МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ, КОМП'ЮТЕРНОЇ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ
КАФЕДРА КОМП'ЮТЕРИЗОВАНИХ СИСТЕМ УПРАВЛІННЯ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____ Литвиненко О. Є.
«_____»_____2021 р.

ДИПЛОМНИЙ ПРОЄКТ
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ
«БАКАЛАВР»

Тема: «Система відтворення креслень. Підсистема перетворення керуючих програм»

Виконавець: Майструк М.П.

Керівник: Масловський Б.Г.

Нормоконтролер: Тупота Є.В.

Київ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет Кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютеризованих систем управління
Спеціальність 123 «Комп'ютерна інженерія»
(шифр, найменування)

Освітньо-професійна програма «Системне програмування»
Форма навчання денна

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Литвиненко О. Є.
«_____»_____2021 р.

ЗАВДАННЯ
на виконання дипломної роботи (проєкту)

Майструку Максиму Петровичу
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема дипломної роботи (проекту) «Система відтворення креслень. Підсистема перетворення керуючих програм»
затверджена наказом ректора від «_04_» _лютого_ 2021 р. № 135/ст.

2. Термін виконання роботи (проєкту): з _17 травня 2021 р._ по _20 червня 2021 р._

3. Вихідні дані до роботи (проєкту): мова програмування *C#*, інтегроване середовище розробки *Microsoft Visual Studio Community* 2017, документація до програмної бібліотеки *Emgu CV*, ДСТУ 3008:2015, *ISO* 6983-1:2009.

4. Зміст пояснювальної записки:
Аналіз принципів побудови сучасних систем перетворення керуючих програм для пристроїв з числовим програмним керуванням. Проектування підсистеми перетворення керуючих програм. Програмна реалізація підсистеми перетворення керуючих програм.

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:
1) структурна схема системи відтворення креслень,
2) діаграма послідовності підсистеми перетворення керуючих програм,
3) схема алгоритму перетворення керуючих програм,
4) діаграма класів підсистеми перетворення керуючих програм,
5) екранні форми підсистеми перетворення керуючих програм.

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1.	Провести огляд літератури за темою дипломного проекту та аналіз існуючих систем	17.05.2021-18.05.2021	
2.	Зробити вибір середовища програмування і компонентів підсистеми	18.05.2021-19.05.2021	
3.	Розробити структуру програмних засобів системи	19.05.2021-21.05.2021	
4.	Розробити програмні модулі. Провести відладку програмних модулів	22.05.2021-23.05.2021	
5.	Написати пояснювальну записку	24.05.2021-05.05.2021	
6.	Підготувати графічний та ілюстративний матеріал	06.06.2021-09.06.2021	
7.	Підготувати доповідь та презентацію	10.06.2021-14.06.2021	

7. Дата видачі завдання: « 17 » травня 2021 р.

Керівник дипломної роботи (проєкту) _____ Масловський Б. Г.
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Майструк М. П.

РЕФЕРАТ

Пояснювальна записка до дипломного проекту «Система відтворення креслень. Підсистема перетворення керуючих програм.»: 64 сторінки, 28 рисунків, 14 літературних джерел, 2 таблиці, 1 додаток.

ЧПК, КЕРУЮЧА ПРОГРАМА, *G*-КОД, *WINDOWS FORMS*, *C#*, *.NET*

Об'єкт – методи генерації керуючих програм мовою *G*-кодів на основі растрових зображень.

Предмет – програма підсистеми перетворення керуючих програм.

Мета дипломного проекту – створити програму підсистеми перетворення керуючих програм.

Метод проектування – мова програмування *C#*, інтегроване середовище розробки *Microsoft Visual Studio Community* 2017, програмна бібліотека для обробки зображень *Emgu CV*.

Результати дипломного проектування рекомендується використовувати для створення керуючих програм мовою *G*-кодів на основі растрових зображень з метою подальшого їх застосування для керуванням пристроями з числовим програмним керуванням.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ 6
ВСТУП 7
РОЗДІЛ 10 АНАЛІЗ ПРИНЦИПІВ ПОБУДОВИ СУЧАСНИХ СИСТЕМ ЧИСЛОВОГО ПРОГРАМНОГО КЕРУВАННЯ 10
1.1. Основні поняття в системах з числовим програмним керуванням 10
1.2. Аналіз статистики використання засобів генерації керуючих програм. 13
1.3. Висновки до розділу 21
РОЗДІЛ 2 ПІДСИСТЕМА ПЕРЕТВОРЕННЯ КЕРУЮЧИХ ПРОГРАМ 24
2.1. Функціональні вимоги 24
2.2. Інструментальні засоби розробки 26
2.3. Структура програмного модуля 28
2.4. Висновки до розділу 34
РОЗДІЛ 3 ПРОГРАМА ПЕРЕТВОРЕННЯ КЕРУЮЧИХ ПРОГРАМ 35
3.1. Системні вимоги 35
3.2. Алгоритми роботи підсистеми перетворення керуючих програм 36
3.3. Етапи створення програми 46
3.4. Тестування 51
3.5. Інструкція користувача 57
3.6. Висновки до розділу 61
ВИСНОВКИ 62
СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ 63
ДОДАТОК А 65

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

АСТПВ – автоматизована система технологічної підготовки виробництва

САПР ТП – система автоматизованого проектування і розрахунку технологічних процесів

ППКП – підсистема перетворення керуючих програм

CNC – computer numerical control

CAD – computer-aided design

CAM – computer-aided manufacturing

CAPP – computer-aided process planning

IDE – integrated development environment

ВСТУП

Одним із основних напрямків розвитку технологічних методів обробки матеріалів в сучасному світі є підвищення їх універсальності та продуктивності. Досягти цього дозволяє шлях широкого застосування багатоцільових верстатів з ЧПК, гнучких виробничих систем. Стрімкий розвиток галузі мікроелектроніки відчутно сприяє швидкому поширенню цього напрямку верстатобудування.

Верстати з ЧПК дозволяють підвищити рівень автоматизації процесів обробки матеріалів, характеризуються малим часом на переналагодження при невеликих партіях деталей та забезпечують високий рівень якості обробки цих деталей.

Перший верстат з числовим програмним керуванням був винайдений американцем Джоном Персонсом, який працював на фірмі, що виготовляла ґвинти для ґвинтокрилів, ним було запропоновано використовувати верстат, який працював за програмою, що вводилась з перфокарт. В 1952 році було представлено перший верстат з ЧПК. Перший серійний пристрій ЧПК був виготовлений і встановлений на верстат в 1954 році. Одними із перших вітчизняних верстатів з ЧПК для промислового використання були металообробні 1К62ПУ та 1541П.[1]

Найбільший сегмент ринку ЧПК верстатів – це металообробні верстати, однак застосування систем з ЧПК продовжує поширюватися в інших сферах, таких як адитивне виробництво, лазерне різання, гравіювання, виробництво електронних плат, струменеве різання, обробка дерева, пластмас та скла, паперу.

Як правило, верстати з ЧПК дорожчі, ніж верстати з ручним управлінням. Крім того, для забезпечення належної роботи з такими пристроями персонал потребує відповідної підготовки та оволодіння спеціальними професійними навичками. Для часткового нівелювання зазначених проблем ведеться активна робота в області розробки програмного забезпечення задля підвищення рівня автоматизації у виробництві.

В теперішніх обставинах виділяють два напрямки застосування ЕОМ в машинобудуванні, це автоматизація виробничих процесів і автоматизація інженерної праці. До першої групи відносять устаткування з ЧПК, гнучкі виробничі комплекси і автоматизовані системи керування технологічними процесами і виробництвом. До другої ж належать САПР що дозволяють розробляти технологічні процеси, керуючі програми для устаткування з ЧПК та інше. Відповідно цю сукупність вирішуваних задач можна прийнято розрізняти за видами вихідних матеріалів та два види:

– машинний друк і створення різноманітної технологічної документації в рамках вимог, тобто креслень, графіків, набору карт технологічних процесів та іншої документації;

– створення та запис керуючих програм на різних носіях, необхідних для верстатів з ЧПК, використання пристроїв керованих ЕОМ, а також безпосередня передача керуючих програм на верстати з ЧПК.

Програмне забезпечення, призначене для роботи з ЧПК верстатами – це міст який з’єднує модель розроблюваної деталі з реальністю. Загалом, можна розділити процес на чотири етапи.

1) Проектування моделі.

2) Генерація керуючої програми.

3) Передача та обробка керуючої програми підсистемою керування виконуючими пристроями.

4) Конвертація цифрових сигналів в аналогові, їх подача на органи управління.[2]

В дипломному проєкті розглядаються перші три з зазначених вище чотирьох пунктів. В групу програмного забезпечення призначеного для використання на цих двох етапах входять: системи автоматизованого проєктування і розрахунків технологічних процесів, автоматизовані системи технологічного підготовки виробництва, система автоматизації інженерних розрахунків,

Метою дипломного проєкту є розробка програмної реалізації підсистеми перетворення керуючих програм. Розробка та реалізація основних функції САПР,

АСТПВ. Підсистема має забезпечувати користувача інструментами базової (легкої) САПР призначеної для двовимірного проектування та підготовки вхідної інформації до наступного етапу обробки. Також система покликана виконувати основні функції АСТПВ, а саме планування траєкторій руху виконуючого інструменту та їх корегування. В більшості своїй продукт представляє собою прикладне програмне забезпечення із інтегрованими модулями системного програмного забезпечення, наприклад драйверами для зв'язку із пристроями з ЧПК.

Об'єктом проектування є процес генерації керуючих програм за допомогою стандартизованої мови *G*-кодів на основі моделей потенційних виробів в форматі растрових зображень. Предметом проектування є програмна реалізація підсистеми перетворення керуючих програм.

Для реалізації вирішено використати сучасні методи розробки програмного забезпечення, а саме розробка з використанням стеку технологій *.NET* мовою *C#* в інтегрованому середовищі розробки *Microsoft Visual Studio Community 2017* та інтерфейсом програмування додатків Windows Forms. Для функціонального забезпечення деяких алгоритмів використано бібліотеку для обробки зображень *Emgu CV*. Оформлення документації та ілюстративних матеріалів виконано графічний редактор *yEd Graph Editor* та текстовий процесор *Microsoft Word*.

В результаті створюється програмне забезпечення, як важлива складова системи для роботи із пристроями ЧПК, так і окремий продукт здатний до роботи в відособленому вигляді, наприклад генерація керуючих програм без наступної їх відправки пристрою, а для збереження в окремому файлі і будь якого іншого використання. Можливі варіанти використання для перегляду, створення та редагування графічних зображень, креслень та безпосередньо управляючих програм. Система покликана зменшити трудомісткість проектування, планування, скорочення собівартості проектування та виготовлення, зменшення затрат на експлуатацію, підвищення якості і технічно-економічного рівня проектування і виготовлення.

РОЗДІЛ 1
АНАЛІЗ ПРИНЦИПІВ ПОБУДОВИ СУЧАСНИХ СИСТЕМ ЧИСЛОВОГО ПРОГРАМНОГО КЕРУВАННЯ

1.1. Основні поняття в системах з числовим програмним керуванням

На сьогодні верстати з ЧПК знайшли широке розповсюдження як на великих, середніх і малих підприємствах так і в побуті. Розвиток напрямку потребує від операторів верстатів і розробників ПЗ розуміння структури, можливостей пристроїв з ЧПК та вмінь для ефективного застосування цього обладнання. Для повноцінного сприйняття згаданих вище пунктів важливо знати та орієнтуватись у галузевій термінології.

Числове програмне керування – це автоматичне управління шляхом передачі інформації у формі чисел від програмоносія до виконавчого органу, яка визначає його рух або виконання ним інших функцій.

Верстат з ЧПК – це моторизований маневрений інструмент та платформа якими за допомогою керуючих програм керує комп'ютер. Інструкції доставляються до верстату у вигляді послідовних програм складених зазвичай за допомогою *G*-кодів та *M*-кодів. Програми можуть створюватись людьми, але частіше всього створюються за допомогою програмного забезпечення (САПР).

Система ЧПК (СЧПК) – це поняття під яким розуміють сукупність функціонально взаємозалежних і взаємодіючих технічних і програмних засобів, що забезпечують числове програмне управління устаткуванням. Безпосередньо пристрій числового програмного керування (ПЧПК) складає частину системи і конструктивно виконується у вигляді окремого блока.

Керуюча програма – це системна програма, що реалізує набір функцій керування, який включає у себе керування ресурсами та взаємодію з оточуючим середовищем системи обробки інформації, відновлення роботи системи після виявлення несправностей у технічних засобах. Керуюча програма включає в себе закодовану інформацію про траєкторії, швидкості та інші дані що необхідні для виконання обробки деталі. Під час роботи керуюча програма кадр за кадром поступає на виконання. Відповідно до команд керуючої програми контролер викликає із постійної пам'яті відповідні системні підпрограми, які і змушують працювати підключене до ЧПК обладнання в заданому режимі.[3]

G-код – загальноприйнята назва однієї з найбільш широковживаних мов програмування числовим програмним керуванням. Здебільшого застосовується в галузі автоматизації. Структура мови стандартизована документом *ISO6983* міжнародного комітету стандартів. Попри назву включає в себе програмування за допомогою *G*-кодів та *M*-кодів. Офіційно мова *G*-кодів визнана стандартом для європейських та американських виробників пристроїв ЧПК також має назву «*ISO* 7 біт».[2]

Графопобудовник – пристрій, призначений для виведення даних в графічній формі на папір.

Система автоматизованого проектування і розрахунку (САПР, *CAD*) – інформаційний комплекс призначений для автоматизації проектування, що складається із апаратного забезпечення (ЕОМ), програмного забезпечення, описів способі і методів роботи зі системою, протоколом правил зберігання. Однак з поширенням в середовищі спеціалістів термін дещо втратив свій початковий сенс, і тепер став позначати тільки програмну частину від початкової системи автоматизованого програмування. Зараз здебільшого вживається у значенні деякої програми призначеної для автоматизації технологічного процесу проєктування виробу, результатом якого є набір проєктно-конструкторської документації, достатньої для виготовлення та подальшої експлуатації деталі

На сучасному ринку існує велика кількість САПР, які дозволяють вирішувати різні задачі. Умовно групу програм можна розділити на два типи:

– базові або ж легкі САПР призначені для двовимірного проектування та креслення, а також для створення деяких окремих тривимірних моделей без можливості роботи зі складальними одиницями.

– складні або ж важкі САПР призначені для роботи із складними виробами, наприклад складними конструкціями авіабудування, суднобудування та інше. Функціонально мають розширений набір вирішуваних задач, іншу архітектуру та алгоритми роботи у порівнянні із легкими.

Система автоматизованого проєктування і розрахунків технологічних процесів (САПР ТП, *CAPP*) – допоміжні програмні продукти для автоматизації процесу технологічної підготовки виробництва, а саме: планування технологічних процесів та оформлення відповідної технологічної документації. Основним завданням САПР ТП є складання плану виробництва, маршруту виготовлення виробу по його електронній моделі. У цей маршрут мають включатись відомості про послідовність технологічних операцій виготовлення деталі, режими здійснення технологічних операцій та інше. Система САПР ТП є елементом, що сполучає САПР і АСТПВ.[4]

Автоматизована система технологічної підготовки виробництва (АСТПВ, *CAM*) – система для підготовки технологічного процесу виробництва виробів, орієнтована на використання ЕОМ. Під терміном розуміють як сам процес комп'ютеризованої підготовки виробництва, так і програмно обчислювальні комплекси. Фактично етап технологічної підготовки зводиться до автоматизації програмування устаткування з ЧПК, генерації керуючих програм. [5]

Підсистема керування виконуючими пристроями – підсистема є основною частиною загальної системи числового програмного керування. З одного боку вона зчитує керуючу програму та інтерпретує їх в аналогові сигнали та віддає різним агрегатам верстата. З іншого – дозволяє верстату часткового взаємодіяти із людиною оператором, дозволяючи контролювати процес обробки. Існують закриті та відкриті (ПК-сумісні) системи керування.

Закриті системи керування мають власні алгоритми обробки, власні методи. Виробники закритих систем, в більшості випадків не надають інформацію про їх структуру. В таких випадках практично неможливо оновлювати та редагувати програмне забезпечення. Водночас закриті системи мають суттєву перевагу – високу ступінь надійності, оскільки усі компоненти системи тестуються на предмет сумісності.

ПК-сумісні системи мають вищий ступінь відкритості. Їх апаратні складові такі ж як і у звичайного побутового комп'ютера. Перевагою такого типу систем є доступність та відносно невелика вартість електронних компонентів, проте прийнято вважати, що надійність таких систем поступається надійності закритих систем. [1]

1.2. Аналіз статистики використання засобів генерації керуючих програм

Для розуміння ситуації в цілому проаналізовано ситуацію на ринку ЧПК верстатів. Очікується, що ринок буде розвиватись через зростаючий попит на підвищення ефективності обробки складних виробів. Крім того, існує тенденція на збільшення потреби точності обробки і зниження експлуатаційних витрат, що позитивно впливає на ріст популярності верстатів з ЧПК. Також спостерігається сплеск масового виробництва в різних галузях промисловості, що додатково пояснює ріст попиту на пристрої ЧПК.

Навіть враховуючи вплив пандемії *Covid* 19, що різко вразила виробничі сектори по всьому світу, виникли зупинки виробничих підприємств, зриви ланцюгів поставок та дефіцит робочої сили, раптово зріс попит на ЧПК системи у секторах охорони здоров'я та фармацевтики. В підсумку рівень попиту на промислову автоматизацію зберігається і очікується стабільний ріст розмірів ринку ЧПК пристроїв.

