Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
магістр
(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА РЕАЛІЗАЦІЯ МЕТОДІВ ГЕНЕРАЦІЇ КЕРУЮЧОЇ ПРОГРАМИ ДЛЯ ВЕРСТАТА З ЧПК
RESEARCH AND IMPLEMENTATION OF METHODS FOR GENERATING CONTROL PROGRAMS FOR CNC MACHINES

Виконав: студент 2 курсу, групи КНТ-214м
Спеціальності 122 Комп'ютерні науки
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Системи штучного інтелекту

АРХИПОВ Д.В.
(ПРІЗВИЩЕ та ініціали)
Керівник ПАРХОМЕНКО А.В.
(ПРІЗВИЩЕ та ініціали)
Рецензент ГОЛУБ Т.В.
(ПРІЗВИЩЕ та ініціали)

2025

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук та технологій
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Системи штучного інтелекту
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
_____Сергій СУББОТІН
"____" _____2025 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

АРХИПОВА Данила Володимировича
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та реалізація методів генерації керуючої програми для верстата з ЧПК. Research and Implementation of Methods for Generating Control Programs for CNC Machines

керівник проєкту (роботи) к.т.н., доцент, ПАРХОМЕНКО Анжеліка Володимирівна,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
затверджені наказом закладу вищої освіти від "30" вересня 2025 року № 447
2. Строк подання студентом проєкту (роботи) 08 грудня 2025 року
3. Вихідні дані до проєкту (роботи) рекомендована література,

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань роботи. 2. Аналіз методів генерації керуючої програми. 3. Проєктування програмного забезпечення. 4. Конструювання програмного забезпечення. 5. Тестування та практичне впровадження.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ПАРХОМЕНКО А.В., доцент		
Нормоконтроль	БЄЛОВА А.В., асистент		

7. Дата видачі завдання « 30 » вересня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2-3 тижні	Розділ 1
3	Розробка та удосконалення методів, моделей й алгоритмів вирішення задачі	4-5 тижні	Розділ 2
4	Розробка архітектури програми.	6 тиждень	Розділ 3
5	Розробка програми.	7-8 тижні	Розділ 4
6	Тестування та експериментальне дослідження програмного забезпечення.	9 тиждень	Розділ 5
7	Оформлення пояснювальної записки та документів до неї.	10-11 тижні	Додатки
8	Нормоконтроль та рецензування.	12 тиждень	
9	Захист роботи.	12 тиждень	

Студент _____ Данило АРХИПОВ
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ Анжеліка ПАРХОМЕНКО
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра: 90 с., 5 табл., 33 рис., 2 дод., 41 джерело.

ВЕРСТАТ З ЧПК, CAM-СИСТЕМА, ПОСТПРОЦЕСОР, ТРАЄКТОРІЯ ОБРОБКИ, NC-ПРОГРАМА.

Об’єкт дослідження – процес формування керуючих програм для металорізальних верстатів з числовим програмним керуванням.

Предмет дослідження – методи, алгоритми та програмні засоби генерації NC-коду на основі CAM-траєкторій та формату CLDATA.

Мета роботи - програмна реалізація масштабованого, адаптивного постпроцесора для підвищення ефективності роботи обладнання з ЧПК.

Матеріали, методи та технічні засоби: мова програмування Python, бібліотеки стандартної математичної обробки, персональний комп’ютер із ОС Windows.

Результати. Створено програмний Windows-застосунок, здатний постпроцесувати дані CAM-траєкторії у коректний, безпечний та оптимізований G-code для контролера Siemens SINUMERIK.

Висновки. Впровадження розробленого постпроцесора забезпечує підвищення точності та відтворюваності процесу підготовки керуючих програм, зменшує залежність від кваліфікації оператора та покращує ефективність роботи обладнання з ЧПК.

Галузь використання – підприємства машинобудівної галузі, що працюють з верстатами Siemens SINUMERIK.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 90 p., 5 tables, 33 figures, 2 appendices, 41 sources.

CNC MACHINE, CAM SYSTEM, POSTPROCESSOR, MACHINING PATH, NC PROGRAM.

The object of the study is the process of forming control programs for metal-cutting machines with numerical program control.

The subject of the study is methods, algorithms and software for generating NC code based on CAM trajectories and the CLDATA format.

The purpose of the work is the software implementation of a scalable, adaptive postprocessor to increase the efficiency of CNC equipment functioning.

Materials, methods and technical means: Python programming language, standard mathematical processing libraries, a personal computer with Windows OS.

Results. A Windows software application has been created that is capable of post-processing CAM trajectory data into correct, safe and optimized G-code for the Siemens SINUMERIK controller.

