%M:%S")                                    #поточний час
        self.txt_console.config(state=tk.NORMAL)
         #ввімкнути зміну текствого поля self.txt_console
        self.txt_console.insert(tk.END, f"[{current_time}]
{message}\n")               #написати текст сповіщення
        self.txt_console.see(tk.END)
         #прокрутити вниз
        self.txt_console.config(state=tk.DISABLED)
         #вимкнути зміну текствого поля self.txt_console

    #метод очищення зони зі сповіщеннями
```

```
    def clear_console_history(self):
        self.txt_console.config(state=tk.NORMAL)
         #ввімкнути зміну текствого поля self.txt_console
        self.txt_console.delete("1.0",
tk.END)                                          #видалити весь текст і поставити
знак кінця текстового поля на координаті 1
        self.txt_console.config(state=tk.DISABLED)
         #вимкнути зміну текствого поля self.txt_console

    #метод завантаження в програму
    def load_image(self):
        file_types = [("Зображення JPEG", "*.jpg
*.jpeg")]                                    #список розширень файлів які будуть
відображатись в провіднику при його відкритті
        path = filedialog.askopenfilename(title="Оберіть знімок поверхні скла
(.jpg)", filetypes=file_types)    #в змінну записується шлях до файлу що обрав
користувач у викликаному системному вікні "відкрити файл"
        if
path:                                                                          #
якщо файл було обрано
            self.image_path =
path                                                        #збереження шляху до
файлу
            self.original_image =
Image.open(path)                                        #відкриття зображення за
шляхом і збереження його в змінну
            self.img_w, self.img_h =
self.original_image.size                                #запис розміру
завантаженого зображення в змінні
            self.update_params_pane()
          #оновити зону "параметри обробки"

            #обчислення коефіцієнта масштабування для того щоб зображення влізло в
потрібний контейнер
            lbl_w, lbl_h = 380,
400                                                      #запис розмірів вікна
в змінні
            #визначення коефіцієнта масштабу шляхом порівняння коефіцієнтів
відношення розміру зображення і вікна по горизонталі і по вертикалі
            scale_factor = min(lbl_w / self.original_image.size[0], lbl_h /
self.original_image.size[1])
            new_w = int(self.original_image.size[0] *
scale_factor)                            #обчислення ширини масштабованого зображення
            new_h = int(self.original_image.size[1] *
scale_factor)                            #обчислення висоти масштабованого зображення
            resized_img = self.original_image.resize((new_w, new_h),
Image.Resampling.LANCZOS)  #зміна розміру зображення з максимальним збереженням
якості за допомогою якісного алгоритму інтерполяції Ланцоша
            self.tk_input_image =
ImageTk.PhotoImage(resized_img)                          #конвертація зображення
з PIL об'єкту у підходящий для Tkinter об'єкт
            self.lbl_input_image.config(image=self.tk_input_image,
text="")                  #відображення зображення
            file_name = path.split('/')[-
1]                                                  #відрізання частини тексту від
шляху файлу щоб залишилась тільки назва файлу

            self.update_results_zone(f"Завантажено зображення:
{file_name}\nВиконайте обробку щоб побачити результат")  #оновлення зони з
результатами обробки щоб там відображалась назва зображення
            self.log_message(f"Успішно завантажено файл: {file_name}
({self.img_w}x{self.img_h} px)")   #написати сповіщення про успішно завантажений
файл в зону сповіщень
```

```
            self.detected_defects.clear()
          #очистити список знайдених дефектів
            self.cached_thresh =
None                                                    #прибрати з пам'яті
бінаризоване зображення

            if self.table_win is not None and
self.table_win.winfo_exists():                #якщо вікно з таблицею дефектів
існує
                self.refresh_table_data()
          #оновити його