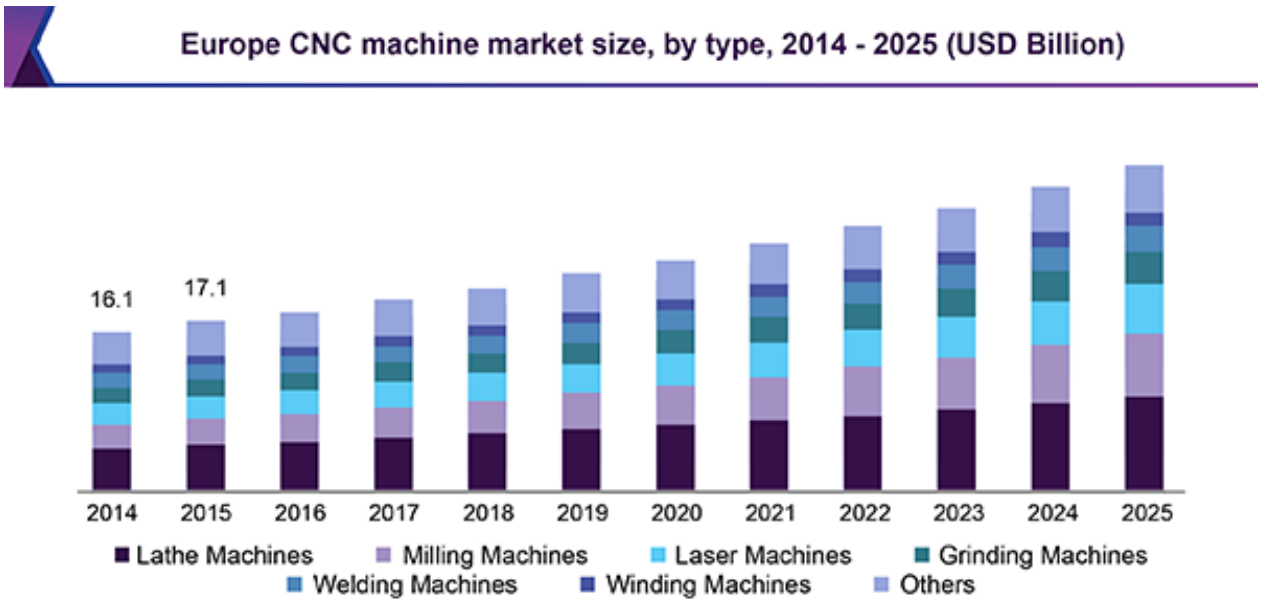


Рис. 1.1. Графік прогнозу розмірів та розподілу ринку верстатів з ЧПК Європейського регіону до 2025 року

Обсяг глобального ринку верстатів з ЧПК становив 82.40 мільярдів. доларів США в 2019 році і за прогнозами досягне 117.65 мільярдів доларів США до 2027 року, отже показник сукупного середньорічного темпу росту сягає 5.3 відсотка за прогнозований період. На рис. 1.1 можна бачити графік прогнозування розмірів ринку верстатів ЧПК Європейського регіону до 2025 року, який тільки підтверджує прогнози росту світового ринку пристроїв ЧПК. [5]

Важливо звернути увагу на розподіл ринку, більша частина якого припадає на токарні, фрезерувальні та лазерні верстати, трохи менше припадає на шліфувальні, зварювальні та намотувальні машини область використання яких не викликає питань, на відміну від частки «Інші» це категорія що займає приблизно 1/6 всього ринку і включає величезну кількість видів верстатів з ЧПК, саме для обслуговування цієї категорії призначена система, яка є об’єктом реалізації дипломного проєкту.

Як підсумок можна сказати, що ринок пристроїв з ЧПК і відповідно програмного забезпечення для їх обслуговування, стабільно зростає з кожним роком, не зважаючи на неочікуваний негативний вплив ситуації з промисловістю в світі.

На сьогоднішній день розробка керуючих програм здебільшого виконується з використанням спеціальних цілісних комплексів або ж з допомогою окремих систем автоматизованого програмування (САПР, САПР ТП, АСТПВ).

Загалом групи програмного забезпечення для ЧПК складають досить фрагментований ринок розподіл якого зображено на рис. 1.2. На рис. 1.2 зліва продемонстровано розподіл між п’ятнадцятьма найпопулярнішими пакетами ПЗ сектору у 82.6 відсотків всього ринку. На рис. 1.2 справа зображено розподіл між менш відомими виробниками що в сумі займають залишкові 17.4 відсотки ринку.

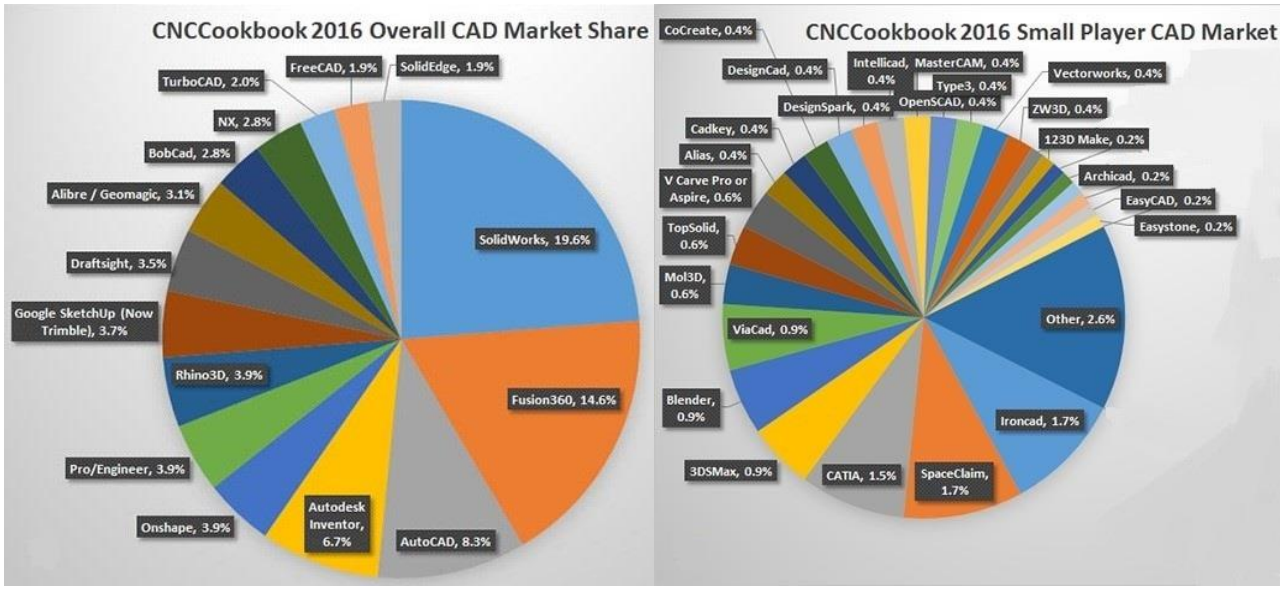


Рис. 1.2. Розподіл ринку *CAD* та *CAM* пакетів на період 2016 року

Умовно ринок прийнято поділяти на декілька сегментів (рис. 1.3):

– професійний сегмент повнофункціональних досить коштовних пакетів таких як «*Solidworks*»;

– середній сегмент пакетів з достатньо широким функціоналом, ці пакети теж платні, але не такі дорогі як сегмент професійних пакетів;

– безкоштовні модулі. Завжди існує набір безкоштовного програмного забезпечення розроблюваного ентузіастами;

– окремі пакети *CAM*. Окремі платні модулі призначені для використання разом із САПР.

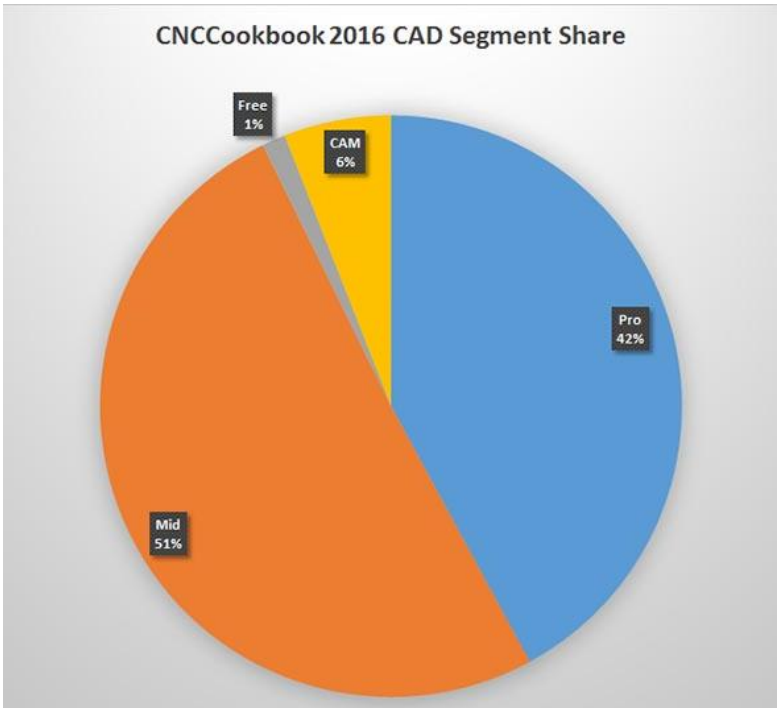


Рис. 1.3. Частка кожного з сегментів ринку ПЗ для ЧПК на період 2016 року

Відповідно до розподілу на рис. 1.3 частка середнього сегменту ПЗ для ЧПК найбільша і сягає 51 відсотку, до цього списку входять такі системи як «*Fusion 360*», «*AutoCad*», «*Onshape*», «*Drafsight*» та інші. За ним слідує сегмент професійних пакетів САПР у обсязі 42 відсотки до яких можна віднести: «*SolidWorks*», «*Autodesk Inventor*», «*Pro/Engineer*», «*Unigraphics NX*» та інші. Наступний сегмент *CAM* модулів в обсязі 6 відсотків серед яких: «*BobCad*», «*TopSolid*», «*MisterCAM*», «*ZW3D*» та інші. Останнім по популярності все ж є сегмент безкоштовного ПЗ для ЧПК в розмірі 1 відсотку, серед безкоштовних пакетів найпопулярнішими є «*123D Design*», «*123D Make*», «*Emachineshop*», «*HeeksCAD*», «*NanoCAD*» та «*OpenSCAD*». [6]

Варто зазначити, що більшість платних пакетів надає можливість користування протягом певного періоду часу, зазвичай 30 днів. Деякі компанії дозволяють використання навчальних версій для студентів. Але обидва варіанти не дають можливості повноцінно користуватись комплексами в потрібний для користувача час. Також на час написання пояснювальної записки з переліку безкоштовних пакетів зазначених пакетів залишились лише «*OpenSCAD*» та «*Emachineshop*», всі інші компанії або закінчили підтримку пакетів (як наприклад

Autodesk припинив підтримку пакету *123D*), або після досягнення популярності серед користувачів перейшли на платну основу.

В дослідженнях наступних років дослідники додають до сегменту безкоштовних пакетів САПР суму відсотків використаних пробних версій платних пакетів, навчальні версії та частку використовуваного піратського програмного забезпечення, які в підсумку на 2020 рік сягають близько 20 відсотків від загалу.

З описаних вище даних можна зробити висновок про відчутний дефіцит безкоштовних пакетів (САПР, САПР ТП, АСТПВ) в умовах наростаючого попиту, що змушує людей використовувати навчальні або ж піратські версії платного ПЗ.

Розглянемо декілька прикладів пакетів програм для роботи з ЧПК, для встановлення функціональних вимог розроблюваного продукту.

Програма «Конструктор *G*-кода, версия 3.3.0.43» представляє собою щось на зразок конструктора, що складається із набору блоків які можна зібрати в певну послідовність. Функціонал дозволяє адаптувати програму під конкретний ЧПК верстат. Наявна версія знаходиться в процесі розробки. Розробник має особистий веб сайт для зв'язку та пропозицій, що є плюсом. Головне вікно програми зображено на рис. 1.4.

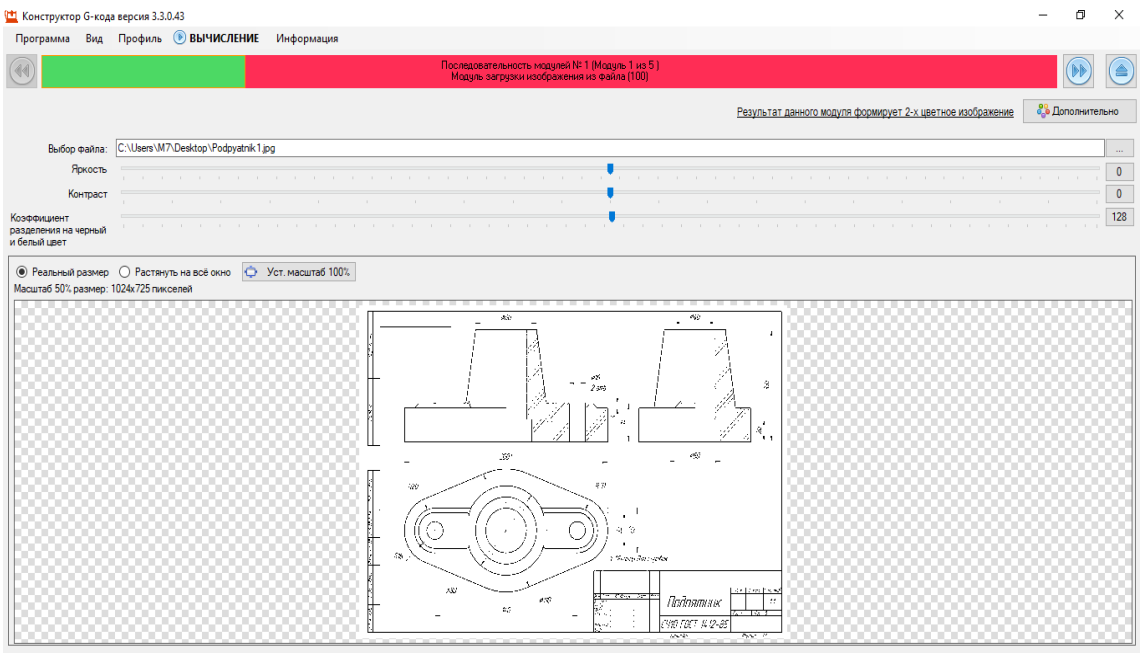


Рис. 1.4. Головне вікно програми «Конструктор *G*-кода»

Структура програми реалізована у вигляді класичного додатку майстра, що крок за кроком веде користувача до виконання поставленої задачі. З однієї сторони це дозволяє уникнути зайвих проблем під час взаємодії користувача з додатком, з іншої дещо обмежує його в послідовності виконання операцій. Також розглянутий комплекс має недостатню кількість засобів для початкового редагування зображень (розмиття, накладення фільтрів). Відсутній модуль для встановлення зв'язку з пристроєм ЧПК. Приклад вікна налаштувань одного з кроків майстра налаштувань зображено на рис. 1.5. Варто додати про відсутність української локалізації програми.

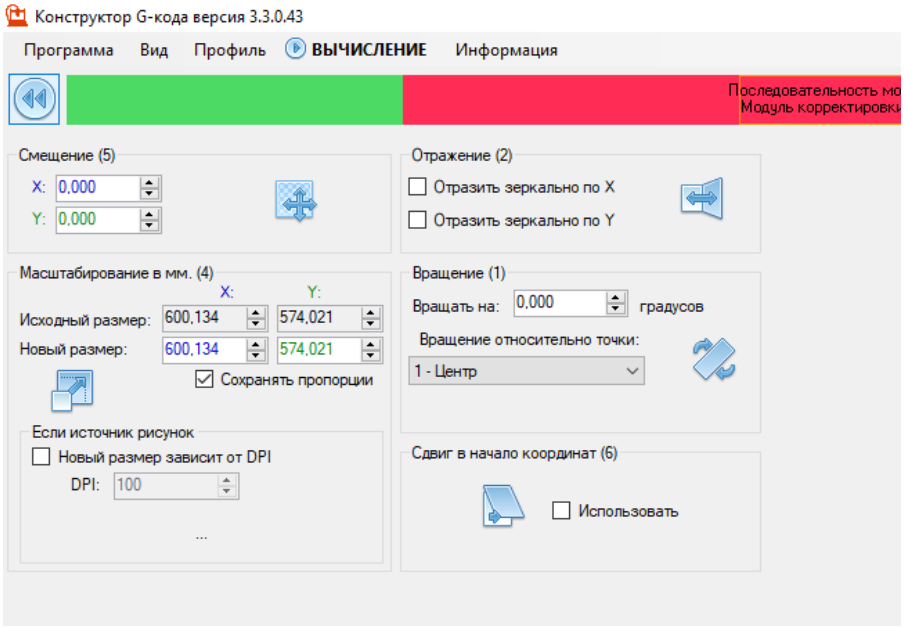


Рис. 1.5. Вікно налаштувань корегування векторних даних програми «Конструктор *G*-кода»

Наступним розглянемо онлайн сервіс, що дозволяє генерувати керуючі програми, його можна знайти за адресою «*ecnc.ru*». Сервіс призначений для створення програми управління ЧПК із встановленим лазерним модулем на різних поверхнях дерево, фанера, скло, метали. Для гравіювання на склі передбачений режим псевдо тонування який дозволяє змоделювати ефект напівтонів з двох кольорів: чорного і білого. Дозволяється вибір розмірів готового зображення, визначається відповідність між растровим зображенням та заготовкою. Генератор

генерує тільки тіло програми. Швидкість переміщення і початкове та кінцеве положення треба вказувати самостійно. Сервіс не потребує реєстрації, надається безкоштовно та зберігає програми доступними для завантаження на протязі однієї години.

Рис. 1.6. Зовнішній вигляд онлайн сервісу «*ecnc.ru*»

Недоліками даного сервісу можна назвати надзвичайно низьку кількість можливих налаштувань системи та залежність від стабільності інтернет зв’язку, оскільки встановити сервіс на комп’ютер і використовувати його в офлайн режимі

неможливо. З іншого боку згадані недоліки забезпечують простоту користування і гарний результат у випадку підходящих обставин.

Програма «*G-code Generator by Opr Lasers v 1.11*» призначена для створення керуючих програм для верстатів однойменної компанії, на відміну від аналогів розповсюджується окремо від обладнання і на безкоштовній основі. Одне із вікон цієї програми зображено на рис. 1.7.

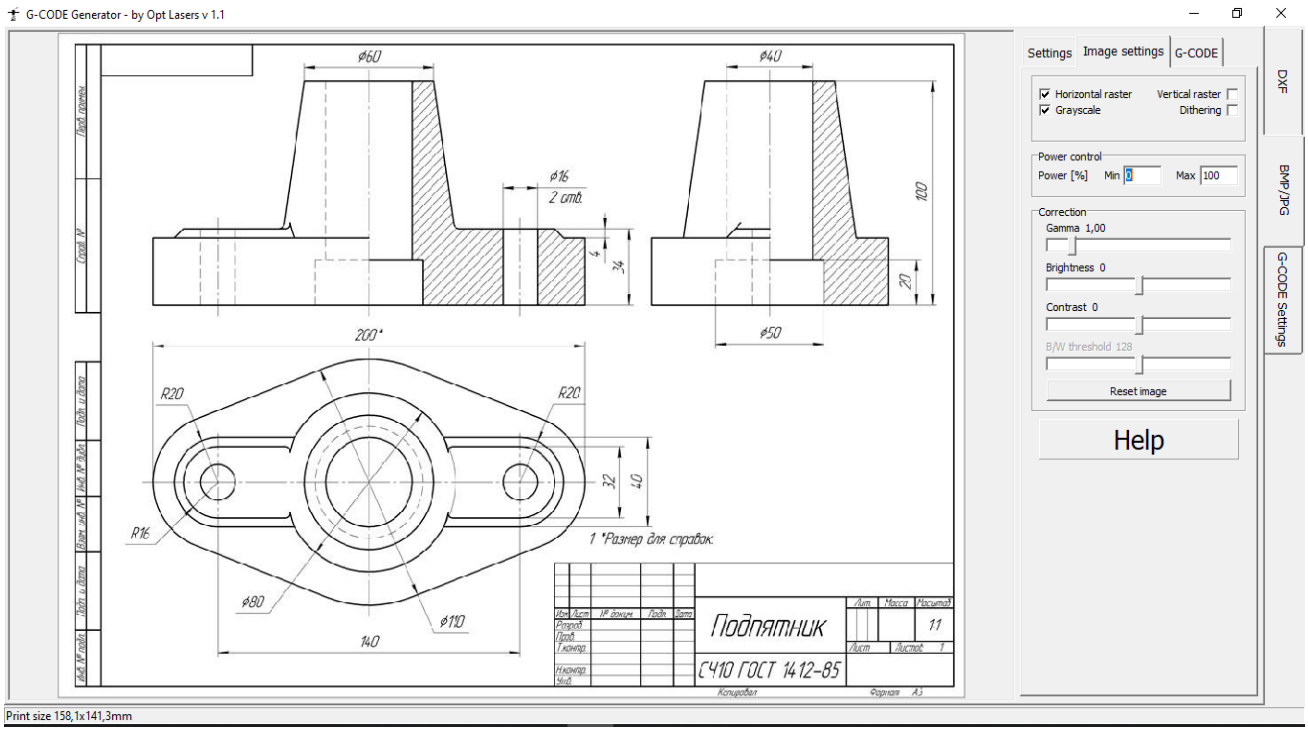


Рис. 1.7. Вікно генерації *G*-кодів із растрових зображень

Мінусом є відсутність можливості налаштування параметрів верстату, що пояснюється задумкою використання тільки для пристроїв однієї компанії. Іншим недоліком є суттєва обмеженість для редагування вхідних зображень. Також відсутня будь яка локалізація окрім англійської, що теж є недоліком. Отже після генерації керуючої програми її необхідно, додатково редагувати, також в програмі відсутній засоби для зв’язку з верстатом.