Conclusions. The implementation of the developed postprocessor ensures increased accuracy and reproducibility of the process of preparing control programs, reduces dependence on operator qualifications and improves the efficiency of CNC equipment.

Field of application – enterprises of the machine-building industry working with Siemens SINUMERIK machine tools.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface;

CAD – Computer-Aided Design;

CAM – Computer-Aided Manufacturing;

CAE – Computer-Aided Engineering;

CLDATA – Cutter Location Data;

CNC – Computer Numerical Control;

NC-code – Numerical Control Code;

ML – Machine Learning;

SSA – Sparrow Search Algorithm;

КП – керуюча програма;

ПЗ – програмне забезпечення;

ПК – персональний комп’ютер;

ЧПК – числове програмне керування.

ВСТУП

Сучасний розвиток машинобудівної галузі характеризується широким застосуванням верстатів з числовим програмним керуванням (ЧПК), які забезпечують високу точність та продуктивність обробки деталей. Умови глобальної конкуренції та зростаючі вимоги до якості продукції висувають нові завдання до систем автоматизованої підготовки виробництва, серед яких ключове місце займає формування коректних керуючих програм для верстатів з ЧПК [1].

Формування керуючого коду є завершальним етапом у ланцюгу CAD–CAM–CNC і здійснюється за допомогою постпроцесорів, що виконують трансляцію даних із CAM-системи у формат, сумісний з конкретною системою керування верстата [1]. Хоча стандарт ISO 6983 визначає загальні принципи програмування, кожна система ЧПК має власні розширення, особливості синтаксису та специфічні технологічні цикли [2]. Це зумовлює залежність керуючих програм від конкретного обладнання та створює потребу у розробці якісних і надійних постпроцесорів.

Зростання складності обробки, зокрема впровадження багатокоординатних і п’ятиосьових технологій, розвиток високошвидкісного фрезерування та інтеграція верстатів у виробничі інформаційні системи посилюють вимоги до методів генерації керуючих програм [3]. У цих умовах ефективність виробництва значною мірою визначається можливістю отримання оптимізованого, безпечного та відтворюваного керуючого коду. Тому, тема роботи є актуальною.

Метою дипломної роботи є програмна реалізація масштабованого, адаптивного постпроцесора для підвищення ефективності роботи обладнання з ЧПК.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ РОБОТИ

1.1 Аналіз предметної області

1.1.1 Ланцюг цифрового виробництва

У сучасній машинобудівній промисловості підготовка виробництва дедалі більше сприймається не як сукупність окремих етапів, а як інтегрований цифровий ланцюг, або “цифрове виробництво”. Схему цифрового виробництва зображено на рис. 1.1.

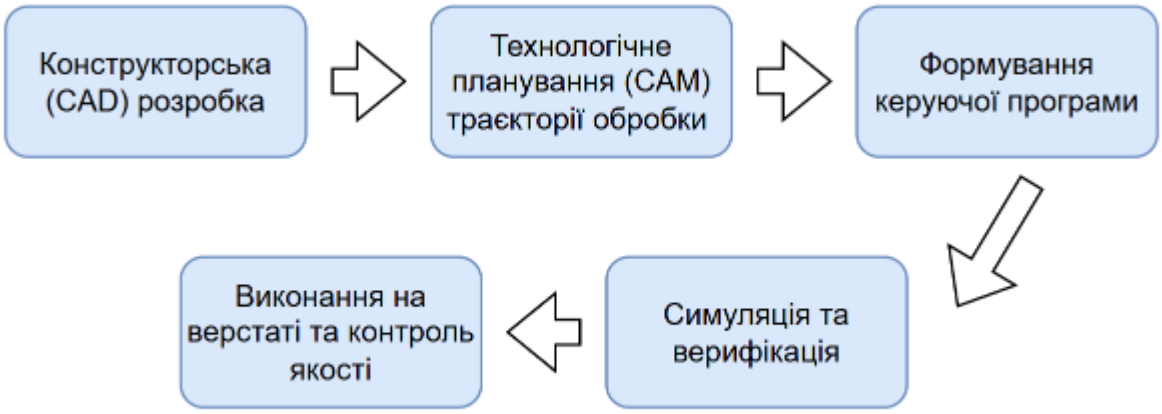

Рисунок 1.1 – Ланцюг цифрового виробництва [4]

Такий ланцюг забезпечує скорочення часу від появи ідеї до серійного виробництва, зменшує ймовірність помилок, пов’язаних з людським фактором, та підвищує повторюваність процесів [4].