    #функція збереженян зображення з поля "Оброблене зображення"
    def save_img(self):
        is_img = getattr(self, 'numpy_output_image',
None)                                  #перевірка на те чи існує зображення
        if is_img is not None:
            file_path = filedialog.asksaveasfilename( initialdir="./",
title="Зберегти оброблене зображення", defaultextension=".jpg;*.jpeg",
                filetypes=[
                    ("PNG файли (*.png)", "*.png"),
                    ("JPEG файли (*.jpg;*.jpeg)", "*.jpg;*.jpeg"),
                    ("Усі файли (*.*)", "*.*")
                ]
            )
            #self.log_message(f"file_path =
{file_path}")                                    #для відладки
            if file_path:
                try:
                    success = cv2.imwrite(file_path,
self.numpy_output_image)                #збереження матиці зображення за вказаним
шляхом і запис результату операції в змінну success
                    #self.log_message(f"success =
{success}")                                  #для відладки
                    if
success:                                                          #якщо файл
збережено успішно
                        self.log_message(f"Зображення успішно збережено:
{file_path}")      #сповіщення в "сповіщення" про успіх
                        #tk.messagebox.showinfo("Успіх", f"Зображення успішно
збережено за шляхом:\n{file_path}")    #системне вікно про успіх
                    else:
                        self.log_message("Не вдалося зберегти
зображення")                  #сповіщення в "сповіщення" про невдачу
                        tk.messagebox.showerror("Помилка", "Не вдалося зберегти
зображення.")   #системне вікно про помилку
                except Exception as
e:                                                    #якщо виникла помилка при
збереження файлу
                    self.log_message(f"Критична помилка при збереженні файлу:
{str(e)}")    #сповіщення в "сповіщення" про це
                    tk.messagebox.showerror("Критична помилка", f"Сталася помилка
при збереженні файлу:\n{e}")  #системне вікно з помилкою
                return
          #закінчити виконання функції
        else:
          #якщо зображення не існує
            self.log_message("Неможливо зберегти неіснуюче оброблене зображення.
Виконайте обробку і спробуйте ще раз.")
            return
          #закінчити виконання функції

    #метод призначення класу чистоти за ознакою царапин і точок, також
перевіряється скупчення дефектів, на вхід світловий діаметр деталі в мм
```

```
    def evaluate_dstu_class(self, light_diameter_mm):
        #оголошення змінних з усіма необхідними для розрахунків обмеженнями для
класів чистоти
        classes_order = ['I', 'II', 'III', 'IV', 'V', 'VI', 'VII', 'VIII',
'VIIIa', 'IX', 'IXa', 'NO_CLASS']
        dstu_classes_requirements = {
            'I': {'table2': {'max_scratch_width': 0.004, 'scratch_len_d_coef':
2.0, 'max_point_diameter': 0.02, 'point_count_d_coef': 0.5, 'cluster_d': 1.0,
'cluster_max_area': 0.004},
                  'table3': None,  #для класів I і II Таблиця 3 не застосовується
                  'table4': {'ignore_scratch_width': 0.001,
'ignore_point_diameter': 0.002}
            },
            'II': {'table2': {'max_scratch_width': 0.006, 'scratch_len_d_coef':
2.0, 'max_point_diameter': 0.05, 'point_count_d_coef': 0.5, 'cluster_d': 1.2,
'cluster_max_area': 0.006},
                   'table3': None,  #для класів I і II Таблиця 3 не застосовується
                   'table4': {'ignore_scratch_width': 0.001,
'ignore_point_diameter': 0.002}
            },
            'III': {'table2': {'max_scratch_width': 0.01, 'scratch_len_d_coef':
2.0, 'max_point_diameter': 0.1, 'point_count_d_coef': 0.5, 'cluster_d': 2.0,
'cluster_max_area': 0.02},
                    'table3': {'scratch_range': (0.006, 0.01),
'scratch_sum_length_d_coef': 1.0, 'point_range': (0.05, 0.1),
'point_count_d_coef': 0.2},
                    'table4': {'ignore_scratch_width': 0.002,
'ignore_point_diameter': 0.004}
            },
            'IV': {'table2': {'max_scratch_width': 0.02, 'scratch_len_d_coef':
2.0, 'max_point_diameter': 0.3, 'point_count_d_coef': 0.5, 'cluster_d': 5.0,
'cluster_max_area': 0.1},
                   'table3': {'scratch_range': (0.01, 0.02),
'scratch_sum_length_d_coef': 1.0, 'point_range': (0.1, 0.3), 'point_count_d_coef':
0.2},
                   'table4': {'ignore_scratch_width': 0.004,
'ignore_point_diameter': 0.01}
            },
            'V': {'table2': {'max_scratch_width': 0.04, 'scratch_len_d_coef': 2.0,
'max_point_diameter': 0.5, 'point_count_d_coef': 0.5, 'cluster_d': 10.0,
'cluster_max_area': 0.4},
                  'table3': {'scratch_range': (0.02, 0.04),
'scratch_sum_length_d_coef': 1.0, 'point_range': (0.3, 0.5), 'point_count_d_coef':
0.2},
                  'table4': {'ignore_scratch_width': 0.006,
'ignore_point_diameter': 0.02}
            },
            'VI': {'table2': {'max_scratch_width': 0.06, 'scratch_len_d_coef':
2.0, 'max_point_diameter': 0.7, 'point_count_d_coef': 0.5, 'cluster_d':25.0,
'cluster_max_area': 3.0},
                   'table3': {'scratch_range': (0.04, 0.06),
'scratch_sum_length_d_coef': 1.0, 'point_range': (0.5, 0.7), 'point_count_d_coef':
0.2},
                   'table4': {'ignore_scratch_width': 0.008,
'ignore_point_diameter': 0.04}
            },
            'VII': {'table2': {'max_scratch_width': 0.1, 'scratch_len_d_coef':
2.0, 'max_point_diameter': 1.0, 'point_count_d_coef': 0.5, 'cluster_d': 50.0,
'cluster_max_area': 10.0},
                    'table3': {'scratch_range': (0.06, 0.1),
'scratch_sum_length_d_coef': 1.0, 'point_range': (0.7, 1.0), 'point_count_d_coef':
0.2},
                    'table4': {'ignore_scratch_width': 0.01,
'ignore_point_diameter': 0.1}
```