В підсумку можна сказати, що вибір безкоштовних пакетів для генерації керуючих програм для ЧПК пристроїв суттєво обмежений, можливі використання демонстраційних версій та версій для навчання від серйозних корпорацій, що не

дуже зручно. Для знаходження безкоштовних версій необхідно затратити багато часу для підбору та ресурсів для тестування. Існуючі рішення мають велику кількість недоліків як суттєвих так і не дуже, хоча при цьому виконують свої функції задовільно. Варто зазначити, в списку присутня лише одна програма з відкритим вихідним кодом, а саме «Конструктор *G*-кода».

1.3. Висновки до розділу

Проаналізовано становище кожної із складових пакетів для ЧПК на наш час. Після пошуків в глобальній мережі можна дійти висновку про те, окремих систем САПР (*CAD*) що можуть повноцінно виконувати свої функції є достатня кількість, отже необхідності реалізації в системі перетворення керуючих програм повноцінного *CAD* модуля немає, достатньо реалізувати частковий фунціонал редагування вхідної інформації для зручності роботи з нею і уникнення необхідності одночасного використання декількох програм.

САПР ТП – система, реалізацію якої нечасто можна знайти у вигляді окремого додатку, оскільки специфіка функціональних задач поставлених перед цією системою передбачає одночасну наявність самого *CAD* модуля, або ж результатів його роботи і АСТПВ. Отже необхідно включити основні функції САПР ТП до розроблюваного продукту.

АСТПВ – система, безкоштовних високоякісних реалізацій якої вкрай невелика кількість. Реалізація основних функцій цього модуля теж необхідна в підсистемі перетворення керуючих програм.

Підсумовуючи можна сказати що завданням для проектування є розробка програмної реалізації *CAD/CAPP/CAM* системи «*CNC Control*» – програмного забезпечення, призначеного для:

- створення та редагування простих зображень;
- автоматизації побудови, розрахунків траєкторій переміщення інструменту згідно траєкторій креслення та процесів проектування обробки виробів;

- генерації управляючих програм для роботи пристроїв з ЧПК;
- автоматизації процесів підготовки та управління виробництвом продуктів пристроями із ЧПК.

Опціонально можна виділити створення драйверу для зв'язку із системою керування виконуючими пристроями, такі системи існують із знаходяться у вільному доступі, з можливістю ручного налаштування параметрів верстатів з ЧПК для індивідуального використання.

Основними цілями впровадження системи є:

- утворення цілісного програмного комплексу разом із системним програмним забезпеченням для пристроїв з ЧПК;
- досягнення зменшення затрат різних видів ресурсів для виготовлення різноманітних продуктів засобами пристроїв з ЧПК, за рахунок автоматизації значної частини процесів виробництва;
- скорочення трудомісткості проектування і планування, скорочення собівартості проектування та виготовлення, зменшення затрат на експлуатацію, підвищення якості і технічно-економічного рівня проектування і виготовлення;
- забезпечення поширення використання у виробництві різних галузей пристроїв із використання ЧПК;
- досягнення балансу між універсальністю в плані можливостей застосування програмного забезпечення із різними типами апаратного забезпечення та об'ємом знань необхідних для комфортної роботи із програмою.
- формування відносно простого програмного комплексу із можливістю вільного, некомерційного використання у різноманітних галузях де доцільно використовувати ЧПК технології .

Розглянуту інформацію враховано для визначення сильних та слабких сторін існуючих рішень, оцінено проблеми які потрібно вирішити та їх актуальність. Відповідно після аналізу програмних продуктів наявних на ринку, їх популярності та актуальності, для отримання адекватного уявлення про те, які завдання має виконувати програма сформульовано задачі для виконання в ході дипломного проектування.

Розглянуто основні поняття та терміни що використовуються в системах з числовим програмним керуванням, підвищено рівень обізнаності в галузі.

Проаналізовано дані статистки використання верстатів з ЧПК, програмного забезпечення для роботи з пристроями ЧПК, проведено їх класифікацію. Як результат визначено дефіцит комплексів для перетворення керуючих програм, що розповсюджуються на безоплатній основі, сформульовано базову структуру підсистеми генеруючих програм.

Відповідно до сучасних тенденцій, сформульоване завдання проектування з урахуванням результатів роботи всіх попередніх етапів розробки сформульовано завдання проєктування, а саме:

– проаналізувати сучасні програмні рішення та методи і технології їх реалізації;

– розробити структуру підсистеми перетворення керуючих програм;

– розробити основні алгоритми роботи підсистеми перетворення керуючих програм;

– провести програмну реалізацію розробленого алгоритму;

– розробити інструкцію користувача;

– провести тестування підсистеми перетворення керуючих програм.

РОЗДІЛ 2
ПІДСИСТЕМА ПЕРЕТВОРЕННЯ КЕРУЮЧИХ ПРОГРАМ

2.1. Функціональні вимоги

Оскільки система структурно складається з декількох різних модулів, з урахуванням результатів роботи попередніх кроків проєктування, до модулів програмної реалізації підсистеми перетворення керуючих програм поставлені наступні функціональні вимоги:

Модуль САПР (*CAD*) має реалізовувати функції:

– завантаження, відображення, створення, редагування та збереження графічних документів групи форматів «*.BMP*; *.GIF*; *.JPG*; *.PNG*; *.TIF*»;

– забезпечення можливості редагування завантажених документів за допомогою наступного набору інструментів: масштабування (за відносними та абсолютними розмірами), поворот зображення в різні сторони на довільну кількість градусів, віддзеркалення по вертикалі та віддзеркалення по горизонталі, зміна яскравості та контрасту, накладення чорно-білого фільтру, виконання розмиття за методами середнього та медіанного розмиття, накладення фільтру «негатив»;

– забезпечення зрозумілого інтерфейсу користувача у вигляді елементів керування, «збільшити», «зменшити» – для масштабування робочої області із зображенням; «відкрити», «зберегти», «зберегти як» – для взаємодії з файловою системою користувача; «вихід», «згорнути», «розгорнути» – як елементи керуванням вікном програми; «назад», «вперед» для орієнтації користувача у процесі застосування команд;

– забезпечення можливості показу оригінального зображення не зважаючи на накладені функції редагування та фільтри.

Модуль САПР ТП (*CAPP*) має включати засоби реалізації наступних функцій:

– побудова плану виготовлення виробу, що називають операційною або маршрутною картою. Цей план включає в себе послідовність технологічних операцій, використовувані інструменти, різновиди режимів обробки та використовувані технічні процеси;

– конвертація вхідної інформації у вигляді графічного зображення (креслення) та вказаних налаштувань у маршрутну карту траєкторій руху інструменту для обробки виробу;

– візуалізація створеної маршрутної карти та додаткової пояснювальної інформації у текстовому, табличному форматі або ж у вигляді графічного зображення;

– можливості для редагування створеної маршрутної карти у разі її не валідності на думку користувача.

Модуль АСТПВ (*CAM*) має включати засоби реалізації наступних функцій:

– функція формування управляючої програми згідно маршрутної карти з допомогою стандартизованих засобів управління пристроями ЧПК, наприклад *G*-кодами;

– функції відображення у текстовому форматі, збереження згенерованих керуючих програм у файловій системі користувача;

– використання інтерфейсу модуля драйвера для встановлення підключення із фізичним пристроєм, моніторинг та контроль отриманих від пристрою даних.

Модуль-драйвер має виконувати такі функції:

– встановлення з’єднання з пристроєм;

– передача керуючих програм встановленими каналами зв’язку;

– отримання інформації про перебіг процесу та її інтерпретація.

2.2. Інструментальні засоби розробки

Основним інструментальним засобом під час розробки програмної реалізації підсистеми перетворення керуючих програм можна назвати мову програмування, засобами якої вона буде написана. Розглянуто найпопулярніші мови програмування, потенційно придатні для використання в дипломному проєкті, їх опис наведено далі.

C++ – одна із фундаментальних мов програмування. Швидка, дозволяє займатись оптимізацією, архітектурою програмного забезпечення, вирішенням задач по автоматизації процесів, створенням систем моделювання. В більшості випадків дозволяє реалізувати «з нуля» проєкти, що тісно взаємодіють з апаратною частиною ЕОМ. Є досить складною в застосуванні, не має вбудованого збирача сміття, стандартні бібліотеки підтримують обмежений набір функцій. Алгоритми реалізовані засобами мови *C++* потребують прискіпливого ставлення до розподілення та управління пам'яттю. Часто технології які дозволяють розробляти віконні додатки мовою *C++* піддають критиці, в основному через недосконаліть засобів які призводять до суттєвого уповільнення процесу розробки додатків.

Мова програмування *Java* – з самого початку позиціонується як універсальне та надійне рішення. Дозволяє розробляти програми для вирішення широкого спектру завдань, що і є її основним мінусом. Додатки написані мовою Java вимагають використання *JVM* (*Java Virtual Machine*). Більшість додатків розроблених на Java не використовують рідний інтерфейс *Win* 32, що є мінусом.

Мова програмування *C#* – об'єктно орієнтована мова програмування з безпечною системою типізації для платформи *.NET*. Синтаксис *C#* близький до *C++* і *Java*. На відміну від *C++* мова виключає потенційно небезпечні речі для роботи з пам'яттю, використовує процес інтерпретації вихідного коду, що робить додатки написані цією мовою мультиплатформенними. Частково успадкував від *C++* можливості для взаємодії з апаратним забезпеченням комп'ютера. Має вбудований збірник сміття, що значно прискорює і спрощує розробку. [8]

Проаналізувавши можливості, позитивні та негативні сторони різних мов програмування, визначено, що для реалізації підсистеми перетворення керуючих програм добре підходить мова *C#*, оскільки вона має добре розвинену базу сторонніх бібліотек, збірник сміття, адаптована для розробки віконних додатків для *Windows*.

Наступним кроком визначено інтегроване середовище програмування. *Microsoft Visual Studio* – серія продуктів фірми Майкрософт, які включають інтегроване середовище розробки програмного забезпечення та ряд інших інструментальних засобів. Ці продукти дозволяють розробляти як консольні програми, так і програми з графічним інтерфейсом, в тому числі з підтримкою технології *Windows Forms*, а також веб-сайти, веб-застосунки, веб-служби як в рідному, так і в керованому кодах для всіх платформ, що підтримуються *Microsoft Windows, Windows Mobile, Windows Phone, Windows CE, .NET Framework, .NET Compact Framework та Microsoft Silverlight*.

Visual Studio 2017 – перший нестабільний випуск версії програми випущено 30 березня 2016 року. Основними змінами стали інтерфейс інсталятора та численні незначні покращення у різних компонентах середовища розробки. Версія *Community* передбачає безкоштовне некомерційне використання і цілком задовольняє вимоги передбачені процесом дипломного проєктування.

.NET Framework – програмна платформа, створена компанією *Microsoft*. Основою якої є загальномовне середовище виконання *Common Language Runtime* (*CLR*), що підтримує різні мови програмування. Технологія розрахована для роботи під операційними системами сімейства *Microsoft Windows*. Основою ідеї при розробці фреймворку було забезпечення свободи розробника за рахунок надання йому можливостей створювати додатки різних типів, що здатні виконуватись на різних типах пристроїв в різних середовищах. Включає в себе інтерфейс програмування *Windows Forms*, що відповідає за графічний інтерфейс користувача і спрощує доступ до елементів інтерфейсу *Microsoft Windows* за рахунок обгортки *Win32 API* в керованому коді.

2.3. Структура програмного модуля

Аналіз сучасних програмних засобів показав, що модуль має бути реалізований у вигляді програмного додатку для ПК під управлінням операційної системи *Windows*, його проект буде включати в себе наступний набір файлів:

– набір файлів для реалізації головного вікна користувацького інтерфейсу та точки запуску програми: *Program.cs*, *MainForm.cs*, *MainForm.Designer.cs*, *MainForm.resx*;

– файл для реалізації засобів кешування дій виконаних користувачем – *Cache.cs*;

– набір файлів для реалізації вікна користувацького інтерфейсу, методів функцій масштабування та повороту зображень: *Rotation.cs, Rotation.Designer.cs*, *Rotation.resx*, *Rotate.cs*, *Scaling.cs*, *Scaling.Designer.cs*, *Scaling.resx*, *ScalingAlgorithms.cs*;

– набір файлів для реалізації вікна користувацького інтерфейсу та методів функцій відображення інформації про зображення: *Scaling.cs*, *Scaling.Designer.cs*, *Scaling.resx, ScalingAlgorithms.cs*;

– файл для реалізації функції віддзеркалення зображення – *Mirror.cs*;

– набір файлів для реалізації меню роботи з кольорами, а саме вікно користувацького інтерфейсу для перегляду графіків розподілення кольорів в зображенні, або ж гістограми інтенсивності, вікно користувацького інтерфейсу для регулювання яскравості і контрасту зображення, вікно користувацького інтерфейсу для застосування чорно-білого фільтру, статичний клас для підтримки функціонального забезпечення вище згаданих вікон: *Histogram.cs*, *Histogram. Designer.cs*, *Histogram.resx*, *BrightnessContrast.cs*, *Brightness Contrast.Designer.cs*, *Brightness_Contrast.resx*, *Threshold.cs*, *Threshold.Designer.cs*, *Threshold.resx*, *Colors.cs*;

– набір файлів для реалізації меню роботи з фільтрами, а саме – екранна форма користувацького інтерфейсу для накладення фільтрів виділення контурів, функціональна підтримка згаданої форми та реалізація функцій накладання

розмиття і «негативного» фільтру на зображення: *EdgeDetection.cs*, *EdgeDetection.Designer.cs*, *EdgeDetection.resx*, *ImageFilters.cs*;

– набір файлів для реалізації вікна користувацького інтерфейсу та методів його функціональної підтримки, призначених для створення файлів заповнених послідовностями команд мовою *G*-кодів: *GCODE.cs*, *GCODE. Designer.cs*, *GCODE.resx*, *GCODE_methods.cs*;

– набір файлів для реалізації вікна користувацького інтерфейсу для доступу до інтерфейсу зв'язку з пристроєм ЧПК: *USB.cs*, *USB.Designer.cs*, *USB.resx*.

Структуру підсистеми перетворення керуючих програм зображено на структурній схемі системи відтворення креслень (рис. 2.1). Відповідно до схеми розглянемо всі модулі підсистеми.

Модуль САПР підсистеми перетворення керуючих програм призначений не стільки для проведення повного циклу розробки креслення, як для підготовки вже раніше розроблених документів до побудови маршрутної карти. Саме для цього слугують визначені в попередніх розділах функції. Засоби масштабування покликані синхронізувати розміри зображення та розміри наявної в ЧПК верстаті робочої області, схожу, але дещо іншу функцію містить вікно «Генерація керуючої програми», яке дозволяє обрати коефіцієнт відношення пікселів зображення до міліметрів робочої області верстату. Вікно інформації та вікно з діаграмою *RGB* забезпечує користувача даними про конфігурацію зображення. Опції віддзеркалення дозволяють правильно обрати конфігурацію зображення для відтворення. Форми «Яскравість та контраст» та «Чорно-білий фільтр» дозволяють змінювати відповідно яскравість, контраст та конфігурацію кольорів зображення, це необхідно для покращення якості виокремлення контурів. Контекстне меню «Фільтри» дозволяє застосовувати фільтри які в багатьох ситуаціях відчутно підвищують якість визначення контурів під час генерації керуючої програми. Різні методи знаходження контурів по різному можуть діяти у різних ситуаціях, це дозволяє підвищити якість генерації масивів контурів під час обробки різних зображень, забезпечує гнучкість та універсальність програми.