Хоча класичні підходи до програмування ЧПК базуються на стандартах типу ISO 6983 (RS-274D), з розвитком технологій зростає потреба у більш семантично багатих та інтегрованих підходах [4]. З розвитком технологій з’явились потреби у виконанні складніших траєкторій, підтримці багатокоординатної, зокрема 5-осьової, обробки, адаптації під високошвидкісне різання, оптимізації подачі та контролю стану інструменту – це призвело до

розширень стандарту в окремих виробників і розробці модифікацій. Для покращення та оптимізації виробничого процесу сучасні інженери розробляють нові стандарти та шукають нові підходи до створення керуючих програм, контролю за якістю кінцевого вироба та швидкістю підбору параметрів [4].

Зокрема, STEP-NC (ISO 14649, ISO 10303 AP-238) пропонує опис процесу обробки на вищому рівні, що дозволяє обмінюватися інформацією про геометрію, технологічні етапи та допуски між CAD/CAM і контролером; дослідження показують, що це підвищує адаптивність і автоматизацію цифрового ланцюга [5].

Паралельно, концепція цифрового двійника (Digital Twin) – віртуальна копія машини та технологічного процесу, уможливлює створення віртуальної копії верстата та технологічного процесу для реалістичної симуляції й моніторингу виконання програм у режимі близькому до реального часу, що сприяє оперативному виявленню відхилень і підвищенню безпеки процесу [6]. Цифровий двійник став передовою темою досліджень останніми роками та важливим напрямком розвитку інтелектуального виробництва. Для обробки з числовим програмним керуванням система цифрового двійника може бути використана як інтелектуальний центр моніторингу та аналізу, відображаючи реальний процес обробки у віртуальному середовищі [6].

Також, методи машинного навчання намагаються доповнити традиційну CAM-логіку. Машинне навчання – інтелектуальний метод прийняття рішень щодо параметрів програмування CAM ЧПК на основі визначення на основі моделі (MBD). Ієрархічна модель процесу використовує технологію MBD для вираження потоку процесу та управління інформацією про процес [7]. Тіснота між процесом та обробкою на ЧПК посилюється шляхом сортування кроків обробки на ЧПК та автоматичного вилучення інформації про процес. Потім використовується нейронна мережа зворотного поширення (BP) для аналізу досвіду програмування ЧПК в історичних випадках. Алгоритм пошуку Sparrow (SSA) та інші методи використовуються для оптимізації початкової ваги та

порогу, швидкості навчання та процесу зворотного зв'язку нейронної мережі BP [7].

Отже, сьогодні практична генерація керуючого коду здебільшого означає якісну CAM-стратегію + коректний постпроцесор під конкретний ЧПК + керовану передачу програм та валідацію. Перехід до нових технологій поки не змінює цю реальність, але задає вектор розвитку на “розумні” дані та зворотний зв’язок.

1.1.2 Технологічні особливості та практичні вимоги до CAM систем та постпроцесорів

Окрім загальної структури цифрового ланцюга, існує сукупність технологічних вимог, які визначають ефективність і надійність генерації керуючих програм. При обробці складних поверхонь траєкторії мають бути достатньо деталізованими для досягнення потрібної точності, проте без надлишкової сегментації, що може перевантажити інтерполятор контролера; баланс деталізації й компактності траєкторії – ключове завдання CAM-алгоритмів [8].

Для правдоподібного прогнозування часу циклу і забезпечення якості обробки постпроцесор разом із CAM має враховувати динамічні характеристики системи – обмеження на прискорення й швидкості, інерційні ефекти та поведінку інтерполятора – що дозволяє більш точно оцінити фактичний час обробки й уникнути небажаних динамічних помилок [9]. Важливим є контроль геометричних похибок: підтримка обмежень на кривизну, адаптивна подача залежно від локальної геометрії та компенсація похибок, пов’язаних із зносом інструменту або тепловими деформаціями.

Крім технологічних аспектів, постпроцесор повинен забезпечувати сумісність із синтаксисом конкретного контролера (формат чисел, системи координат, набір M- і G-кодів), мінімізувати холості переходи та оптимізувати

логіку змін інструментів, автоматично проводити перевірку на можливі колізії та вихід за межі ходів осей, а також бути масштабованим для налаштування під різні конфігурації верстатів. Інтеграція постпроцесора із засобами симуляції та моніторингу (зокрема цифровими двійниками) підвищує надійність і дозволяє виконувати попередні перевірки ще до фізичного запуску програми [10].

Таким чином, вимоги до CAM і постпроцесорів поєднують у собі точні технологічні критерії (деталізація траєкторій, динамічний прогноз, контроль похибок) та практичні аспекти сумісності, безпеки і масштабованості, що вказує на нерозривний зв’язок між постпроцесором та CAM системою, для якої він розроблений.