```
            },
            'VIII': {'table2': {'max_scratch_width': 0.2, 'scratch_len_d_coef':
2.0, 'max_point_diameter': 2.0, 'point_count_d_coef': 0.4, 'cluster_d': None,
'cluster_max_area': None},
                     'table3': {'scratch_range': (0.1, 0.2),
'scratch_sum_length_d_coef': 1.0, 'point_range': (1.0, 2.0), 'point_count_d_coef':
0.2},
                     'table4': {'ignore_scratch_width': 0.014,
'ignore_point_diameter': 0.14}
            },
            'VIIIa': {'table2': {'max_scratch_width': 0.3, 'scratch_len_d_coef':
1.5, 'max_point_diameter': 2.0, 'point_count_d_coef': 0.4, 'cluster_d': None,
'cluster_max_area': None},
                      'table3': {'scratch_range': (0.2, 0.3),
'scratch_sum_length_d_coef': 0.7, 'point_range': (1.0, 2.0), 'point_count_d_coef':
0.2},
                      'table4': {'ignore_scratch_width': 0.014,
'ignore_point_diameter': 0.140}
            },
            'IX': {'table2': {'max_scratch_width': 0.3, 'scratch_len_d_coef': 2.0,
'max_point_diameter': 3.0, 'point_count_d_coef': 0.3, 'cluster_d': None,
'cluster_max_area': None},
                   'table3': {'scratch_range': (0.2, 0.3),
'scratch_sum_length_d_coef': 1.0, 'point_range': (2.0, 3.0), 'point_count_d_coef':
0.1},
                   'table4': {'ignore_scratch_width': 0.02,
'ignore_point_diameter': 0.2}
            },
            'IXa': {'table2': {'max_scratch_width': 0.4, 'scratch_len_d_coef':
1.5, 'max_point_diameter': 3.0, 'point_count_d_coef': 0.3, 'cluster_d': None,
'cluster_max_area': None},
                    'table3': {'scratch_range': (0.3, 0.4),
'scratch_sum_length_d_coef': 0.7, 'point_range': (2.0, 3.0), 'point_count_d_coef':
0.1},
                    'table4': {'ignore_scratch_width': 0.02,
'ignore_point_diameter': 0.2}
            },
            'NO_CLASS': {'table2': {'max_scratch_width': 100,
'scratch_len_d_coef': 100, 'max_point_diameter': 100, 'point_count_d_coef': 100,
'cluster_d': None, 'cluster_max_area': None},
                    'table3': {'scratch_range': (1, 1),
'scratch_sum_length_d_coef': 100, 'point_range': (1, 1), 'point_count_d_coef':
100},
                    'table4': {'ignore_scratch_width': 100,
'ignore_point_diameter': 100}
                    }
        }
        #розділення списку дефектів на 2 окремих списки з точками і царапинами
        scratches = [d for d in self.detected_defects if d["type"] == "scratch"]
        points = [d for d in self.detected_defects if d["type"] == "point"]
        #self.log_message(f"scratches =
{len(scratches)}")                                          #для відладки
        #self.log_message(f"points =
{len(points)}")                                             #для відладки
        max_scratch_width_mm = max([s["width_mm"] for s in scratches]) if
scratches else 0.0        #визначення максимальної ширини царапини і запис
результату в змінну
        max_point_diameter_mm = max([p["length_mm"] for p in points]) if points
else 0.0            #визначення найбільшого діаметру точки і запис результату в
змінну