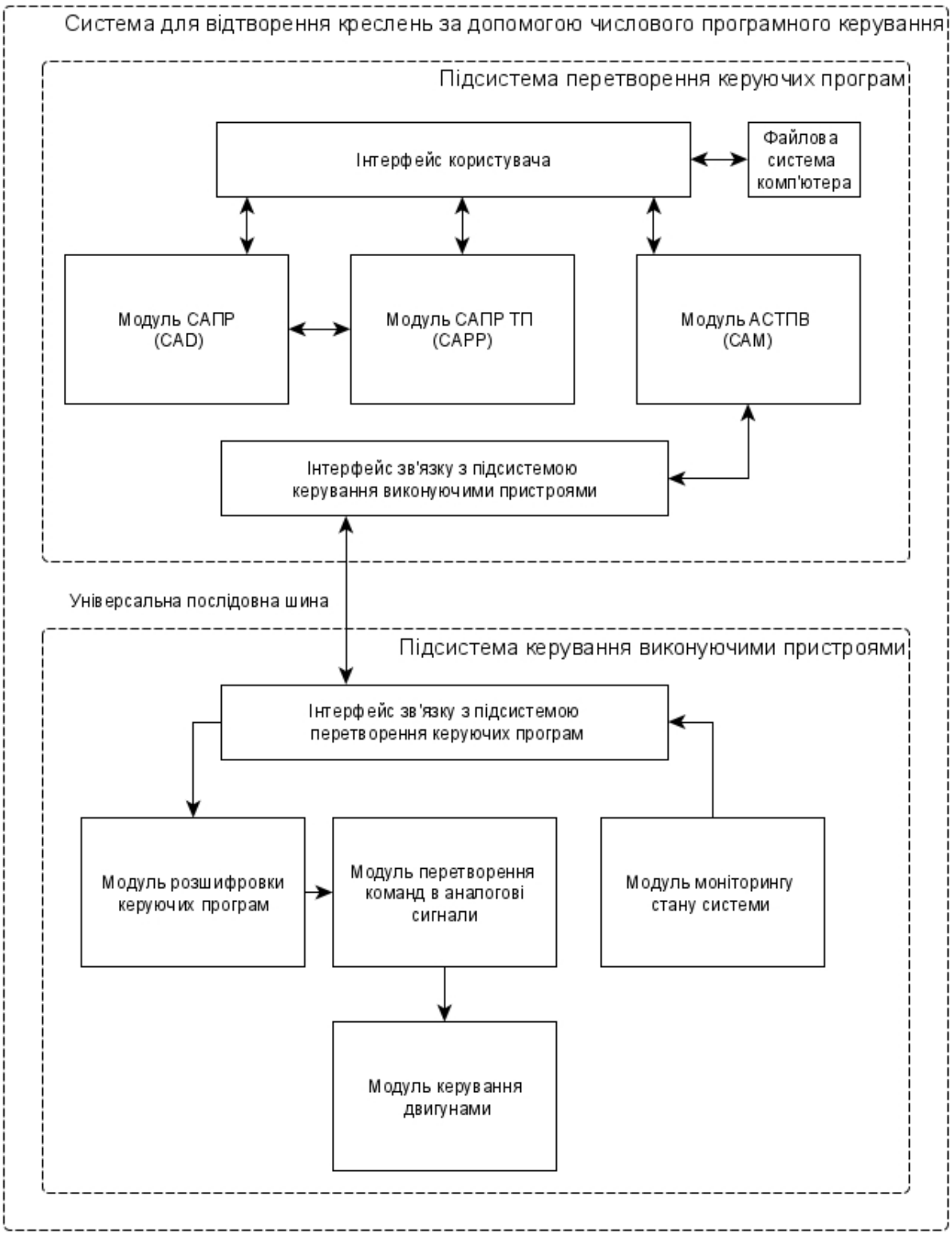


Рис. 2.1. Структурна схема системи відтворення креслень

Інтерфейс для АСТПВ реалізується вікном «Генерація керуючої програми», що містить три основні групи елементів керування. Налаштування генератора контурів дозволяють обрати нижню та верхню межі чутливості генератора, ці

параметри впливають на чутливість генератора до пікселів з різною інтенсивністю кольору, відповідно якщо значення інтенсивності не входить в діапазон генератора, точка до контуру не буде включена. Товщина контуру встановлює товщину контурів, які будуть згенеровані в пікселях. Параметр мінімальної довжини дозволяє виключити зі списку контури, що мають довжину менше заданого значення, це дозволяє ігнорувати «шуми» в зображенні. Апроксимація, параметр що дозволяє оптимізувати кількість точок в контурах, що описують правильні фігури проте іноді може дещо впливати на якість згенерованих контурів.

Панель «Налаштування керуючої програми» дозволяє налаштовувати режими обробки деталі, такі як швидкість руху в міліметрах на хвилину, глибина проникнення інструменту в мікрометрах, налаштування відношення пікселів до міліметрів для регулювання масштабу та встановлення зміщення на певну кількість міліметрів. Панель включає в себе список контурів, згенерованим генератором контурів та дозволяє включати або ж виключати їх із маршрутної карти програми, для візуального контролю та розуміння включених або виключених контурів зміни відображаються в робочій області головного вікна програми.

Застосування параметрів «Налаштування знаходження контурів» відбувається після запуску генератора, тоді ж і заповнюється список контурів зліва. Параметри «Налаштування керуючої програми» застосовуються після запуску генератора, тоді ж відкривається діалогове вікно з пропозицією зберегти згенеровану програму в тестовий файл в файловій системі комп’ютера. Це потрібно для подальшої його відправки на пристрій з ЧПК. На всіх екранних формах користувацького інтерфейсу присутні кнопки «Закрити» для їх закриття.

Екранна форма «Підключені пристрої» забезпечує користувача інтерфейсом для зв’язку з пристроєм ЧПК. Містить в собі групу елементів для вибору і налаштування порту з’єднання (номер порту, швидкість передачі, біт парності, біти даних, стопові біти). Кнопки для підключення та відключення від пристрою, поле для введення команд і кнопка для ручної передачі їх на пристрій, вибору файлу та передачі його на пристрій з ЧПК. Поле «Затримка передачі» дозволяє до і під час

передачі регулювати швидкість передачі команд. Частина вікна справа виводить на екран результати всіх дій програми.

Складено діаграму послідовностей яка зображена на рис. 2.2.

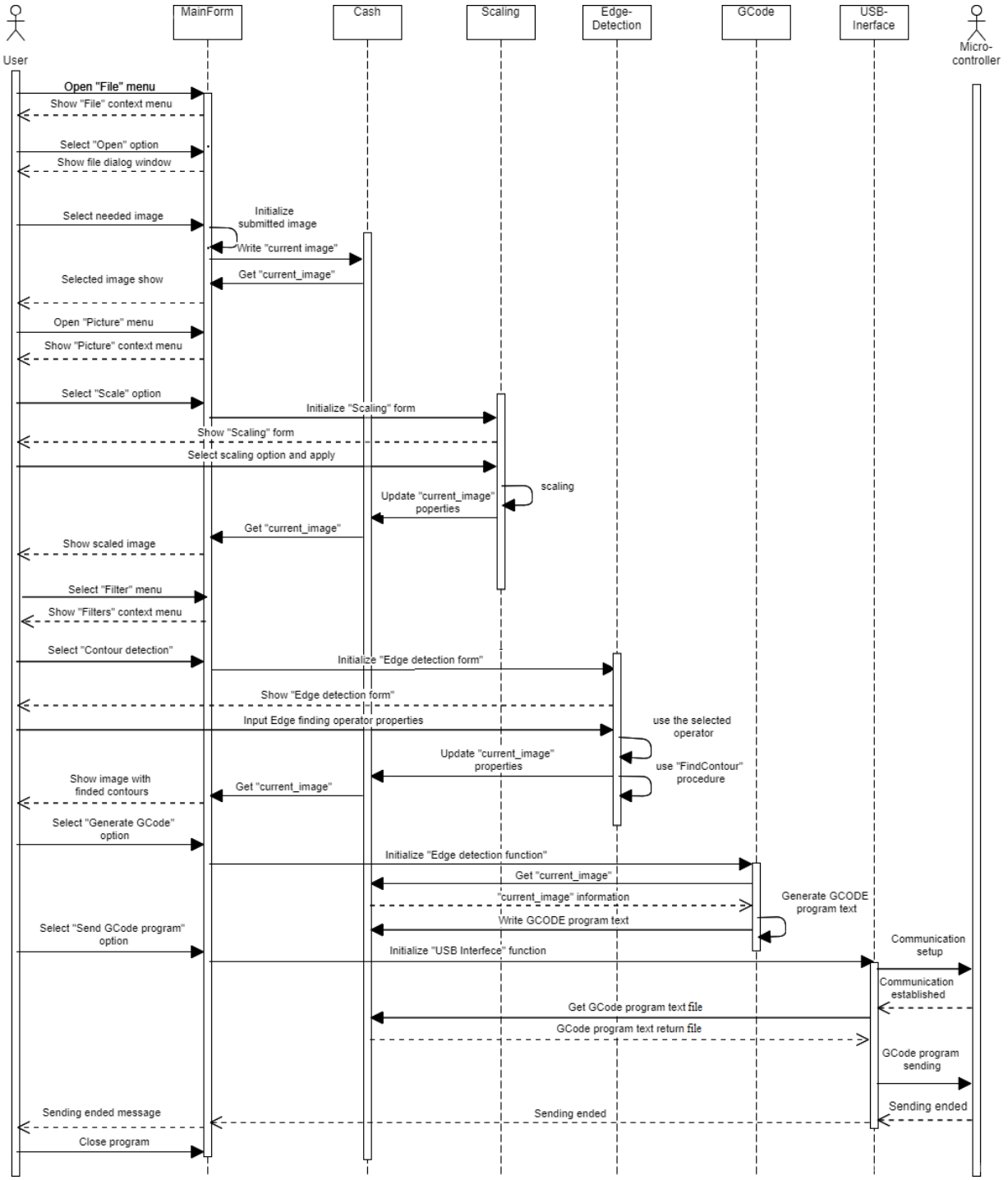


Рис. 2.2. Діаграма послідовностей для одного з варіантів використання підсистеми перетворення керуючих програм

Діаграма послідовностей (рис. 2.2) демонструє зв’язок між модулями та їх елементами підсистеми перетворення керуючих програм. Розглянуто один із основних варіантів використання підсистеми перетворення керуючих програм, а саме відкриття зображення, його масштабування, визначення контурів зображення, генерація керуючої програми, передача керуючої програми на пристрій ЧПК.

Наведена діаграма послідовностей демонструє зв’язки між елементами модулів підсистеми перетворення керуючих програм. Ініціатором роботи програми є користувач, який запускає головне вікно програми та в контекстному меню обирає пункту головного меню «Файл», після чого відкривається діалогове вікно файлової системи для вибору зображення. Наступним етапом на діаграмі показано, як після вибору пункту «Масштабування» контекстного меню пункту головного меню «Зображення» викликається вікно користувацького інтерфейсу «Масштабування» описане в класі *Scaling.cs*, важливо зауважити, що після виконання обраних користувачам функцій масштабування, зображення зберігається за допомогою класу *Cache.cs*, після чого передається в головну форму для відображення в робочій області. Після вибору користувачем пункту «Визначити контури» в контекстному меню пункту головного меню «Фільтри» викликається екранна форма описана в файлі *EdgeDetection.cs*. Наступним кроком після застосування фільтрів зображення знову зберігається за допомогою класу *Cache.cs* та виводиться в поле робочої області головного вікна програми. Виклик вікна для генерації програми за допомогою вибору пункту головного меню «Генерація керуючої програми» звертається до класу *GCODE.cs* для генерації керуючої програми і її збереження.

Виклик вікна роботи з драйвером передачі програми за допомогою вибору пункту головного меню «Підключені пристрої», головна екранна форма програми звертається до класу до *USB.cs* та виводить його вміст на екран, після чого зв’язується з класом *Cache.cs* та отримує шлях до файлу з керуючою програмою. Методи класу *USB.cs* зв’язуються з пристроєм ЧПК , після чого команди покадрово відправляються на нього. Після з передачі керуючої програми в обраному варіанті використання, застосування підсистеми завершується і вона закривається.

2.4. Висновки до розділу

1) Розглянуто результати аналізу статистики та сучасних засобі генерації керуючих програм, що дало змогу сформулювати функціональні вимоги до кожного з модулів підсистеми перетворення керуючих програм.

2) Проаналізовано сучасні мови програмування, технології та інтегровані середовища розробки. Обрано інструментальні засоби, які задовольняють потребам розробки підсистеми перетворення керуючих програм.

3) За допомогою розроблених функціональних вимог сформульовано список файлів, що включає в себе проект підсистеми перетворення керуючих програм.

4) Розроблено і описано структурну схему підсистеми перетворення керуючих програм у складі системи відтворення креслень, що містить такі основні складові: модуль САПР, модуль САПР ТП, модуль АСТПВ, інтерфейс користувача, інтерфейс зв’язку з підсистемою керування виконуючими пристроями.

5) За допомогою діаграми послідовності, описано зв’язки модулів підсистеми перетворення керуючих програм.

РОЗДІЛ 3
ПРОГРАМА ПЕРЕТВОРЕННЯ КЕРУЮЧИХ ПРОГРАМ

3.1. Системні вимоги

Для опису характеристик, яким має відповідати ЕОМ для коректної роботи розроблюваної підсистеми використовують системні вимоги, розрізняють мінімальні та рекомендовані системні вимоги. Сформульовано мінімальні системні вимоги до підсистеми перетворення керуючих програм, які включають:

1) 32 або 64-розрядний ноутбук чи настільний комп'ютер під управлінням операційної системи: *Windows Server Core 2012 R2; Windows Server 2019, 2016, 2012 R2; Windows 7 SP1; Windows 8.1; Windows 10, Version 1607, Version 1709, Version 1803, Version 1809; Windows 10 / Windows Server, Version 1903, Version 1909, Version 2004, Version 20H2*;

2) Наявність додаткового програмного забезпечення: *Microsoft .NET Framework 4.7*;

3) Процесор з тактовою частотою 2 *ГГц* або більше;

4) Оперативна пам'ять –1024 *Мб* або більше;

5) Вільне місце на диску – 512 *Мб*;

6) Відеокарта з 256 *Мб* оперативної пам'яті або більше;

7) Наявність *USB* інтерфейсу *A*, *B* або *C* типу з підтримкою версії *USB* 2.0 або вище;

8) Прилади взаємодії з інтерфейсом – монітор, клавіатура.

3.2. Алгоритми роботи підсистеми перетворення керуючих програм

Розглянуто основні алгоритми комп'ютерного зору, що використовуються для знаходження контурів зображення, до реалізації обрано наступний набір диференціальних операторів для двовимірних зображень: оператор Собеля, оператор Прюітт, перехресний оператор Робертса, ЛОГ фільтр та детектор контурів Кенні.

Диференціальний оператор – не неперервний, не обмежений і не лінійний оператор, визначений деяким диференціальним виразом і діючий в просторі векторнозначних функцій на об'єктах евклідового простору розмірності *n*, або спряжених до них просторів.

Під процесом фільтрації зображень розуміють операцію, яка у вигляді результату повертає зображення того ж розміру, отримане із вихідного за деякими правилами. Правила фільтрації можуть бути досить різноманітними, це одна із фундаментальних операцій комп'ютерного зору, розпізнавання образів та обробки зображень.

Згортка – процес додавання кожного елементу зображення до його сусідів, зважених ядром. Хоча і операція згортки позначається символом «*» це не звичайне множення, а операція знаходження суми добутків значень матриці згортки і відповідних пікселів вхідного зображення. У випадку зі знаходженням контурів використовується для оцінки градієнта функції інтенсивності *f(x, y)*.

Найзагальнішим способом пошуку контурів є фільтрація зображення за допомогою згортки, ядром згортки, що являє собою матрицю, відповідну вказаній групі пікселів вихідного зображення, елементи матриці часто називають коефіцієнтами. Процес просторової фільтрації для маски 3×3 зображено на рис. 3.1. Процес заснований на переміщенні ядра згортки від точки до точки зображення, в кожній точці результат згортки обчислюється за допомогою підрахунку заданих функцій. Наприклад для лінійної просторової фільтрації результат задається сумою добутків коефіцієнтів фільтру на відповідне значення пікселя в області покритій маскою.

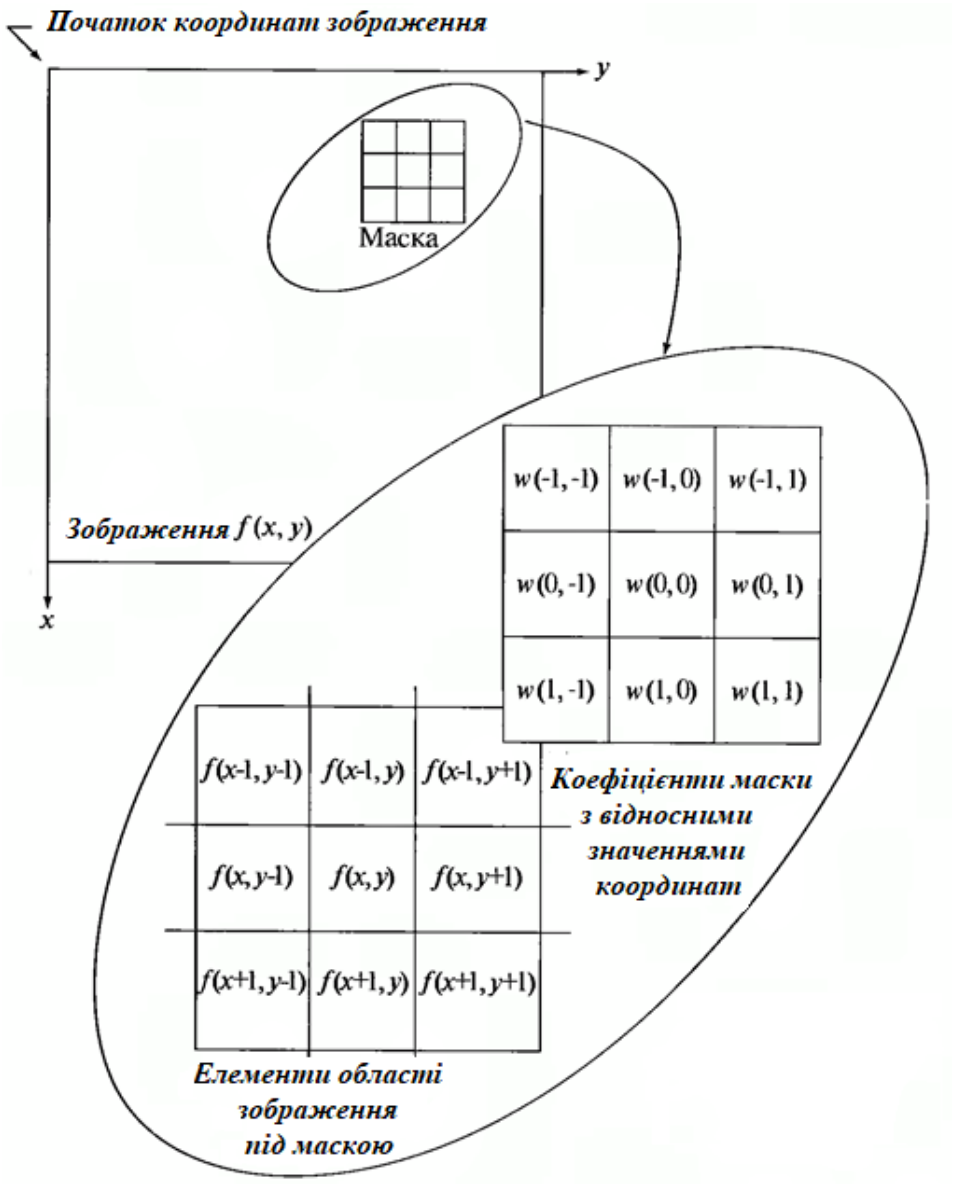


Рис. 3.1. Просторова фільтрація для маски 3×3

Використання диференціальних операторів для двовимірних зображень пов'язане зі знаходженням напрямку зміни контрасту двовимірної функції інтенсивності *f(x; y)*. Із математичного аналізу відомо, що максимальні зміни значень функції проходять уздовж осі направлення її градієнта. Вектор градієнта відповідно має координати $\left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$. На рис. 3.2 показано результат застосування поняття градієнту для знаходження значення і направлення контрасту зображення шляхом обрахунку модуля і направлення градієнта функції інтенсивності *f(x; y)* з використання дискретних значень із піксельного масиву зображення.