1.2 Огляд парку верстатів підприємства та особливостей постпроцесорів для них

1.2.1 FANUC

FANUC – одна з найпоширеніших платформ числового програмного керування у світовій промисловості. Контролери FANUC підтримують класичний формат ISO-G-коду, параметричне програмування та внутрішні цикли для різних технологічних операцій, таких як свердління, зенкування, нарізання різьби тощо. Крім того, платформа підтримує реалізації 4–5-осьової обробки через специфічні G-команди та функції керування орієнтацією інструменту.

Постпроцесор для FANUC формує коректні блоки керуючої програми, дотримується форматів координат і числових значень, контролює модальні команди (перемикання між G90/G91, G17/G18/G19, G43/G44 тощо) та забезпечує правильну послідовність M-кодів. Важливим аспектом є також дотримання обмежень на довжину рядка, перевірка на перевищення ходів осей і правильна генерація підпрограм.

Згідно з даними офіційної документації FANUC [11], одним із ключових принципів цієї системи є стабільність та сумісність між різними поколіннями контролерів, що дозволяє інтегрувати старі та нові моделі верстатів в одну виробничу мережу.

Таким чином, контролери FANUC забезпечують високу надійність, проте вимагають ретельного налаштування постпроцесора через різноманітність реалізацій М-кодів серед виробників обладнання.

1.2.2 Siemens SINUMERIK

Контролери Siemens SINUMERIK серії 840D і 828D вирізняються гнучкістю, високою точністю та широкими можливостями для багатокоординатної і п’ятиосьової обробки. Система має розвинені засоби діалогового програмування, підтримку кінематичних трансформацій, циклів для позиціонування, свердління, фрезерування, а також можливість використання розширених параметрів і R-змінних [12].

SINUMERIK підтримує функцію TRAORI, яка забезпечує точне керування орієнтацією інструменту відносно поверхні обробки. Це особливо важливо при п’ятиосьових операціях, де точність розрахунку кутів нахилу визначає якість поверхні. Постпроцесор для SINUMERIK повинен генерувати виклики TRAORI у відповідні моменти програми та передавати параметри орієнтації відповідно до активної кінематичної моделі верстата.

Перевагою системи є наявність структурованого синтаксису, який робить керуючі програми більш зрозумілими для оператора, а також можливість адаптації під конкретні виробничі завдання. Як зазначено у дослідженні [13], SINUMERIK відзначається високою стабільністю при багатокоординатній обробці, простотою інтеграції з CAD/CAM-системами та підтримкою «цифрових двійників» у середовищі Siemens NX.

Отже, SINUMERIK поєднує точність, зручність і розширюваність, що робить його оптимальним вибором для впровадження у високотехнологічне виробництво.

1.2.3 HEIDENHAIN TNC

Сімейство HEIDENHAIN TNC відоме своїм орієнтованим на користувача підходом і зручним текстовим форматом Klartext. Завдяки цьому програми легко читаються і редагуються безпосередньо на панелі управління верстата. TNC контролери мають розвинену систему діалогів і потужні цикли обробки, зокрема CYCL DEF для свердління, нарізання різьби, розточування тощо [14].

Особливістю HEIDENHAIN є можливість використання комбінації діалогових і традиційних G-команд у межах однієї програми. Це дозволяє оператору гнучко змінювати логіку виконання операцій без складних корекцій коду. Крім того, Klartext-коди полегшують візуальну перевірку програм і знижують ризик помилок на етапі налагодження.

Дослідження [15] відзначають, що HEIDENHAIN TNC активно використовується у прецизійній обробці завдяки своїм розвиненим засобам компенсації похибок і точному контролю положення інструменту.

Таким чином, контролери HEIDENHAIN забезпечують високу точність і зручність для оператора, але вимагають спеціалізованого постпроцесора з підтримкою специфічного формату Klartext і циклів CYCL DEF.

1.2.4 Порівняльна характеристика систем ЧПК

На сучасному ринку ЧПК існує багато різноманітних систем, призначених для виконання різного профілю задач.

Порівняльна характеристика систем ЧПК наведена у табл. 1.1.

Таблиця 1.1 – Порівняльна характеристика систем ЧПК [11]-[15]

Критерій	FANUC	SINUMERIK	HEIDENHAIN
Зручність програмування	Складний синтаксис	Інтуїтивна структура	Простий формат Klartext
Підтримка 5-осьової обробки	Так, із розширеннями	Так, через TRAORI	Так, через PLANE SPATIAL
Гнучкість і адаптація	Середня	Висока	Висока
Інтеграція з CAD/CAM	Обмежена	Повна з NX	Часткова
Зручність для оператора	Середня	Висока	Дуже висока

Отже, Siemens SINUMERIK є найбільш збалансованим рішенням серед розглянутих систем за критеріями точності, зручності, інтеграції та автоматизації. Також система SINUMERIK має повну інтеграцію з цифровим ланцюгом виробництва, що робить її більш переважною з точки зору програміста.