        #self.log_message(f"Макс ширина царапини =
{max_scratch_width_mm}")                            #для відладки
```

```
        #self.log_message(f"Макс point diametr =
{max_point_diameter_mm}")                                   #для відладки

        #попередня класифікація тільки по ознаці максимальної ширини царапини
відповідно до таблиці 2 ДСТУ (клас IX не призначається, бо приналежність до нього
контролюється іншими параметрами потім)
        scratch_class_by_w = "NO_CLASS"
        for class_ in classes_order:
            if max_scratch_width_mm <=
dstu_classes_requirements[class_]['table2']['max_scratch_width']:
                scratch_class_by_w = class_
                self.log_message(f"Попередній клас по царапинам:{class_}")
                break
        #попередня класифікація тільки по ознаці максимального діаметру точки
відповідно до таблиці 2 ДСТУ (класи VIIIa і IXa тут не призначаються бо
приналежність до них контролюється іншими параметрами потім)
        point_class_by_d = "NO_CLASS"
        for class_ in classes_order:
            if max_point_diameter_mm <=
dstu_classes_requirements[class_]['table2']['max_point_diameter']:
                point_class_by_d = class_
                self.log_message(f"Попередній клас по точкам:{class_}")
                break

        scratch_class_index =
classes_order.index(scratch_class_by_w)                          #визначення індексу
елементу попередньо обчисленого класу чистоти за ознакою максимальної ширини
цаарпин
        point_class_index =
classes_order.index(point_class_by_d)                             #визначення
індексу елементу попередньо обчисленого класу чистоти за ознакою діаметру точок

        #цикл для перевірки всіх умов для царапин
        for class_n in classes_order[scratch_class_index:]:
            scratch_total_l_mm =
0.0                                                          #сумарна довжина
царапин
            scratch_total_l_mm_3 =
0.0                                                        #сумарна довжина великих
царапин за умовою в таблиці 3
            self.log_message(f"Перевірка умов по царапинам для класу:{class_n}")
            for scr in scratches:
                if scr["width_mm"] >=
dstu_classes_requirements[class_n]['table4']['ignore_scratch_width']: #враховувати
тільки царапини більші за допуск в таблиці 4
                    scratch_total_l_mm +=
scr["length_mm"]                                       #інкримент довжини царапин
                if scr["width_mm"] >=
dstu_classes_requirements[class_n]['table3']['scratch_range'][0] and
scr["width_mm"] <=
dstu_classes_requirements[class_n]['table3']['scratch_range'][1]: #враховувати
тільки великі царапини з таблиці 3
                    scratch_total_l_mm_3 +=
scr["length_mm"]                                     #інкримент довжини великих царапин
            #перевірка царапин за обома умовами
            if {scratch_total_l_mm <=
(dstu_classes_requirements[class_n]['table2']['scratch_len_d_coef']*light_diameter
_mm) and (scratch_total_l_mm_3 <=
(dstu_classes_requirements[class_n]['table3']['scratch_sum_length_d_coef']*light_d
iameter_mm))}:
                scratch_class =
class_n                                                    #якщо відповідає усім
умовам то присвоюєме зміній яка повертається значення поточного класу і
закінчуємо цикл перевірок для царапин
```