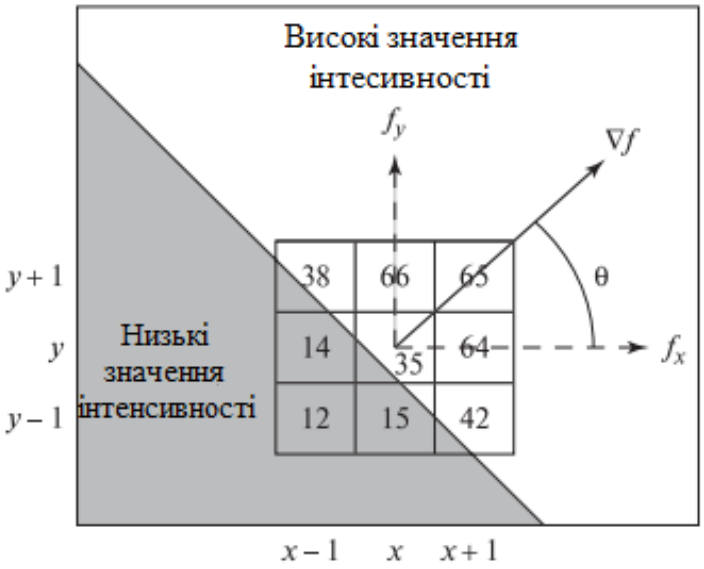


Рис. 3.2. Приклад оцінки величини і направленості контрасту в точці зображення

Для оцінки контрасту зображення в точці *I[x; y]* вздовж направлення осі *x* можна використати різницю *(I[x + 1; y] - I[x – 1; y]) / 2*. Це значення представляє зміну інтенсивності між лівим та правим пікселями точки, що розглядається. Наприклад, в околі точки що розглядається на рис. 3.2 контраст в направленні осі *x* буде дорівнювати (64 – 14) / 2 = 25.

Так як в значеннях пікселів може бути присутній шум, а також через те що край може перетинати піксельний масив під будь яким кутом для більш точної оцінки контрасту вздовж осі x можна виокремити три різні оцінки контрасту в околі пікселя *I[x; y]*, що і зроблено в формулі (3.1).

Приблизне значення контрасту зображення в направленні осі *x* обчислюється як середнє значення контрасту зображення вздовж рядка з координатою *y*, а також рядків, розміщених вище і нижче рядка *y*.

Приблизне значення контрасту зображення в направленні осі *y* можна визначити аналогічним способом, що і продемонстровано в формулі (3.2).

Для прискорення обрахунків ділення на 6 часто не виконується і дані оцінюються просто в збільшеному масштабі. Маски розглянутих операторів є визначенням для оператора Прюітт, названого в честь доктора Джудіт Прюітт що вперше використала його для визначення границь на медичних зображеннях. Ці та інші маски, позначені як M_x та M_y, а саме зверху – маска Прюітт, в середині – маска Собеля, знизу – маска Робертса, зображені на рис. 3.3.

Ядро Прюітт $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

Ядро Собеля $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

Ядро Робертса $M_x = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

Рис. 3.3. Маски, які використовуються для оцінки градієнта функції інтенсивності

$$\frac{\partial f}{\partial x} \approx \left(\frac{1}{6}\right)(M_x * N_8[x,y]), \tag{3.3}$$

$$\frac{\partial f}{\partial x} \approx \left(\frac{1}{6}\right)\left(M_y * N_8[x,y]\right), \tag{3.4}$$

$$|\nabla f| \approx \sqrt{\left(\frac{\partial f^2}{\partial x}\right) + \left(\frac{\partial f^2}{\partial y}\right)}, \tag{3.5}$$

$$\theta \approx arctg\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial y}\right). \tag{3.6}$$

Застосовуючи описані маски для деякого зв'язного околу $N_8[x,y]$ пікселя (x, y) у відповідності з формулами (3.3; 3.4; 3.5; 3.6) можна отримати оцінку градієнта інтенсивності. Для виконання формул (3.3) та (3.4) між маскою M та околом N виконують операцію скалярного добутку. Застосування масок Собеля та Робертса аналогічно масці Прюітт. Маски Собеля при оцінці контрасту використовують подвоєне значення оцінок по різні сторони від центральної точки. Маски Робертса мають розміри два на два, використовуються в меншому околі і операції з ними виконуються швидше, часто їх називають хрестоподібним оператором Робертса.

Функція Гауса що залежить від двох змінних має вигляд :

$$g(x,y) = ce^{-\frac{(x^2+y^2)}{2\sigma^2}} \tag{3.7}$$

де σ – середньоквадратичне відхилення;

c – деякий масштабний множник.

Ця функція аналогічна функцій нормального розподілення. При формуванні маски для фільтрації зазвичай значення всіх елементів маски обираються цілочисельними.

$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \tag{3.8}$$

Константа *c* у функції нормального розподілення була вибрана таким чином, щоб площа під кривою розподілення дуло рівною одиниці. Саме те, що інтеграл від функції *g(x)* описаної в формулі (3.8), дорівнює 1 дозволяє використовувати її в якості згладжувального фільтру, який не змінює вмісту областей постійних значень.

$$g''(x) \approx c_1 e^{-\frac{x^2}{2\sigma_1^2}} - c_2 e^{-\frac{x^2}{2\sigma_2^2}} \tag{3.9}$$

Диференціальним фільтром другого порядку, заснованим на застосуванні оператора Лапласа до функції Гауса називають ЛОГ-фільтр. Його представляють як різницю двох гаусіанів, що описані в формулі (3.9).

$\mathbf{G_{3\times3}} =$

1	2	1
2	4	2
1	2	1

;

$\mathbf{G_{7\times7}} =$

1	3	7	9	7	3	1
3	12	26	33	26	12	3
7	26	55	70	55	26	7
9	33	70	90	70	33	9
7	26	55	70	55	26	7
3	12	26	33	26	12	3
1	3	7	9	7	3	1

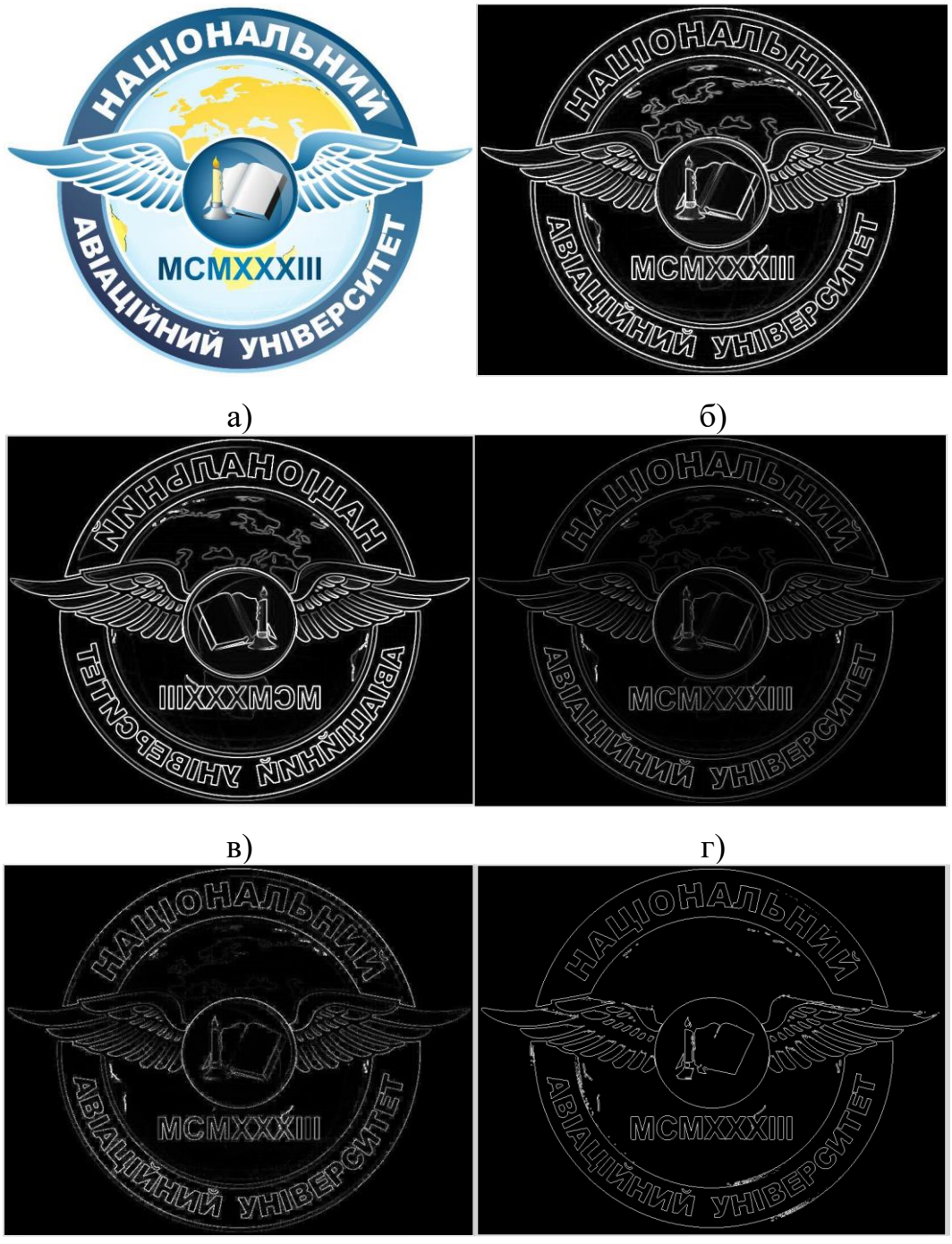
Рис. 3.4. Маски ЛОГ-фільтру розміром 3×3 та 7×7

Маски наведені на рис. 3.4 сформовані за допомогою застосування виразу описаного формулою (3.7). Спочатку обраховано значення функції при цілочисельних *x* та *y*, а потім підібране значення константи *c = 90* так, щоб мінімальний елемент маски дорівнював 1. Розмір маски 3×3 є мінімально можливим і дозволяє визначати дрібні деталі на зображенні. Маска розміром 7×7 дозволяє ігнорувати деякі дрібні деталі та визначати лише контури більшого розміру.

Детектор країв Кенні досить популярний та ефективний оператор, саме він був обраний в якості основного засобу для знаходження контурів зображення. Цей

детектор спочатку виконує згладжування півтонового зображення, після чого формує протяжні контурні сегменти. Цей алгоритм характеризується параметром згладжування σ та двома пороговими значеннями інтенсивності. Спочатку зображення згладжується Гаусовим фільтром з розмахом σ. Після чого в кожному пікселі згладженого зображення обчислюється величина і направлення градієнту. Направлення градієнту використовується для уточнення контурів, за рахунок видалення пікселів, величина градієнта яких не перевищує величину градієнта двох сусідніх пікселі в будь яку сторону від даного пікселя по осі направлення градієнта, цей метод називають немаксимальним придушенням. Він добре показує свою ефективність у разі необхідності генерації витончених контурів. Після обробки пікселів за величиною градієнта простежуються контури, що складаються із пікселів з великими значеннями градієнта . На завершальному етапі відбувається послідовне простеження неперервних контурів. Після простежування контури можуть проходити через пікселі в яких величина градієнту менше верхнього, але більше нижнього порогового значення. Зазвичай нижнє порогове значення прийнято обирати наполовину менше верхнього.

Проаналізуємо результати роботи всіх розглянутих алгоритмів визначення границь. На рис. 3.5 можна бачити тестове зображення для аналізу роботи алгоритмів визначення контурів. Частина рис. 3.5 а) містить оригінал тестового зображення розміром 800 на 600 пікселів. Частина рис. 3.5 б) містить результат обробки тестового зображення фільтром зі згорткою оператором Собеля з розміром ядра 3×3. Частина рис. 3.5 в) містить результат обробки тестового зображення фільтром зі згорткою оператором Прюітт з розміром ядра 3×3. Частина рис. 3.5 г) містить результат обробки тестового зображення фільтром зі згорткою перехресним оператором Робертса з розміром ядра 2×2. Область рис. 3.5 д) містить результат обробки тестового зображення ЛОГ-фільтром з розміром ядра 3×3. Елемент рис. 3.5 е) містить результат обробки тестового зображення за алгоритмом Кенні, з параметрами нижньої границі 120, верхньої 255.



а) б)

в) г)

д) е)

Рис. 3.5. Результати роботи алгоритмів визначення контурів: а) оригінал; б) оператор Собеля; в) оператор Прюітт; г) оператор Робертс; д) ЛОГ-фільтр; е) алгоритм Кенні

Метою впровадження цілого набору цих алгоритмів є забезпечення універсальності та гнучкості в роботі підсистеми перетворення керуючих програм.

За рахунок послідовного використання деяких фільтрів редагування зображень та певних фільтрів знаходження контурів, можна нівелювати певні недоліки та підсумувати деякі плюси використання тих чи інших фільтрів для знаходження контурів зображення. Наглядний приклад цього можна бачити на рис. 3.6. Це зображення є результатом застосування спочатку фільтру за згорткою оператором Прюітт, потім корекцією яскравості та контрасту, із наступним використанням фільтру «негатив» та кінцевим формуванням контурів за допомогою алгоритму Кенні.

Рис. 3.6. Результат застосування набору фільтрів

В цілому підсистема перетворення керуючих програм використовує досить велику кількість алгоритмів, до них входять алгоритми описані раніше в цьому підрозділи та інші алгоритми, які є досить поширеними у використанні. Наприклад алгоритм кешування даних для реалізації елементів управління програмою у вигляді кнопок «назад» та «вперед». Для масштабування використано методи найближчого сусіда, білінійної та бікубічної інтерполяції. Для реалізації чорно-білого фільтру використано метод Оцу. На рис. 3.7 зображено загальну схему алгоритму роботи програми, що включає всі етапи обробки зображення для процесу відтворення креслень.

Рис. 3.7. Загальна схема алгоритму роботи підсистеми перетворення керуючих програм

Схема алгоритму зображає послідовне виконання кроків обробки зображення для відтворення його на пристрої з ЧПК.

Рис. 3.8 демонструє зовнішній вигляд діаграми класів розроблюваної підсистеми, що включає в себе всі необхідні елементи визначені для реалізації в розділі 2.

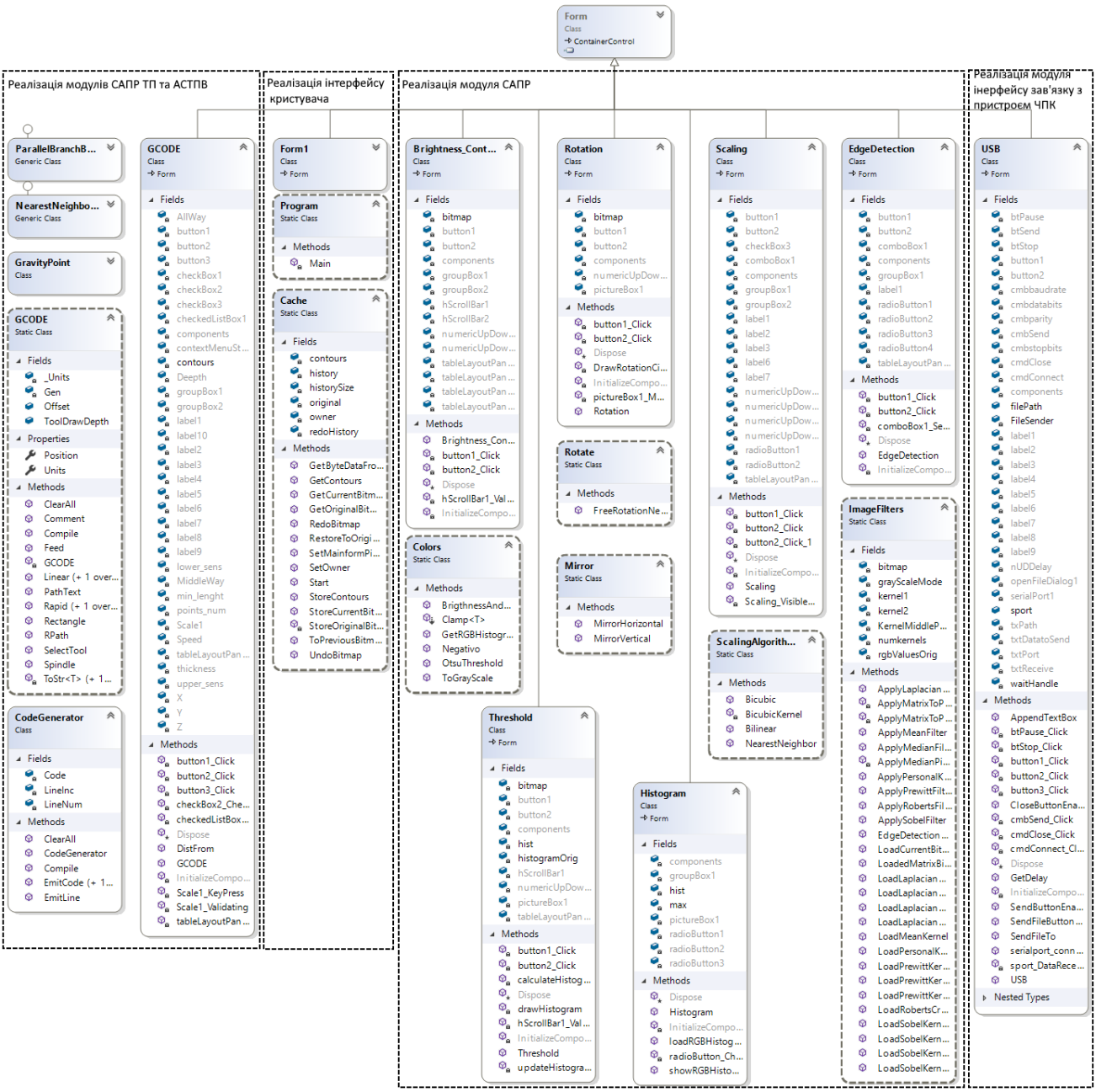


Рис. 3.8. Діаграма класів підсистеми перетворення керуючих програм

3.3. Етапи створення програми

Відповідно до розділу з вибором інструментальних засобів, підсистему розроблено засобами інтегрованого середовища розробки *Microsoft Visual Studio 2017 Community*. Після створення додатку *Windows Forms* необхідно заповнити проєкт всіма файлами описаними раніше в (пп.2.3). Наступним кроком розробки є заповнення екранних форм елементами користувацького інтерфейсу. Елементи

інтерфейсу користувача розміщено згідно загальноприйнятих правил, для того щоб уникнути непорозумінь під час роботи користувача з програмою.

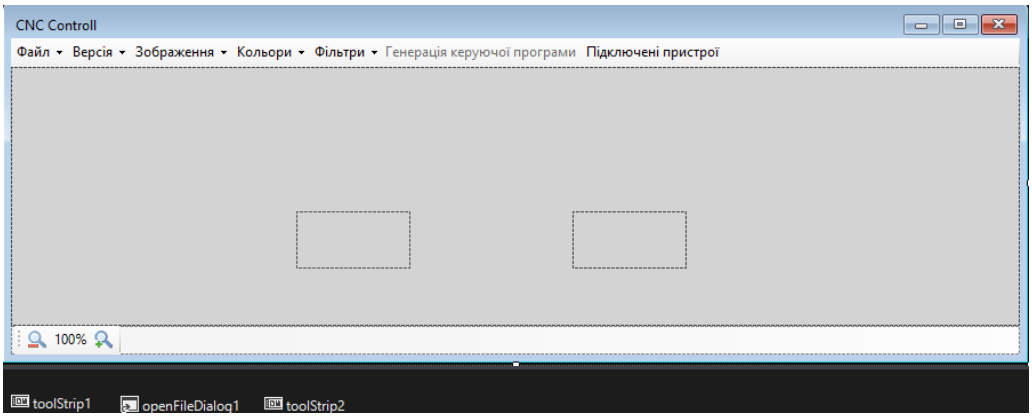


Рис. 3.9. Екранна форма головного вікна програми

Зовнішній вигляд екранної форми головного вікна програми зображено на рис. 3.9. Варто зазначити, що окрім візуальних елементів, вона також включає в себе і не візуальні компоненти. Іншими важливими вікнами можна назвати вікно «Генерація керуючих програм» та «Підключені пристрої», зображені на рис. 3.10 та 3.11 відповідно.