1.3 Методика розробки керуючого коду для верстата з ЧПК

Методику розробки керуючого коду, або NC-програми, можна умовно розбити на декілька етапів. Блок-схему методики зображено на рис. 1.2. Методика розробки керуючого коду складається з низки послідовних блоків:

Блок 1. Підготовка вихідних даних – на цьому етапі виконується аналіз 3D-моделі деталі, вибір технологічних баз та інструменту, а також визначення основних режимів різання. Формуються всі необхідні параметри, що забезпечують коректність подальшого моделювання та генерування траєкторій [16].

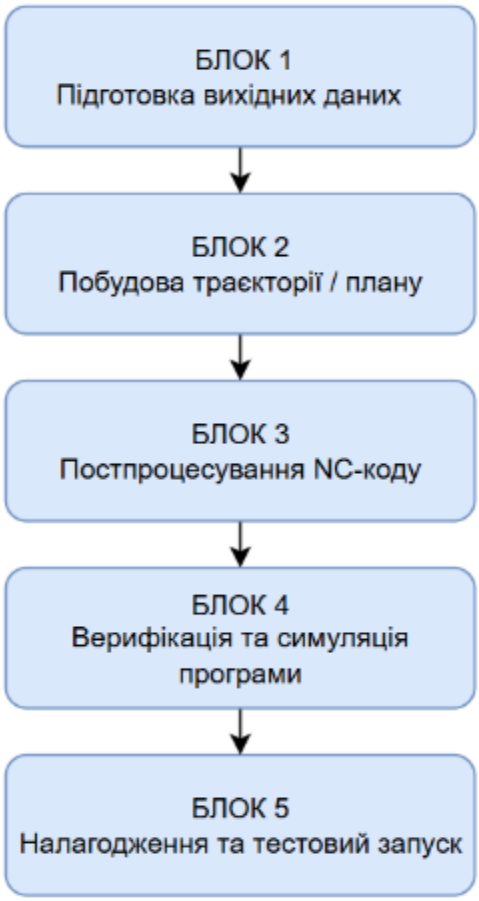

Рисунок 1.2 – Методика розробки керуючого коду [16]-[19]

Блок 2. Побудова траєкторії/плану – на цьому етапі формується логічна та послідовна структура рухів або дій, необхідних для досягнення поставленої технологічної мети. Визначаються послідовність переходів, характер обробки, типи операцій, а також ключові параметри, що впливають на шлях інструмента або порядок виконання команд. Результатом є узгоджений план обробки, який слугує основою для подальшої генерації керуючої програми [17].

Блок 3. Постпроцесування NC-коду – створені траєкторії перетворюються у керуючу програму відповідно до синтаксису контролера Siemens SINUMERIK. Постпроцесор формує службові блоки, встановлює необхідні модальні команди та забезпечує сумісність NC-коду з кінематикою конкретного верстата [18].

Блок 4. Верифікація та симуляція програми – генерована програма перевіряється у CAM-середовищі або симуляторі ЧПК на наявність помилок,

можливих колізій та перевищення робочих ходів. Цей етап гарантує коректність поведінки інструменту та відповідність програми технологічним вимогам [19].

Блок 5. Налагодження та тестовий запуск – у завершальній фазі програма відпрацьовується на верстаті в режимі тестового прогону з подальшим коригуванням параметрів за потреби. Після підтвердження працездатності NC-коду виконується остаточне налаштування і програма допускається до виробничого використання.

Ці етапи можуть відрізнятися на різних підприємствах та у різних умовах праці, а також повторюватись навіть після введення програми у використання.

Отже, методика поєднує інженерний аналіз, CAM-планування, постпроцесування, симуляцію та практичне налагодження для досягнення точності й безпеки обробки.

1.4 Постановка завдань дипломної роботи

За результатами аналізу процесу та специфіки сучасної розробки NC-програм, можна виділити наступні виклики:

– існуючі відмінності в інтерпретації G-коду у різних верстатів з ЧПК;

– недостатня універсальність стандартних постпроцесорів, які вбудовані в CAM-програми;

– висока залежність якості програми від кваліфікації оператора та програміста.

Проблема полягає в тому, що вбудовані постпроцесори зазвичай є закритими, не забезпечують необхідної гнучкості, оскільки заточені під найбільш популярні верстати і не надають можливостей тонкого налаштування (наприклад ввімкнення корекції по G41/G42, постпроцесування траєкторій за векторами замість точок, вимкнення TRAORI і т.д.). Тому, тема роботи є актуальною.

Метою дипломної роботи є програмна реалізація масштабованого, адаптивного постпроцесора для підвищення ефективності роботи обладнання з ЧПК.