```
                self.log_message(f"Перевірки по царапинам успішно пройдені для
класу:{class_n}")
                break

        #цикл для перевірки всіх умов для точок
        for class_n in classes_order[point_class_index:]:
            points_amount =
0                                                                   #сумарна кількість
точок
            points_amount_3 =
0                                                                 #сумарна кількість
великих точок
            self.log_message(f"Перевірка умов по точкам для класу:{class_n}")
            for pt in points:
                if pt["width_mm"] >=
dstu_classes_requirements[class_n]['table4']['ignore_point_diameter']:
#враховувати тільки царапини більші за допуск в таблиці 4
                    points_amount +=
1                                                          #інкримент кількості точок
                if pt["width_mm"] >=
dstu_classes_requirements[class_n]['table3']['point_range'][0] and pt["width_mm"]
<= dstu_classes_requirements[class_n]['table3']['point_range'][1]:  # враховувати
тільки точки більші за допуск в таблиці 3
                    points_amount_3 +=
1                                                       #інкримент кількості великих
точок
            #перевірка точок за обома умовами
            if {points_amount <=
(dstu_classes_requirements[class_n]['table2']['point_count_d_coef']*light_diameter
_mm) and (points_amount_3 <=
(dstu_classes_requirements[class_n]['table3']['point_count_d_coef']*light_diameter
_mm))}:
                point_class = class_n     #якщо відповідає усім умовам то
присвоюєме змінінй яка повертається значення поточного класу і закінчуємо цикл
перевірок
                self.log_message(f"Перевірки по точкам успішно пройдені для
класу:{class_n}")
                break

        #перевірка скупчення дефектів
        cluster_class_index = max(classes_order.index(scratch_class),
classes_order.index(point_class))   #визначення індексу найгіршого класу з точок
або царапин для початку перевірки скупчення
        cluster_class = classes_order[cluster_class_index]
        #for defect in self.detected_defects:
        for class_n in classes_order[cluster_class_index:]:
            #обробка випадків коли обмеження не накладається
            if (dstu_classes_requirements[class_n]['table2']['cluster_d'] is None)
or (dstu_classes_requirements[class_n]['table2']['cluster_max_area'] is None):
                cluster_class = class_n
                self.log_message(f"Для класу {class_n} скупчення не нормується.")
                break

            #обчислення радіусу в якому будуть рахуватись дефекти
            cluster_radius_px =
int((dstu_classes_requirements[class_n]['table2']['cluster_d']/self.pixel_scale_x)
/2.0)
            class_passed = True

            for defect in
self.detected_defects:                                          #для кожного
знайденого дефекту
                cnt = self.cached_contours[defect["raw_index"]]
```

```
                M =
cv2.moments(cnt)                                                    #обчисленн
я просторових і центральних моментів зображення для знаходження геометричного
центру дефекту
                if M["m00"] !=
0:                                                                  #момент "m00" для
контуру це по-факту його площина
                    cx = int(M["m10"] /
M["m00"])                                               #ділення m10 суми координат
усіх точок на площу дає точний центр фігури по осі x
                    cy = int(M["m01"] /
M["m00"])                                               #ділення m01 суми координат
усіх точок на площу дає точний центр фігури по осі y
                else:
                    cx, cy = cnt[0][0][0],
cnt[0][0][1]                                        #обробка ситуації коли внутрішня
площа контуру дорівнює нулю для точок в 1 піксель