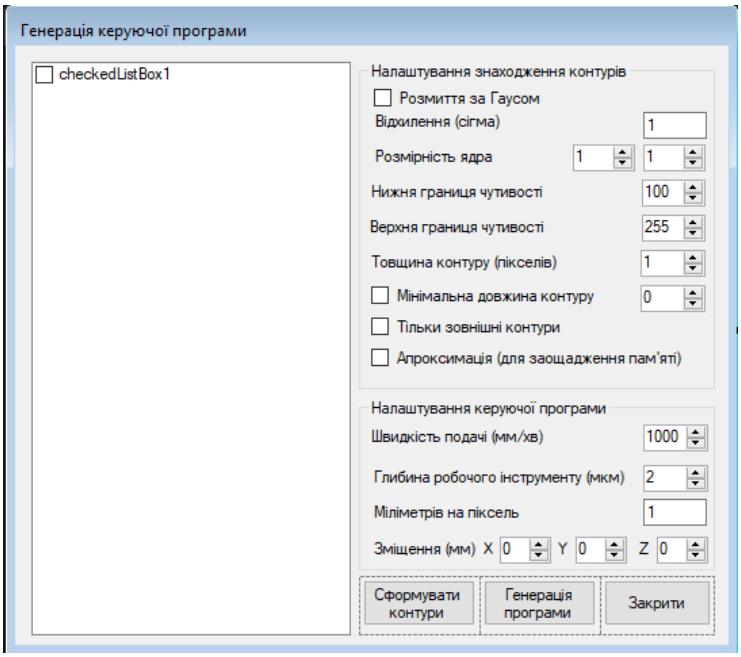


Рис. 3.10. Екранна форма «Генерація керуючої програми»

Вікно для генерації керуючої програми скомпоновано таким чином, щоб користувач міг встановити всі необхідні параметри для генерації керуючої програми без переходу до інших вікон програми.

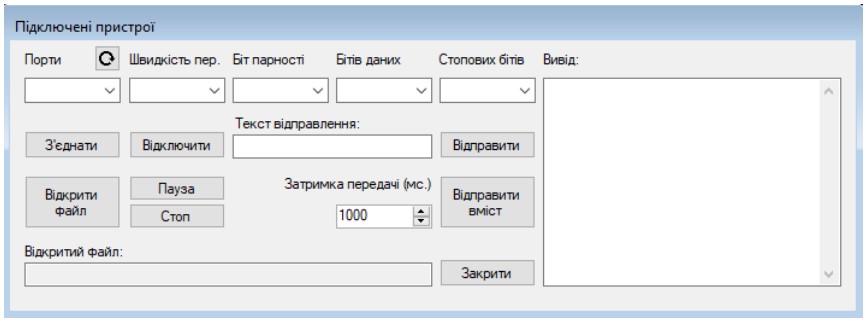


Рис. 3.11. Екранна форма «Підключені пристрої»

Після розміщення всіх необхідних компонентів на екранних формах, необхідно визначити та запрограмувати обробники подій для компонентів візуальних форм. Для прикладу розглянуто програмний код обробки події після натиснення кнопки «Генерації програми». Перша частина функції обробляє задані користувачем параметри технологічних режимів обробки, та зберігає їх в локальні змінні для подальшого використання, також обробляється список контурів на предмет їх включення в програму що генерується за допомогою прапорців в списку контурів.

```
private void button1_Click(object sender, EventArgs e)
{float FeedSpeed = Convert.ToSingle(Speed.Value);
float DrawDepth = Convert.ToSingle(Deepth.Value)/ -1000;
float Scale = Convert.ToSingle(Scale1.Text);
Feed(FeedSpeed);
Offset = new Vector3(Convert.ToSingle(X.Value), Convert.ToSingle(Y.Value),
Convert.ToSingle(Z.Value));
ToolDrawDepth = DrawDepth;
VectorOfVectorOfPoint code_contours = new VectorOfVectorOfPoint();
VectorOfVectorOfPoint contours_t = new VectorOfVectorOfPoint();
contours_t = Cache.GetContours();
```

for (int i = 0; i < checkedListBox1.Items.Count; i++){

if (checkedListBox1.GetItemCheckState(i) == CheckState.Checked){ code_contours.Push(contours[i]);}}

Наступна частина функції проводить пошук гравітаційних центрів фігур, які описують обрані контури за допомогою функцій програмної бібліотеки *Emgu.CV*.

MCvPoint2D64f center;

Moments moments;

IList<GravityPoint> PathPoints = new List<GravityPoint>();

for (int i = 0; i< code_contours.Size; i++){

moments = CvInvoke.Moments(code_contours[i]);

center = moments.GravityCenter;

PathPoints.Add(new GravityPoint(i, center.X, center.Y)); }

Пошук гравітаційних центрів необхідний для мінімізації довжини переміщень між контурами, за умови кількості контурів менше 12, відбувається мінімізація шляхом використання методу гілок та границь, у разі більшої кількості контурів, виконується мінімізація методом найближчого сусіда. Саме це виконує наступний фрагмент коду.

IProblem<GravityPoint> problem = new ProblemBuilder<GravityPoint> (typeof(GravityPoint))

Locations(PathPoints)

Distances((GravityPoint p1,GravityPoint p2)=>DistFrom(p1.X, p1.Y, p2.X, p2.Y))

FixedFirstLocation(0)BuildIntegerArray();ISolver<GravityPoint> solver;

if (code_contours.Size < 12){

solver = new ParallelBranchBoundSolver<GravityPoint>(problem);}

else{solver = new NearestNeighborSolver<GravityPoint>(problem);}

IPath<GravityPoint> path = solver.Solve();

int[] loc = path.GetLocations();

VectorOfVectorOfPoint optimized_contours = new VectorOfVectorOfPoint();

for (int i = 0; i < loc.Length; i++)

{ optimized_contours.Push(code_contours[loc[i]]);}

Після мінімізації шляху виконуючого пристрою функція починає створення керуючої програми за допомогою обходу масиву обраних контурів у визначеному попереднім етапом виконання функції порядку, забезпечуючи проходження по контурам точок із опущеним на задану глибину інструментом, перехід між контурами забезпечується у пришвидшеному режимі із піднятим інструментом.

Для забезпечення цілісності керуючої програми попередні значення кешу генерованих програм очищаються. Також важливо правильно обробляти кінці контурів, при замкненому контурі треба з'єднати першу і останню точки у разі незамкненого контуру цього робити не слід.

```
ClearAll();
for (int i = 0; i < code_contours.Size; i++){
Rapid(optimized_contours[i][0].X* Scale, optimized_contours[i][0].Y * Scale, 5);
for (int j = 0; j < optimized_contours[i].Size; j++){
Linear(optimized_contours[i][j].X * Scale, optimized_contours[i][j].Y * Scale, DrawDepth);}
if (CvInvoke.ContourArea(optimized_contours[i]) > CvInvoke.ArcLength (optimized_contours[i], true)){
Linear(optimized_contours[i][0].X * Scale, optimized_contours[i][0].Y * Scale, DrawDepth);
Rapid(optimized_contours[i][0].X * Scale, optimized_contours[i][0].Y Scale, 5);}
else{Rapid(optimized_contours[i][optimized_contours[i].Size-1].X * Scale, optimized_contours[i][optimized_contours[i].Size-1].Y * Scale, 5);}}
```

Завершальний етап функції призначений для збереження згенерованої програми, для цього з міркувань безпеки програма зберігається у резервний файл, що знаходиться у папці з програму і тільки тоді виводить на екран користувача діалог з файловою системою для збереження основної версії файлу. Також важливим є запис адреси збереженого файлу в кеш програми для подальшого використання на етапі передачі програми на пристрій ЧПК.

```
File.WriteAllText("test.gcode", Compile());
SaveFileDialog savedialog = new SaveFileDialog();
```

savedialog.Title = "Зберегти програму як...";

savedialog.OverwritePrompt = true;

savedialog.CheckPathExists = true;

savedialog.Filter = "Текстові файли(.txt)|*.txt|Всі файли(*.*)|*.*";*

if (savedialog.ShowDialog() == DialogResult.OK) {

try{File.WriteAllText(savedialog.FileName, Compile());}

catch{MessageBox.Show("Неможливо зберегти програму", "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);}}

Cache.StoreFileName(savedialog.FileName);}

За аналогією, згідно необхідних функціональних вимог, описуються всі інші обробники подій компонентів форм програми. Детальний розгляд кожного із них передбачає використання великого об'єму тексту, що виходить за рамки пояснювальної записки. Після опису всіх функцій, проведено первинне налагодження програми, виправлено помилки в вихідному коді, протестовано роботу основних функцій на тестових зразках.

3.4. Тестування

Останнім етапом розробки програми є її тестування. Тестування є важливим та невиключним етапом розробки програми, що дозволяє уникнути нераціонального розподілення ресурсів на всіх етапах розробки та забезпечити належний рівень якості програмного продукту.

Ручне тестування. Процес та результати ручного тестування будуть представлені за допомогою тест-кейсів. *Test Case* – це тестовий артефакт, суть якого полягає у виконанні певної кількості дій і або умов, необхідних для перевірки певної функціональності програмної системи, що розробляється.

Набір тест-кейсів є досить поширеною та ефективною методикою тестування. Якщо фактичний результат відповідає очікуваному, то тест пройдено і в третю колонку записується «пройдено», а в іншому випадку – записується «не пройдено». Сам тестовий випадок складається з трьох частин:

– передумова – список кроків, які приводять систему до стану, придатного для тестування, або список перевірок умов, що система вже знаходиться в необхідному стані;

– опис тестового випадку – список дій, за допомогою яких здійснюється основна перевірка функціоналу;

– постумова – список дій, які повертають систему в початковий стан.

Тестування проведено за допомогою набору тест кейсів. Знайдено і виправлено певну кількість помилок та дефектів. Для прикладу в пояснювальній записці наведено декілька тест кейсів.

В таблиці 3.1 розглянуто тестовий випадок відкриття та збереження зображення.

Таблиця 3.1

Тестовий випадок відкриття та збереження зображення

Передумова		
Відкрите головне вікно програми.		
Подія	Очікуваний результат	Фактичний результат
Тестовий випадок		
Обрати пункт «Відкрити» в розділі «Файл» головного меню програми.	Відкривається діалогове вікно файлової системи. Тестовий рисунок відображається у списку об’єктів файлової системи.	Пройдено.
Вибрати тестовий рисунок формату *.BMP;.GIF; .JPG; .PNG;* або *.TIF*.	Тестовий рисунок виділяється рамкою обраного об’єкту, ім’я рисунку заноситься у поле «Ім’я файлу».	Пройдено.

Закінчення таблиці 3.1

Обрати пункт «Зберегти як» в розділі «Файл» головного меню програми.	Відкривається діалогове вікно файлової системи комп’ютера.	Пройдено.
Вказати назву та в списку обрати один із форматів зображення для збереження. Натиснути «Зберегти».	Діалогове вікно збереження зникає, а у файловій системі з'являється зображення із вказаною назвою та форматом.	Пройдено.
Постумова		
–	На екран виводиться головне вікно програми, що включає в себе обране раніше тестове зображення.	Пройдено.

Це один із найпростіших тестових випадків, для його виконання застосовуються елементи керування програмою, такі як «Відкрити» та «Зберегти як», після процесу збереження зображення програма сама повертається в початковий стан. Процес який в ньому розглядається виконується майже кожного разу після запуску програми. Тест-кейс пройдено успішно.

Наступним розглянуто складніший тестовий випадок для знаходження контурів відкритого зображення. Оскільки знаходження контурів одна із основних функцій, її тестуванню приділено багато уваги. Цей тестовий випадок складається із багатьох кроків, що дозволяє протестувати правильність взаємодії різних модулів підсистеми перетворення керуючих програм. В результаті проходження цього тест-кейсу підсистема перетворення керуючих програм успішно завершила всі кроки та показала високий рівень якості функціональної реалізації методів обробників подій. Таблиця, що докладно показує кроки виконання тест-кейсу для знаходження контурів зображення наведена далі (таблиця 3.2).

Таблиця 3.2

Тестовий випадок знаходження контурів відкритого зображення

Передумова		
Відкрити головне вікно програми. За допомогою пункту «Відкрити »в розділі «Файл» головного меню програми відкрите тестове зображення формату *.BMP;.GIF; .JPG; .PNG*; або *.TIF*.		
Подія	Очікуваний результат	Фактичний результат
Тестовий випадок		
Обрати «Визначити контури» в розділі «Фільтри» головного меню програми.	Відкривається вікно «Знаходження контурів».	Пройдено.
Активувати перемикач з розміром ядра 3×3.	Перемикач активується.	Пройдено.
Натиснути кнопку «Прийняти»	Вікно «Знаходження контурів» закривається. Зображення на головному вікні стає чорно білим, залишаються тільки контури тестового зображення.	Пройдено.
Постумова		
Обрати пункт «Назад» з розділу «Версія» головного меню.	В робочому полі головного вікна програми замість чорно-білого з’являється оригінальне тестове зображення.	Пройдено.

Також до ручного тестування можна віднести процес перевірки коректності генерації керуючої програми за допомогою емулятора пристрою з ЧПК веб-сервісу *NC Viewer*. За результатами роботи емулятора (рис. 3.12) можна бачити, керуюча програма сформульовано коректно, також прослідкувавши за переходами між контурами бачити що мінімізація пройденого шляху теж працює непогано.

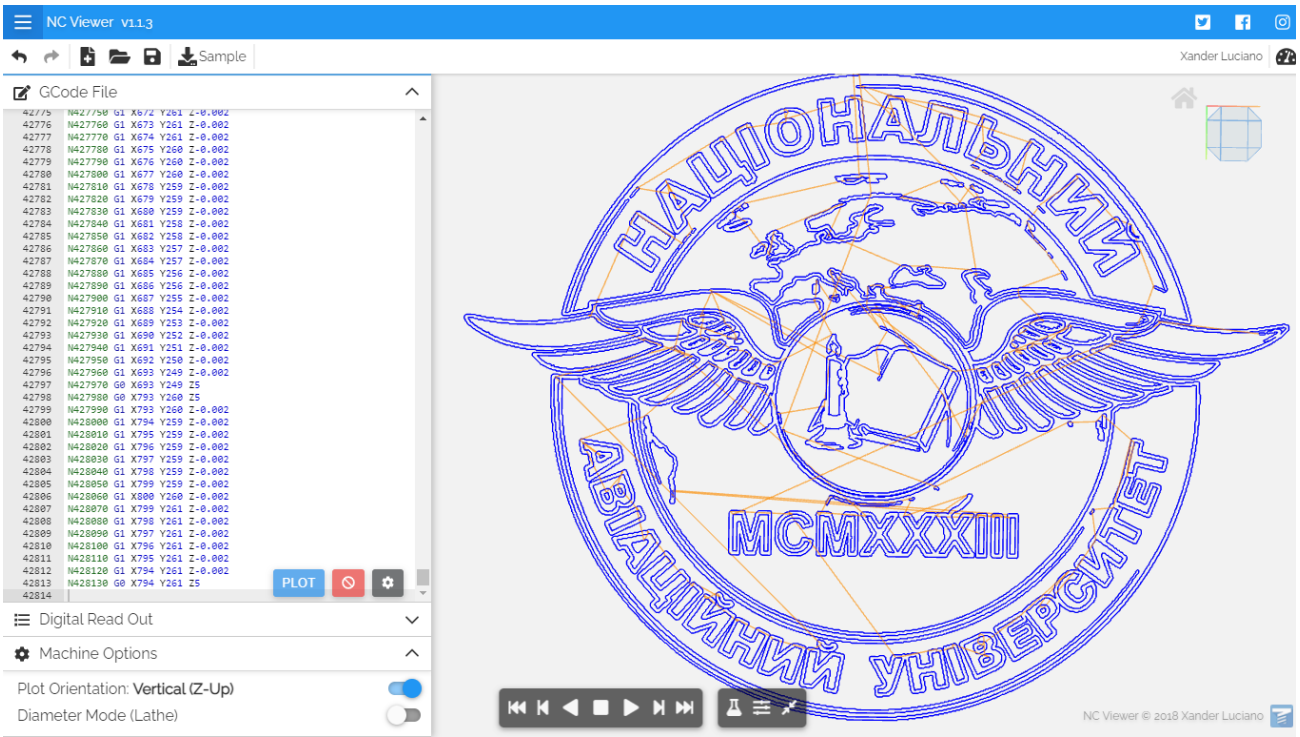


Рис. 3.12. Результат роботи емулятора пристрою з ЧПК

Іншим потужним інструментом для тестування програм вважається написання автоматизованих тестів. Функція передачі даних на пристрій з ЧПК добре підходить для тестування автоматизованими засобами. Для тестування функції передачі даних створено програмний модуль, що частково копіює зовнішній вигляд та функціональну наповненість візуальної форми «Підключені пристрої». Зовнішній вигляд модуля для автоматизованого тестування можна бачити на рис. 3.13.

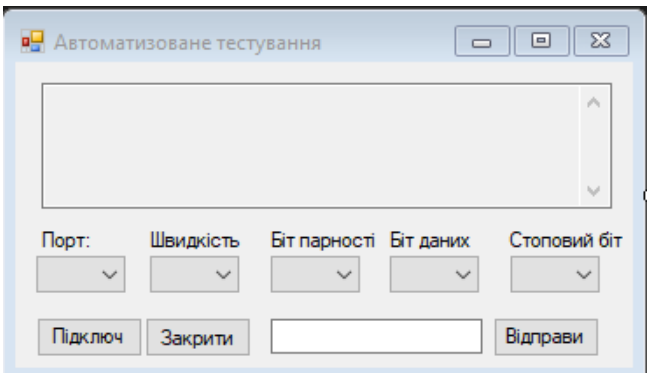


Рис. 3.13. Екранна форма модуля автоматизованого тестування

Після чого на комп'ютері створено пару портів для підключення за допомогою емулятора віртуальних портів «*Virtual Serial Port Driver Pro*». Результатом роботи автоматизованого модуля є записи про успішну передачу і отримання даних. Цілісність передачі даних підтверджується фактом співпадіння розмірів переданих та отриманих даних. Загальний вигляд вікон підсистеми перетворення керуючих програм, автоматизованого модуля для тестування емулятора віртуальних портів можна бачити на рис. 3.14.