Завданнями дипломної роботи є:

– провести аналіз сучасних методів генерації керуючої програми;

– виконати аналіз вимог до постпроцесорів для найбільш популярних контролерів;

– розробити постпроцесор під верстати з контролером Siemens SINUMERIK;

– створити траєкторії та постпроцесувати їх за допомогою розробленого постпроцесора;

– виконати симуляцію та перевірку отриманих програм;

– сформувати рекомендації щодо використання результатів у виробничому процесі.

2 АНАЛІЗ МЕТОДІВ ГЕНЕРАЦІЇ КЕРУЮЧОЇ ПРОГРАМИ

2.1 Класифікація методів

Генерація керуючої програми (КП) для верстатів із числовим програмним керуванням (ЧПК) є одним із найважливіших етапів технологічної підготовки виробництва. Саме на цьому етапі віртуальні моделі переходять у конкретні інструкції для обладнання. Від того, яким методом створюється керуючий код, залежать точність, стабільність, безпека та продуктивність обробки.

Згідно з дослідженням [20], методи генерації керуючих програм можна поділити за рівнем автоматизації, алгоритмічною основою та способом взаємодії оператора з системою. У роботі [21] подано систематизацію методів формування траєкторій для ЧПК на основі структурованих даних та моделей процесу обробки.

Усі відомі методи формування КП можна класифікувати за ступенем автоматизації на три основні групи. Схему класифікації методів зображено на рис. 2.1.

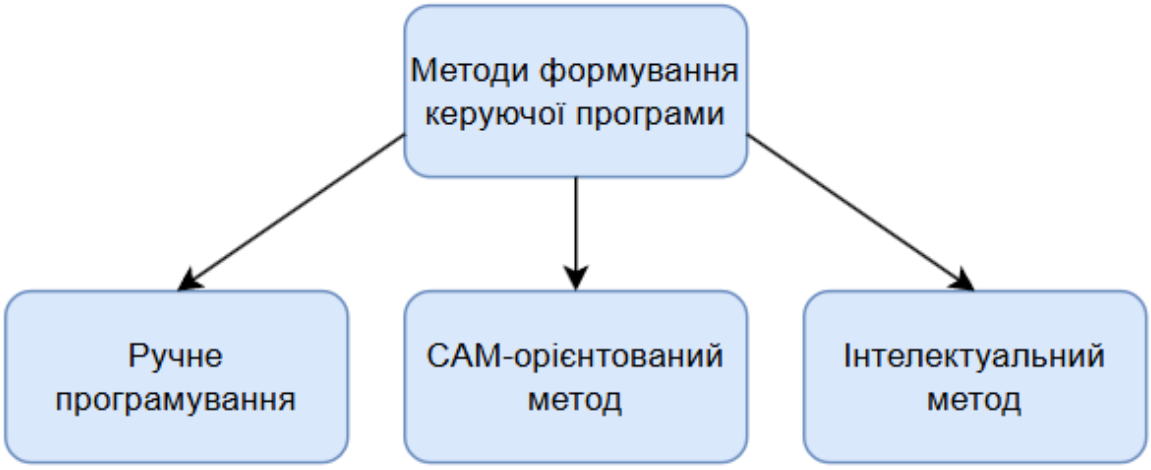

Рисунок 2.1 – Методи формування керуючої програми [22]-[24]

Кожен із цих підходів має свої переваги й обмеження, тому вибір методу визначається типом виробництва, складністю деталі, доступними ресурсами та вимогами до точності. У більшості сучасних досліджень підкреслюється, що тенденція розвитку полягає у переході від ручного до автоматизованого програмування з подальшим використанням адаптивних інтелектуальних систем [25].

Отже, класифікація методів формування КП відображає поступовий перехід від ручного управління до комплексної цифрової автоматизації виробництва, що є природним наслідком розвитку технологій сучасної індустрії машинобудування.

2.2 Ручне програмування

Ручне програмування є класичним методом створення керуючих програм, який історично з’явився першим. Програміст самостійно формує G- та M-коди, визначає координати руху інструменту, швидкості подачі, обертання шпинделя, зміни інструменту й усі інші параметри процесу обробки [26]. Цей підхід дає повний контроль над процесом і дозволяє врахувати специфічні вимоги до кожної операції. Він використовується, коли потрібно виготовити одиничні деталі, виконати експериментальні роботи або швидко скорегувати існуючу програму без CAM-системи [27].

Разом із тим ручне програмування має суттєві недоліки: воно потребує високої кваліфікації оператора, займає багато часу, не дозволяє ефективно працювати зі складними 3D-поверхнями та є схильним до помилок. Крім того, складність контролю логічних залежностей у великій програмі збільшує ризик помилкових команд, що може призвести до зупинки процесу або пошкодження інструменту [28].