                #чорна маска з колом радіусу з дефектами
                mask =
np.zeros_like(self.cached_thresh)                                   #чорна маска
                cv2.circle(mask, (cx, cy), cluster_radius_px, 255, -
1)          #коло з визначеним радіусом на масці
                cluster_roi = cv2.bitwise_and(self.cached_thresh,
mask)                      #накладання маски з колом на бінаризоване зображення
                contours_in_cluster, _ = cv2.findContours(cluster_roi,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)  #пошук контурів на новому зображенні
                total_area_px2 =
0                                                                   #сумарна площа
знайдених контурів
                for contour_ in
contours_in_cluster:                                                #цикл для перебору
кожного знайденого контуру
                    area = cv2.contourArea(contour_)
                    if area ==
0:                                                                  #обробка ситуації
коли внутрішня площа контуру дорівнює нулю для точок в 1 піксель
                        area = 1.0
                    total_area_px2 +=
area                                                            #інкримент площі
                total_area_mm2 = total_area_px2 * (self.pixel_scale_x **
2)                #перерахунок з квадратних пікселів в квадратні міліметри

                if total_area_mm2 >
dstu_classes_requirements[class_n]['table2']['cluster_max_area']:   #перевірка
площі скупчення дефектів з центром в одному дефекті
                    class_passed = False
                    break

            if
class_passed:                                                                   #якщо
для всіх дефектів в цьому класі умова скупчення виконана
                cluster_class = class_n
                self.log_message(f"Перевірка на скупчення успішно пройдена для
класу: {class_n}")
                break
            else:
                self.log_message(f"Площа скупчення дефектів перевищує норму для
класу {class_n}. Зниження класу")

        return scratch_class, point_class

    def process_image_trigger(self):
```

```
        if self.original_image is
None:                                                    #якщо не було
завантажено зображення
            self.update_results_zone("Неможливо почати обробку. Спочатку
завантажте фото!")
            return

        file_name = self.image_path.split('/')[-
1]                                          #обрізання файлового шляху для
визначення назви зображення
        self.log_message(f"Запущено обробку зображення: {file_name}")
        self.detected_defects.clear()
          #видалення попередньо знайдених дефектів

        pil_array =
np.array(self.original_image)                                                #конвертац
ія зображення в матрицю NumPy для роботи з цим зображенням інструментами OpenCV
        open_cv_image = cv2.cvtColor(pil_array,
cv2.COLOR_RGB2BGR)                          #конвертація з формату RGB в BGR
        gray = cv2.cvtColor(open_cv_image,
cv2.COLOR_BGR2GRAY)                              #конвертація зображення в сіре
        blurred = cv2.GaussianBlur(gray, (5, 5),
0)                           #фільтрація гаусса для прибирання мікрошумів матриці
камери

        #адаптивна бінаризація за заданими налаштуваннями self.current_block_size,
self.current_constant_c
        thresh = cv2.adaptiveThreshold(blurred, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY_INV, self.current_block_size,
self.current_constant_c)

        #морфологічна операція закриття реалізована через дилатацію і ерозію з
заданим в налаштуваннях розміром ядра, потрібно для об'єднання частин одного
дефекту до купи
        kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,
(self.current_morph_size, self.current_morph_size))
        closed_mask = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel)

        h, w =
closed_mask.shape[:2]                                                        #висо
та і ширина зображення
        cx, cy = w // 2, h //
2                                                  #визначення центру
кола
        r = min(w, h) //
2                                                       #визначення
радіусу кола

        light_diameter_px = 2 *
r                                                  #світловий діаметр в
пікселях
        light_diameter_mm = light_diameter_px *
self.pixel_scale_x                          #світловий діаметр в міліметрах

        circle_mask =
np.zeros_like(closed_mask)                                           #чорна маска
        cv2.circle(circle_mask, (cx, cy), r, 255, -
1)                          #коло з визначеним радіусом на масці
        closed_mask = cv2.bitwise_and(closed_mask,
circle_mask)                            #накладання маски з колом на бінаризоване
зображення

        contours, _ = cv2.findContours(closed_mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE) #пошук контурів дефектів в світловому діаметрі
```