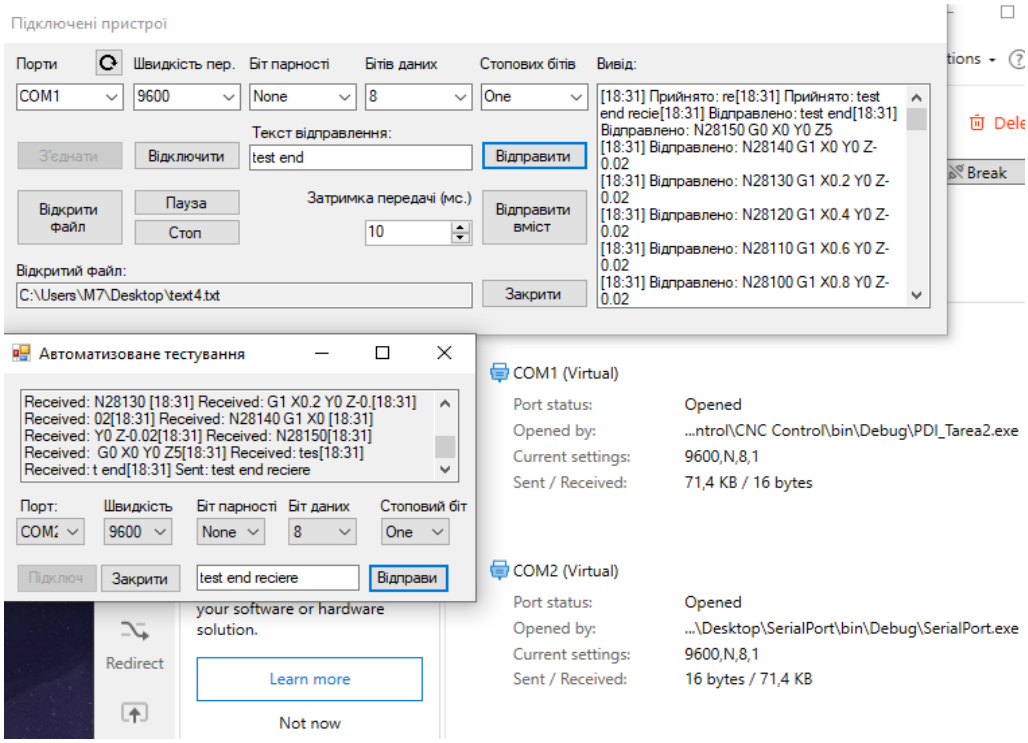


Рис. 3.14. Результати роботи модуля автоматизованого тестування

Отже, на етапі тестування було виконане ручне і автоматизоване тестування програмної реалізації підсистеми перетворення керуючих програм, під час тестування виправлено виявлені помилки в роботі програми, після чого підтверджено правильність роботи всіх функцій програми.

3.5. Інструкція користувача

Елементи користувацького інтерфейсу програми можна розділити на декілька груп:

– засоби управління програмою під час роботи;
– засоби перегляду та редагування зображень;
– засоби для знаходження контурів та генерації керуючих програм;
– засоби для зв’язку з пристроєм ЧПК та передачі керуючих програм.

Група засобів для управління програмою під час роботи відповідає класичному стилю розташування для віконних додатків *Windows*. Головне вікно програми у верхньому правому куті містить набір кнопок для згортання, масштабування та закриття головного вікна. Також кнопка «Закрити» існує в розділі «Файл» головного меню програми. Всі інші вікна в програмі включають в себе кнопку закриття в їх нижній правій області.

Також для управління програмою відведено розділ головного меню «Версія», що включає в себе три пункти: «Назад», «Вперед» та «Завантажити оригінал». Кнопка «Назад» дозволяє повернути програму в стан, що передує останній виконаній користувачем дії. Кнопка «Вперед» виконує протилежну кнопці «Назад» функцію і дозволяє повернутись до останнього стану програми після застосування кнопки «Назад». Пункт «Завантажити оригінал» відновлює початковий стан завантаженого в програму зображення.

Група засобів для перегляду, збереження та редагування зображень також розташована в головному меню програми. В розділах «Файл», «Зображення» «Кольори» та «Фільтри».

Розділ «Файл» включає в себе стандартні функції для виклику діалогів відкриття та збереження зображень та швидкого збереження зображення без виклику діалогу з файловою системою. Розділ «Зображення» дозволяє проводити редагування зображення шляхом зміни його масштабу за допомогою виклику вікна «Масштабування. Вікно «Масштабування» можна бачити на рис. 3.15 зліва.

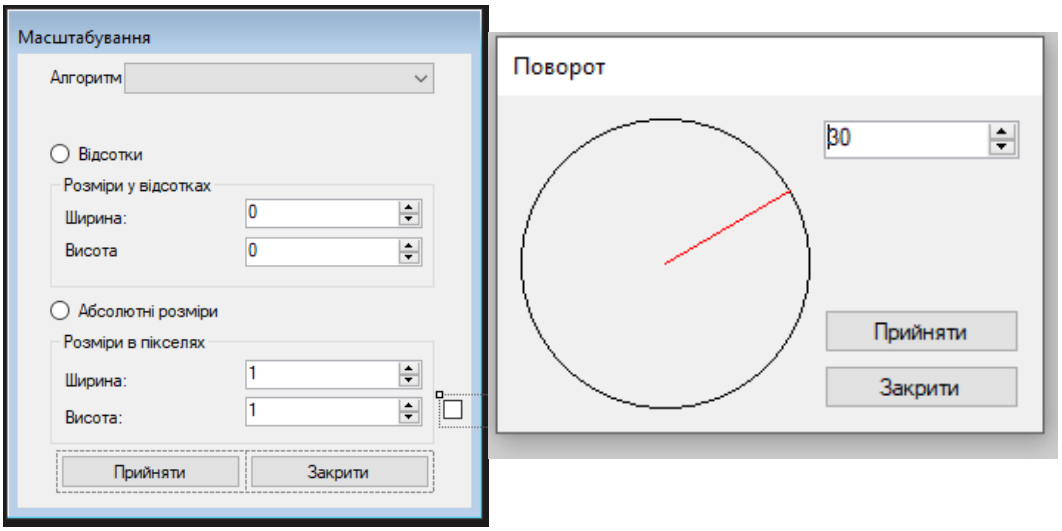


Рис. 3.15. Вікна «Масштабування» та «Поворот»

Поворот зображення за допомогою вікна «Поворот» дозволяє повернути відкрите зображення на довільну кількість градусів в довільному напрямку. Вікно «Поворот» можна бачити на рис. 3.15 справа.

Пункт меню «Інформація» (рис. 3.16) дозволяє переглянути інформацію про відкрите зображення для інформування користувача.

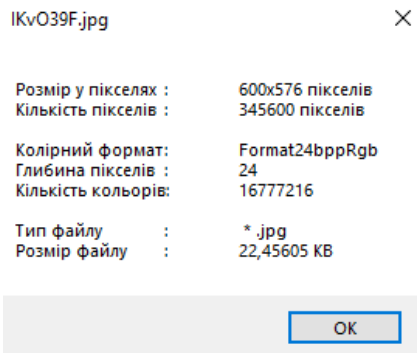


Рис. 3.16. Вікно з інформацією про зображення

Пункти «Віддзеркалити по горизонталі» та «Віддзеркалити по вертикалі» повністю відповідають своїм назвам та відображають зображення відповідно по горизонталі та по вертикалі.

Пункт «Гістограма *RGB*» дозволяє переглядати графік інтенсивності кожного із каналів *RGB* за допомогою перемикачів «Червоний», «Зелений» і «Синій» назви перемикачів відповідають своєму каналу кольору.

Вікно «Яскравість і контраст» відповідно до назви дозволяє змінювати значення яскравості (у верхній області форми) та контрасту (у нижній області форми) за допомогою полів для вводу або ж повзунків.

Вікно «Чорно-білий фільтр» призначене для редагування рівню тіней зображення, дозволяє вказувати потрібне значення за допомогою повзунка або поля для вводу.

Розділ «Фільтри» включає в себе пункти «Знаходження контурів» (рис. 3.17), «Розмиття» та «Негатив». В свою чергу пункт розмиття включає в себе два види розмиття зображення «Середнє» та «Медіана».

Пункт «Визначити контури» дозволяє виконати один із операторів визначення контурів для подальшого формування керуючої програми.

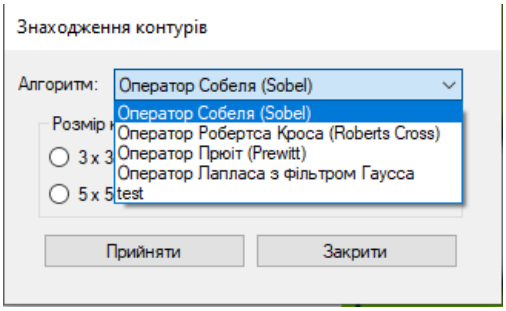


Рис. 3.17. Вікно «Знаходження контурів»

На рис. 3.18 зображено зовнішній вигляд вікна «Генерація керуючих програм». Налаштування генератора контурів дозволяють обрати нижню та верхню межі чутливості генератора, ці параметри впливають на чутливість генератора до пікселів з різною інтенсивністю кольору. Товщина контуру встановлює товщину контурів, які будуть згенеровані в пікселях. Параметр мінімальної довжини

дозволяє виключити зі списку контури, що мають довжину менше заданого значення. Апроксимація – параметр що дозволяє оптимізувати кількість точок в контурах, що описують правильні фігури.

Панель «Налаштування керуючої програми» дозволяє налаштовувати режими обробки деталі, такі як швидкість руху в міліметрах на хвилину, глибина проникнення інструменту в мікрометрах, налаштування відношення пікселів до міліметрів для регулювання масштабу та встановлення зміщення на певну кількість міліметрів. Панель зліва включає в себе список контурів, згенерованим генератором контурів та дозволяє включати або ж виключати їх із маршрутної карти програми, для візуального контролю та розуміння включених або виключених контурів зміни відображаються в робочій області головного вікна програми.

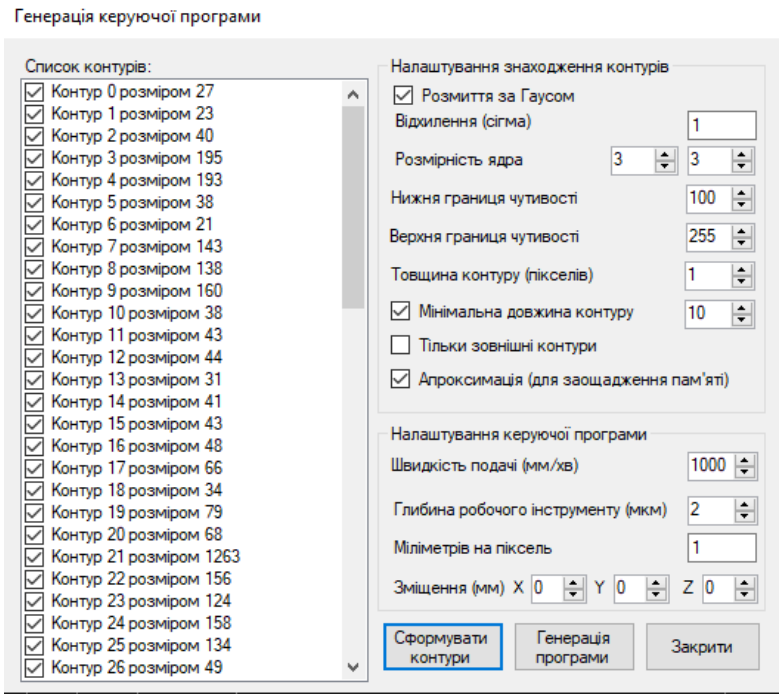


Рис. 3.18. Вікно «Генерація керуючої програми»

Екранна форма «Підключені пристрої» забезпечує користувача інтерфейсом для зв’язку з пристроєм ЧПК. Містить в собі групу елементів для вибору і налаштування порту з’єднання (номер порту, швидкість передачі, біт парності, біти даних, стопові біти). Кнопки для підключення та відключення від пристрою, поле

для введення команд і кнопка для ручної передачі їх на пристрій, для вибору файлу та передачі його на пристрій з ЧПК. Поле «Затримка передачі» дозволяє до і під час передачі регулювати швидкість передачі команд. Частина вікна справа виводить на екран результати всіх дій програми. Зовнішній вигляд вікна «Підключені пристрої зображено на рис. 3.19.

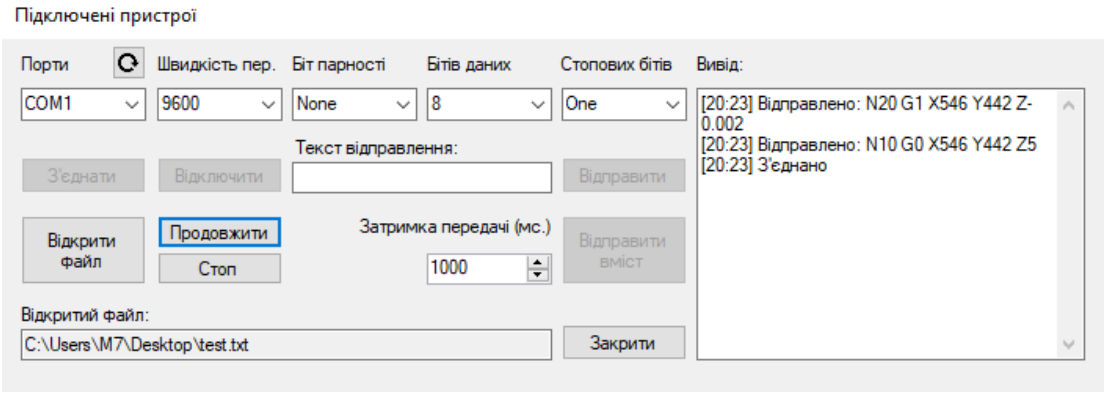


Рис. 3.19. Вікно «Підключені пристрої»

3.6. Висновки до розділу

1) Розглянуто основні алгоритми для знаходження контурів на зображенні, визначено набір алгоритмів до реалізації та розглянуто принципи їх роботи. Відповідно до обраного набору алгоритмів створено загальну схему алгоритму роботи підсистеми перетворення керуючих програм.

2) Складено діаграму класів для програмної реалізації підсистеми перетворення керуючих програм.

3) Створено набір візуальних форм та розміщено на них компоненти користувацького інтерфейсу. Створено та функціонально забезпечено обробники подій для створених компонентів користувацького інтерфейсу.

4) Протестовано програмний комплекс за допомогою методів ручного та автоматизованого тестування, що показало рівень якості програмного продукту.

5) Описано інструкцію користувача для роботи з підсистемою перетворення керуючих програм.

ВИСНОВКИ

1) Розглянуто основні поняття та терміни що використовуються в системах з числовим програмним керуванням, підвищено рівень обізнаності в галузі.

2) Проаналізовано дані статистки використання верстатів з ЧПК, програмного забезпечення для роботи з пристроями ЧПК. Як результат визначено проблему дефіциту комплексів для генерації керуючих програм, що розповсюджуються на безоплатній основі.

3) Сформульоване завдання проектування.

4) Розглянуто результати аналізу статистики та сучасних засобі генерації керуючих програм, що дало змогу сформулювати функціональні вимоги до кожного з модулів підсистеми перетворення керуючих програм.

5) Проаналізовано сучасні засоби для розробки програм та обрано ті, які задовольняють потребам розробки підсистеми перетворення керуючих програм.

6) Розроблено і описано структурну схему підсистеми перетворення керуючих програм у складі системи відтворення креслень.

7) За допомогою діаграми послідовності, описано зв’язки модулів підсистеми перетворення програм.

8) Розглянуто основні алгоритми для знаходження контурів на зображенні, визначено набір алгоритмів до реалізації та створено загальну схему алгоритму роботи підсистеми перетворення керуючих програм.

9) Складено діаграму класів для програмної реалізації підсистеми перетворення керуючих програм.

10) Створено програмну реалізацію підсистеми перетворення керуючих програм.

11) Протестовано модулі підсистеми за допомогою методів ручного та автоматизованого тестування.

12) Описано інструкцію користувача.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Ковальов В.А. Конструктивні особливості та основи програмування верстатів з числовим програмним керуванням. / В.А. Ковальов, А.Ю. Гаврушкевич, Н.В. Гаврушкевич; – КПІ ім. Ігоря Сікорського. – Київ, 2020. 158с.

2) Лозінський Д.О. Системи автоматизованого проектування верстатів з ЧПК. / Д.О. Лозінський; ВНТУ. – Вінниця, 2018. 84 с.

3) Джемелінський В.В. Методичні вказівки з лабораторних і практичних робіт до вивчення дисципліни «Основи програмування на верстатах з числовим програмним керуванням» / В.В. Джемелінський, О.Д. Кагляк, Д.А. Лесик. ; НТУУ «КПІ». – Київ, 2011, 56 с.

4) Обзор популярных систем автоматизированного проектирования (CAD) [Електронний ресурс]. – 2017. *www.pointcad.ru* – Режим доступу: *https://www.pointcad.ru/novosti/obzor-sistem-avtomatizirovannogo-proektirovaniya* (дата звернення 18.05.2021) – Назва з екрана.

5) *CNC Machine Tools Market Size; COVID-19 Impact Analysis, By Type, By Application and Regional Forecast.* [Електронний ресурс]. – 2019. *www.fortunebusinessinsights.com* – Режим доступу: *https://www.fortunebusinessinsights.com/industry-reports/computer-numerical-controls-cnc-machine -tools-market-101707* (дата звернення 18.05.2021) – Назва з екрана.

6) *CNCCookbook 2016 CAD Survey Results, Part 1: Market Share* / [Електронний ресурс]. – 2016. *www.cnccookbook.com* – Режим доступу: *https://www.cnccookbook.com/cnccookbook-2016-cad-survey-results-part-1-market-share*/ (дата звернення 19.05.2021) – Назва з екрана.

7) Шаповалова С.І. Вдосконалення *CAM*-систем для невеликих виробництв. / С.І. Шаповалова, О.М. Бараніченко; – Київ, 2017. 10 с.

8) Шапиро Л. Компьютерное зрение. / Л. Шапиро, Дж. Стокман; – Москва: Бином. Лаборатория знаний, 2015. 752 с.

9) Шилдт Г. C# 4.0: полное руководство. / Г. Шилдт; – Санкт-Петербург: И. Д. Вильямс, 2011. 1056 с.

10) Алгоритмы: построение и анализ. / Кормен Т. [та ін.]; – Санкт-Петербург: И. Д. Вильямс, 2013. 1324 с.

11) Аносов М.С. Основы разработки управляющих программ для станков с ЧПУ в системе *SIEMENS NX*. / М.С. Аносов Г.Н. Каневский, Р.Ш. Мансуров, С.Б. Сорокин; – Нижегородский государственный технический университет им. Р.Е Алексеева. – Нижний Новгород, 2019, 111 с.