Отже, ручне програмування виправдане лише у випадках невеликої серії або відсутності систем автоматизованого проєктування, тоді як у сучасному машинобудуванні воно поступово витісняється автоматизованими методами.

2.3 Методи CAM-генерації траєкторії

CAM-генерація керуючої програми є найпоширенішим і найбільш технологічно розвиненим методом створення програм для верстатів із ЧПК. У цьому підході процес розпочинається з імпортування геометричної моделі виробу з CAD-системи до CAM-модуля, де формується цифровий опис траєкторій інструменту. Система автоматично визначає тип операцій (фрезерування, свердління, токарна обробка тощо), обирає інструмент, режими різання та спосіб переходів між контурами. Далі за допомогою постпроцесора траєкторії перетворюються у формат сумісний із конкретною системою ЧПК.

Дослідження [29], [30] описують широкий спектр алгоритмів побудови траєкторій, зокрема рівневі (layer-based), контурні (contour-parallel), адаптивні та комбіновані стратегії. Сучасні CAM-системи враховують динаміку руху осей, обмеження на прискорення, інерційні характеристики й навіть вібраційні моделі верстата.

Крім того, новітні підходи включають адаптивне регулювання подачі на основі кривизни поверхні та прогнозу навантаження на інструмент, що дозволяє продовжити його ресурс і забезпечити стабільну якість поверхні [31]. Важливою перевагою CAM-методів є можливість візуалізації та симуляції процесу – це дає змогу виявити потенційні колізії та помилки ще до запуску програми на реальному верстаті.

Отже, CAM-генерація поєднує високу швидкість створення програм, гнучкість налаштування параметрів та інтеграцію з іншими цифровими системами, що робить її найдоцільнішим методом для сучасного виробництва.

2.4 Автоматизація та інтелектуальні методи

Інтелектуальні методи генерації керуючих програм є логічним розвитком CAM-підходу. Їх мета – мінімізувати участь людини в процесі створення програми шляхом використання штучного інтелекту, графів знань і цифрових двійників.

Згідно з дослідженням [32], автоматичне формування G-коду може здійснюватися на основі доменного моделювання – створення бібліотек шаблонів технологічних процесів і правил відображення між параметрами моделі та командами керування. Це дозволяє створювати програми без ручного втручання, адаптуючи їх під конкретне обладнання.

Інший підхід, описаний у [33], базується на графах знань: система зберігає зв’язки між геометричними елементами, інструментами й параметрами режимів обробки. За допомогою алгоритмів машинного навчання система аналізує попередні успішні обробки та формує нові програми, які відповідають заданим вимогам точності та часу.

Такі технології відкривають перспективи створення самонавчальних виробничих систем, де ЧПК може адаптуватися до зносу інструменту, змін матеріалу або навіть до деформацій деталі в процесі обробки.

Таким чином, автоматизація на основі інтелектуальних методів є наступним кроком після CAM-технологій, забезпечуючи адаптивність і самостійність виробничих систем, але на даний момент все ще недостатньо дослідженою та адаптованою до сучасного виробничого процесу.

2.5 Порівняння методів та обґрунтування вибору

Порівняльна характеристика методів генерації керуючої програми наведена у табл. 2.1.

Таблиця 2.1 – Порівняльна характеристика методів генерації керуючої програми [26]-[33]

Критерій	Ручне програмування	CAM-методи	Інтелектуальні методи
Рівень автоматизації	Низький	Високий	Дуже високий
Точність	Середня	Висока	Потенційно дуже висока
Швидкість підготовки	Низька	Висока	Висока після навчання системи
Залежність від оператора	Максимальна	Середня	Мінімальна
Підтримка складних 3D поверхонь	Обмежена	Повна	Повна
Відтворюваність результату	Низька	Висока	Дуже висока

Порівнюючи розглянуті методи, можна зробити висновок, що кожен із них має свою сферу ефективного застосування.

Ручне програмування залишається корисним лише для простих деталей і одиничного виробництва. Інтелектуальні методи демонструють значний потенціал, але їхня повномасштабна реалізація потребує складного програмного забезпечення, великих обсягів даних і відповідної технічної бази, що поки що обмежує їхнє промислове використання.

Натомість CAM-генерація керуючих програм забезпечує оптимальний баланс між автоматизацією, точністю та надійністю. Вона поєднує високу швидкість розробки програм із можливістю тонкого налаштування траєкторій і

параметрів різання, а також дозволяє інтегрувати процес у загальний цифровий ланцюг підприємства.

Проведений аналіз методів формування керуючих програм показав наявність суттєвих відмінностей між ручним, CAM-орієнтованим та інтелектуальним підходами. Встановлено, що CAM-генерація забезпечує оптимальне поєднання точності, автоматизації та інтеграції з виробничим цифровим ланцюгом. Тому в роботі обрано підхід, який включає:

– використання CAM-системи для побудови траєкторій;

– розробку постпроцесора під контролер Siemens SINUMERIK;

– симуляцію та перевірку NC-коду за допомогою верифікаційних систем.