```
        self.cached_thresh =
closed_mask                                                    #збереження
зображення з маскою
        self.cached_contours =
contours                                                     #збереження знайдених
контурів

        total_points =
0                                                                #загальна
кількість точок для вікна "результати обробки"
        total_scratches =
0                                                              #загальна
кількість царапин для вікна "результати обробки"
        assigned_id =
1                                                                 #id дефекту
для структури даних з усіма збереженими дефектами

        for idx, cnt in
enumerate(contours):                                           #індекс і
контур для кожного знайденого контуру
            if len(cnt) ==
0:                                                            #якщо контури
закінчились
                continue

            rect =
cv2.minAreaRect(cnt)                                             #пошук
найменшого прямокутника який може описати контур дефекту
            side_a =
rect[1][0]                                                        #1ша
сторона прямокутника
            side_b =
rect[1][1]                                                        #2га
сторона прямокутника
            length_px = max(side_a,
side_b)                                                #пошук довшої сторони
прямокутника
            width_px = min(side_a,
side_b)                                                 #пошук коротшої сторони
прямокутника

            #обробка ситуації коли дефект має розмір 1го пікселя
            if length_px < 1.0: length_px = 1.0
            if width_px < 1.0: width_px = 1.0

            length_mm = length_px *
self.pixel_scale_x                                     #переведення довжини
дефекту з пікселів в міліметри
            width_mm = width_px *
self.pixel_scale_x                                      #переведення ширини
дефекту з пікселів в міліметри

            sides_ratio = length_px /
width_px                                             #обчислення співвідношення
сторін дефекту

            if sides_ratio >=
3.0:                                                        #класифікація дефекту
царапина / точка
                defect_type = "scratch"
                total_scratches += 1
            else:
                defect_type = "point"
```

```
                total_points += 1

            #об'єкт дефекту з усіма необхідними для подальших операцій параметрами
            defect_object = {
                "id":
assigned_id,                                                        #id
                "type":
defect_type,                                                      #тип дефекту
царапина/точка
                "length_px": round(length_px,
2),                                         #довжина в пікселях
                "length_mm": round(length_mm,
4),                                         #довжина в міліметрах
                "width_px": round(width_px,
2),                                           #ширина в пікселях
                "width_mm": round(width_mm,
4),                                           #ширина в міліметрах
                "ratio": round(sides_ratio,
2),                                           #співвідношення сторін
                "area_px":
cv2.contourArea(cnt),                                               #площа в пікселях
                "is_highlighted":
False,                                                  #стан підсвічування
користувачем в таблиці
                "raw_index":
idx                                                             #індекс у масиві
OpenCV контурів
            }
            self.detected_defects.append(defect_object)
          #додавання поточного дефекту до списку дефектів
            assigned_id +=
1                                                                   #збільшення
лічильника id дефектів

        self.redraw_contours_canvas()
          #оновлення графічного вікна з обробленим зображенням

        scratch_cls, point_cls =
self.evaluate_dstu_class(light_diameter_mm)                 #виклик функції
визначення класу чистоти

        #формування повідомлення для поля "результати обробки"
        results_formatted = (
            f"Світловий діаметр: {light_diameter_mm:.2f} мм\n"
            f"Всього знайдено дефектів: {len(self.detected_defects)}\n"
            f"З них точок: {total_points}\n"
            f"З них царапин: {total_scratches}\n"
            f"-------------------------------\n"
            f"Клас чистоти за ДСТУ:\n P {scratch_cls}/{point_cls}"

        )
        self.update_results_zone(results_formatted)
          #оновлення поля "результати обробки"
        self.update_params_pane()
          #оновлення поля "параметри обробки"

        #оновлення таблиці з дефектами, якщо відкрите вікно таблиці
        if self.table_win is not None and self.table_win.winfo_exists():
            self.refresh_table_data()

        self.log_message(f"Обробку завершено. Клас чистоти за ДСТУ:\n P
{scratch_cls}/{point_cls}.")    #сповіщення про закінчену обробку
```

```
#вхід в програму
if __name__ == "__main__":
    window =
tk.Tk()                                                                    #ст
ворення базового контейнера вікна
    app =
GlassPurityApp(window)
#створення екземпляру класу додатка GlassPurityApp() з контейнером вікна в якості
аргументу
        window.mainloop()
             #запуск нескінченного циклу обробки подій вікна
```