12) *ISO 6983-1:2009. Automation systems and integration – Numerical control of machines – Program format and definitions of address words – Part 1:Data format for positioning, line motion and contouring control systems.* – Введ. 1983, Ред. 2009, 26 с.

13) ДСТУ 3008-15 Документація. Звіти у сфері науки і техніки. Структура і правила оформлення.

14) Бойченко С.В., Іванченко О.В. Положення про дипломні роботи (проекти) випускників Національного авіаційного університету. – К.: НАУ, 2017. – 63 с.

ДОДАТОК А

Лістинг коду програмних модулів

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Numerics;
using System.IO;
using TheTravelingSalesman;
using TheTravelingSalesman.Solvers;
using Emgu;
using Emgu.CV;
using Emgu.CV.Structure;
using Emgu.CV.Util;
using static GCODE.GCODE;
namespace PDI_Tarea2{
   public partial class GCODE : Form{
      private static VectorOfVectorOfPoint contours = new
VectorOfVectorOfPoint();
      public GCODE(){
         InitializeComponent();}
      private void button1_Click(object sender, EventArgs e) {
```

```
float FeedSpeed = Convert.ToSingle(Speed.Value);
float DrawDepth = Convert.ToSingle(Deepth.Value)/ -1000;
float Scale = Convert.ToSingle(Scale1.Text);
Feed(FeedSpeed);
// Rapid(0, 0, 1);          // холостий хід
// SelectTool(Tool);          //Вибір інструменту
Offset = new Vector3(Convert.ToSingle(X.Value), Convert.ToSingle(Y.Value), Convert.ToSingle(Z.Value));  //зміщення
ToolDrawDepth = DrawDepth;
VectorOfVectorOfPoint code_contours = new VectorOfVectorOfPoint();
VectorOfVectorOfPoint contours_t = new VectorOfVectorOfPoint();
contours_t = Cache.GetContours();
for (int i = 0; i < checkedListBox1.Items.Count; i++){
   if (checkedListBox1.GetItemCheckState(i) == CheckState.Checked){
      code_contours.Push(contours[i]);}}
//GravityCenterFind
MCvPoint2D64f center;
Moments moments;
IList<GravityPoint> PathPoints = new List<GravityPoint>();
for (int i = 0; i< code_contours.Size; i++){
   moments = CvInvoke.Moments(code_contours[i]);
   center = moments.GravityCenter;
   PathPoints.Add(new GravityPoint(i, center.X, center.Y));      }
IProblem<GravityPoint> problem = new ProblemBuilder<GravityPoint>(typeof(GravityPoint))
   .Locations(PathPoints)
   .Distances((GravityPoint p1, GravityPoint p2) => DistFrom(p1.X, p1.Y, p2.X, p2.Y))
   .FixedFirstLocation(0)
```

```
        .BuildIntegerArray();
    ISolver<GravityPoint> solver;
    if (code_contours.Size < 12){
        solver = new ParallelBranchBoundSolver<GravityPoint>(problem);}
    else{
        solver = new NearestNeighborSolver<GravityPoint>(problem);}
    IPath<GravityPoint> path = solver.Solve();
    int[] loc = path.GetLocations();
    VectorOfVectorOfPoint optimized_contours = new
VectorOfVectorOfPoint();
    for (int i = 0; i < loc.Length; i++){
        optimized_contours.Push(code_contours[loc[i]]);}
    ClearAll();
    for (int i = 0; i < code_contours.Size; i++){
        Rapid(optimized_contours[i][0].X* Scale, optimized_contours[i][0].Y *
Scale, 5);          for (int j = 0; j < optimized_contours[i].Size; j++){
            Linear(optimized_contours[i][j].X * Scale,
optimized_contours[i][j].Y * Scale, DrawDepth);}
    if (CvInvoke.ContourArea(optimized_contours[i]) >
CvInvoke.ArcLength(optimized_contours[i], true)){
            Linear(optimized_contours[i][0].X * Scale,
optimized_contours[i][0].Y * Scale, DrawDepth);
            Rapid(optimized_contours[i][0].X * Scale,
optimized_contours[i][0].Y * Scale, 5);  }
        else{
            Rapid(optimized_contours[i][optimized_contours[i].Size-1].X *
Scale, optimized_contours[i][optimized_contours[i].Size-1].Y * Scale, 5); }}
    File.WriteAllText("test.gcode", Compile());
    SaveFileDialog savedialog = new SaveFileDialog();
    savedialog.Title = "Зберегти програму як...";
```

```
            savedialog.OverwritePrompt = true;
            savedialog.CheckPathExists = true;
            savedialog.Filter = "Текстові файли(*.txt)|*.txt|Всі файли(*.*)|*.*";
            if (savedialog.ShowDialog() == DialogResult.OK) //если в диалоговом окне нажата кнопка "ОК"{
                try{
                    File.WriteAllText(savedialog.FileName, Compile());}
                catch{
                    MessageBox.Show("Неможливо зберегти програму", "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);}}
            Cache.StoreFileName(savedialog.FileName); }
        public static double DistFrom(double X1, double Y1, double X2, double Y2){
            double dist = Math.Sqrt(Math.Pow(X2 - X1, 2) + Math.Pow(Y2 - Y1, 2));
            return dist;}

        private void button3_Click(object sender, EventArgs e){
            int Contour_size = Convert.ToInt32(thickness.Value);
            double lower_sensitivity_limit = Decimal.ToDouble(lower_sens.Value);
            double upper_sensitivity_limit = Decimal.ToDouble(upper_sens.Value);
            Bitmap bitmap = Cache.GetCurrentBitmap();
            Mat hierarchy = new Mat();
            Image<Bgr, byte> input_image = new Image<Bgr, byte>(bitmap);
            Image<Gray, byte> output_image = input_image.Convert<Gray, byte>().ThresholdBinary(new Gray(lower_sensitivity_limit), new Gray(upper_sensitivity_limit));
            if (Gsmoth.Checked == true){
                CvInvoke.GaussianBlur(output_image, output_image, new Size(Decimal.ToInt32(MatrixX.Value), Decimal.ToInt32(MatrixY.Value)), Convert.ToDouble(Sigma.Text));}
```

```
if ((this.checkBox1.Checked == false) && (this.checkBox3.Checked == false)){
CvInvoke.FindContours(output_image, contours, hierarchy,
Emgu.CV.CvEnum.RetrType.Tree,
Emgu.CV.CvEnum.ChainApproxMethod.ChainApproxNone);}
        else if ((this.checkBox1.Checked == false) && (this.checkBox3.Checked
== true)){
            CvInvoke.FindContours(output_image, contours, hierarchy,
Emgu.CV.CvEnum.RetrType.Tree,
Emgu.CV.CvEnum.ChainApproxMethod.ChainApproxSimple);}
        else if ((this.checkBox1.Checked == true) && (this.checkBox3.Checked
== false)){
            CvInvoke.FindContours(output_image, contours, hierarchy,
Emgu.CV.CvEnum.RetrType.External,
Emgu.CV.CvEnum.ChainApproxMethod.ChainApproxNone);}
        else if ((this.checkBox1.Checked == true) && (this.checkBox3.Checked
== true)){
            CvInvoke.FindContours(output_image, contours, hierarchy,
Emgu.CV.CvEnum.RetrType.External,
Emgu.CV.CvEnum.ChainApproxMethod.ChainApproxSimple);}
    if (checkBox2.Checked == true){
            int min_lenght = Convert.ToInt32(this.min_lenght.Value);
            VectorOfVectorOfPoint temp_contours = new
VectorOfVectorOfPoint();
            for (int i = 0; i < contours.Size; i++){
              if (contours[i].Size >= min_lenght){
                  temp_contours.Push(contours[i]);}}
            contours = temp_contours;}
          checkedListBox1.Items.Clear();
```

```
            for (int i = 0; i < contours.Size; i++){
                checkedListBox1.Items.Add("Контур " + i + " розміром " +
contours[i].Size, true);}
            Image<Gray, byte> blackBackground = new Image<Gray,
byte>(bitmap.Width, bitmap.Height, new Gray(0));
            CvInvoke.DrawContours(blackBackground, contours, -1, new
MCvScalar(255, 0, 0), Contour_size);
            Cache.SetMainformPictureBox(blackBackground.Bitmap);
            Cache.StoreCurrentBitmapData();
            Cache.StoreContours(contours);}
        private void button2_Click(object sender, EventArgs e){
            this.Close();}

        private void checkBox2_CheckedChanged(object sender, EventArgs e){
            if (this.checkBox2.Checked == true){
                min_lenght.Enabled = true;}
            else{
                min_lenght.Enabled = false;};}
        private void checkedListBox1_SelectedIndexChanged(object sender,
EventArgs e){
            Bitmap bitmap = Cache.GetCurrentBitmap();
            Image<Gray, byte> checked_contours = new Image<Gray,
byte>(bitmap.Width, bitmap.Height, new Gray(0));
            int Contour_size = Convert.ToInt32(thickness.Value);
            VectorOfVectorOfPoint draw_contours = new VectorOfVectorOfPoint();
            VectorOfVectorOfPoint contours_t = new VectorOfVectorOfPoint();
            contours_t = Cache.GetContours();
            for (int i = 0; i < checkedListBox1.Items.Count; i++){
                if (checkedListBox1.GetItemCheckState(i) == CheckState.Checked){
                    draw_contours.Push(contours[i]);}}
```

```
            CvInvoke.DrawContours(checked_contours, draw_contours, -1, new
MCvScalar(255, 0, 0), Contour_size);
            Cache.SetMainformPictureBox(checked_contours.Bitmap);
            Cache.StoreCurrentBitmapData();}
        private void tableLayoutPanel1_Paint(object sender, PaintEventArgs e){}
        private void Scale1_KeyPress(object sender, KeyPressEventArgs e){
            if (e.KeyChar == ','){
                if (this.Text.IndexOf(",") >= 0 || this.Text.Length == 0){
                    e.Handled = true;}}
            else if ((e.KeyChar < '0' || e.KeyChar > '9') && e.KeyChar != '\b'){
                e.Handled = true;}}
        private void Scale1_Validating(object sender, CancelEventArgs e){
            float value;
            if (!float.TryParse(Scale1.Text, out value)) { Scale1.Text = ""; }}

        private void Gsmoth_CheckedChanged(object sender, EventArgs e){
            if (Gsmoth.Checked == true){
                MatrixY.Enabled = true;
                MatrixX.Enabled = true;
                Sigma.Enabled = true;}
            else{
                MatrixY.Enabled = false;
                MatrixX.Enabled = false;
                Sigma.Enabled = false;}}
        private void Sigma_Validating(object sender, CancelEventArgs e){
            float value;
            if (!float.TryParse(Scale1.Text, out value)) { Scale1.Text = ""; }}
        private void Sigma_KeyPress(object sender, KeyPressEventArgs e){
            if (e.KeyChar == ','){
                if (this.Text.IndexOf(",") >= 0 || this.Text.Length == 0){
```

```
                e.Handled = true;}}
            else if ((e.KeyChar < '0' || e.KeyChar > '9') && e.KeyChar != '\b'){
                e.Handled = true;}}
        private void button4_Click(object sender, EventArgs e){
            int Contour_size = Convert.ToInt32(thickness.Value);
            double lower_sensitivity_limit = Decimal.ToDouble(lower_sens.Value);
            double upper_sensitivity_limit = Decimal.ToDouble(upper_sens.Value);
            Bitmap bitmap = Cache.GetCurrentBitmap();
            Mat hierarchy = new Mat();
            Image<Bgr, byte> input_image = new Image<Bgr, byte>(bitmap);
            if (Gsmoth.Checked == true){
                Image<Bgr, byte> blured = new Image<Bgr, byte>(bitmap);
                CvInvoke.GaussianBlur(blured, input_image, new
Size(Decimal.ToInt32(MatrixX.Value), Decimal.ToInt32(MatrixY.Value)),
Convert.ToDouble(Sigma.Text));}
            Cache.SetMainformPictureBox(input_image.Bitmap);
            Cache.StoreCurrentBitmapData();}
        private void groupBox1_Enter(object sender, EventArgs e){}    }
    public class GravityPoint
    {internal int id;internal double X;
        internal double Y;
        public GravityPoint(int id, double X, double Y){
            this.id = id;
            this.X = X;
            this.Y = Y;}}}
    using System;
    using System.Collections.Generic;
    using System.ComponentModel;
    using System.Data;
    using System.Drawing;
```

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Management;
using System.IO;
using System.IO.Ports;
using System.Threading;
namespace PDI_Tarea2.Forms{
   public partial class USB : Form {
      public System.IO.Ports.SerialPort sport;
      public String filePath = string.Empty;
      Thread FileSender;
      private static EventWaitHandle waitHandle = new
ManualResetEvent(initialState: true);
      public USB(){
         InitializeComponent();
         foreach (String s in SerialPort.GetPortNames()){
            txtPort.Items.Add(s);}
         this.cmbbaudrate.SelectedIndex = 0;
         this.cmbparity.SelectedIndex = 2;
         this.cmbdatabits.SelectedIndex = 3;
         this.cmbstopbits.SelectedIndex = 0;
         cmdClose.Enabled = false;
         cmbSend.Enabled = false;
         btSend.Enabled = false;
         btStop.Enabled = false;
         btPause.Enabled = false;
          filePath = Cache.GetFileName();
         txPath.Text = filePath;}
```

```
        private void button1_Click(object sender, EventArgs e){
            txtPort.Items.Clear();
            foreach (String s in SerialPort.GetPortNames()){
                txtPort.Items.Add(s);}}
        public void serialport_connect(String port, int baudrate, Parity parity, int
databits, StopBits stopbits){
            DateTime dt = DateTime.Now;
            String dtn = dt.ToShortTimeString();          try{
                sport = new System.IO.Ports.SerialPort(port, baudrate, parity, databits,
stopbits);
                sport.Open();
                cmdClose.Enabled = true;
                cmbSend.Enabled = true;
                cmdConnect.Enabled = false;
                txtReceive.Text = ("[" + dtn + "] " + "З'єднано\n") + txtReceive.Text;
                sport.DataReceived                    +=                    new
SerialDataReceivedEventHandler(sport_DataReceived);}
            catch (Exception ex) { MessageBox.Show(ex.ToString(), "Помилка"); }}

        private          void          sport_DataReceived(object          sender,
SerialDataReceivedEventArgs e){
            DateTime dt = DateTime.Now;
            String dtn = dt.ToShortTimeString();
            txtReceive.Text = ("[" + dtn + "] " + "Прийнято: " + sport.ReadExisting()
+ "\n") + txtReceive.Text;}
        private void cmdConnect_Click(object sender, EventArgs e){
            String port = txtPort.Text;
            int baudrate = Convert.ToInt32(cmbbaudrate.Text);
            Parity parity = (Parity)Enum.Parse(typeof(Parity), cmbparity.Text);
```

```
            int databits = Convert.ToInt32(cmbdatabits.Text);
            StopBits stopbits = (StopBits)Enum.Parse(typeof(StopBits),
cmbstopbits.Text);
            serialport_connect(port, baudrate, parity, databits, stopbits);
            if (File.Exists(filePath) && (sport != null) && (sport.IsOpen)){
               btSend.Enabled = true;}}
         private void cmbSend_Click(object sender, EventArgs e){
            DateTime dt = DateTime.Now;
            String dtn = dt.ToShortTimeString();
            String data = txtDatatoSend.Text;
            sport.Write(data);
            txtReceive.Text = ("[" + dtn + "] " + "Відправлено: " + data + "\n") +
txtReceive.Text;}
         private void cmdClose_Click(object sender, EventArgs e){
            DateTime dt = DateTime.Now;
            String dtn = dt.ToShortTimeString();
            if (sport.IsOpen){
               sport.Close();
               cmdClose.Enabled = false;
               cmdConnect.Enabled = true;
               cmbSend.Enabled = false;
               txtReceive.Text = ("[" + dtn + "] " + "Відключено\n") +
txtReceive.Text;}}
         private void button2_Click(object sender, EventArgs e){
            openFileDialog1.Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*"
            if (openFileDialog1.ShowDialog() == DialogResult.OK){
               filePath = openFileDialog1.FileName;
               txPath.Text = filePath;}
            if (File.Exists(filePath) && (sport!=null) && (sport.IsOpen)){
               btSend.Enabled = true;}}
```

```
public class SendFile{
    public SerialPort sport;
    public Stream file;
    public int Delay;}

private void button3_Click(object sender, EventArgs e){
    cmdClose.Enabled = false;
    cmbSend.Enabled = false;
    btSend.Enabled = false;
    btPause.Enabled = true;
    btStop.Enabled = true;
    SendFile NewProgramSend = new SendFile();
    NewProgramSend.sport = sport;
    openFileDialog1.FileName =  filePath;
    NewProgramSend.file = openFileDialog1.OpenFile();
    NewProgramSend.Delay = Convert.ToInt32(nUDDelay.Value);
    FileSender = new Thread(new ParameterizedThreadStart(SendFileTo));
    FileSender.Start(NewProgramSend);}
public int GetDelay(){
    return Convert.ToInt32(nUDDelay.Value);}
public void AppendTextBox(string value){
    if (InvokeRequired){
        this.Invoke(new Action<string>(AppendTextBox), new object[] { value
});return;}
    txtReceive.Text = value + txtReceive.Text;}
public void CloseButtonEnable(bool value){
    if (InvokeRequired){
        this.Invoke(new  Action<bool>(CloseButtonEnable),  new  object[]  {
value });return;
    }cmdClose.Enabled = value;}
```

```
        public void SendButtonEnable(bool value){
            if (InvokeRequired){
                this.Invoke(new Action<bool>(SendButtonEnable), new object[] { value
});return;}
            cmbSend.Enabled = value;}
        public void SendFileButtonEnable(bool value){
            if (InvokeRequired){
                this.Invoke(new Action<bool>(SendFileButtonEnable), new object[] {
value });return;}
            btSend.Enabled = value;}
        public void SendFileTo (object obj){
            SendFile C = (SendFile)obj;
            using (StreamReader reader = new StreamReader(C.file)){
                while (reader.Peek() >= 0){
                    DateTime dt = DateTime.Now;
                    String dtn = dt.ToShortTimeString();
                    String data = reader.ReadLine();
                    C.sport.Write(data);
                    AppendTextBox("[" + dtn + "] " + "Відправлено: " + data +
Environment.NewLine);
                    Thread.Sleep(GetDelay());
                    waitHandle.WaitOne();}
                CloseButtonEnable(true);
                SendButtonEnable(true);
                SendFileButtonEnable(true);}}4
        private void btPause_Click(object sender, EventArgs e){
            if (btPause.Text == "Пауза"){
                waitHandle.Reset();
                btPause.Text = "Продовжити";}
```

```
    else {
        waitHandle.Set();
        btPause.Text = "Пауза";}}
private void btStop_Click(object sender, EventArgs e){
    FileSender.Abort();
    btStop.Enabled = false;
    btPause.Enabled = false;
    cmdClose.Enabled = true;
    cmbSend.Enabled = true;
    btSend.Enabled = true ;}
private void button3_Click_1(object sender, EventArgs e){
   // FileSender.Abort();
    this.Close();}}}
```