Запропонований підхід буде реалізований у практичній частині роботи.

2.6 Висновки

У цьому розділі було розглянуто класифікацію методів формування керуючої програми, виконано аналіз ручного, CAM-орієнтованого та інтелектуального підходів. На основі порівняльної характеристики встановлено, що CAM-методи забезпечують оптимальний баланс між точністю, швидкістю підготовки та інтеграцією у цифровий виробничий ланцюг. Саме тому в подальшій частині роботи використано метод CAM-генерації траєкторій у поєднанні з власним постпроцесором для SINUMERIK, що дозволяє забезпечити надійне й відтворюване формування NC-коду.

3 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз вимог

Програмний забезпечення, власне постпроцесор, призначено для автоматизованого перетворення траєкторних даних формату CLDATA у коректний NC-код, сумісний із системою керування Siemens SINUMERIK. Постпроцесор повинен виконувати зчитування та інтерпретацію вихідних записів від CAM-системи, перетворювати орієнтаційні вектори у кути обертання відповідно до кінематики верстата, забезпечувати коректну обробку режимів корекції інструмента (CUTCOM), оптимізувати модальний вихід та формувати готовий до завантаження на верстат NC-файл [23].

До функціональних вимог можна віднести:

– імпорт і читання CLDATA-файлу;

– розбір команд руху GOTO з отриманням координат та орієнтаційного вектора;

– нормалізацію векторів напряму;

– обчислення кутів A і C відповідно до кінематики верстата;

– обробку режимів CUTCOM (LEFT, RIGHT, OFF);

– формування NC-коду для SINUMERIK із модальною оптимізацією;

– визначення та встановлення необхідних режимів різання (подачі та швидкості обертання шпинделя);

– генерацію службових блоків (заголовок, завершення програми);

– обробку помилкових або неповних CLDATA-рядків.

Кожна з перелічених вимог є критичною для забезпечення коректності та безпечності роботи постпроцесора. Перші етапи – імпорт CLDATA та розбір команд GOTO – формують основу подальших розрахунків, оскільки від правильності інтерпретації вихідних траєкторних даних залежить коректність усієї програми [8]. Нормалізація векторів і розрахунок кутів A і C є критично

важливими, тому потребують застосування стійких математичних методів. Оскільки SINUMERIK працює з кутами нахилу, а CLDATA містить орієнтацію у вигляді вектора, ці функції забезпечують правильну адаптацію CAM-траєкторій до кінематики верстата [8]. CUTCOM є одним з найбільш складних аспектів, тому логіка роботи з ним має передбачати відкладене вмикання та плавне вимкнення через компенсаційні переходи. Таке рішення підвищує безпеку та відповідність вимогам SINUMERIK. Також дуже важливим є обробка FEDRAT. Ця команда встановлює режими різання на кожному етапі обробки, тому критично важливо в кінцевому NC-коді передбачити логіку встановлення необхідних швидкостей подачі та обертів шпинделя. Модальна оптимізація зменшує кількість повторюваних команд, покращує читабельність та підвищує ефективність виконання програми на верстаті [8].

До нефункціональних можна віднести вимоги, за якими програмний застосунок повинен:

- бути портативним і не залежати від складних зовнішніх бібліотек;
- забезпечувати стабільну швидкість обробки файлів різного обсягу;
- мати модульну та масштабовану архітектуру;
- давати детермінований результат при однакових вхідних даних;
- бути зручним у супроводі та налагодженні;
- мати захист від аварійних завершень у разі некоректних даних.

Портативність і відсутність тяжких залежностей забезпечать простоту розгортання на робочих місцях виробництва. Масштабованість архітектури досягається модульністю, що дозволяє додавати підтримку нових контролерів без перекомпіляції основного ядра [12]. Детермінованість потребує належного керування внутрішнім станом. Документованість і журналювання спростять відтворення проблем під час введення у виробництво [12]. Вимоги безпеки реалізуються через захищені fallback-сценарії і попередні перевірки даних.

Отже, сформовані вимоги визначають повний обсяг функціональних та технічних характеристик постпроцесора, охоплюючи підтримку формату

CLDATA, виконання трансформацій координат, коректну обробку CUTCOM та оптимізацію NC-коду, що створює передумови для побудови надійного та ефективного програмного забезпечення.

3.2 Архітектура системи

Архітектура постпроцесора визначає його місце у цифровому виробничому ланцюгу та описує взаємодію з зовнішніми системами, що беруть участь у підготовці та виконанні керуючих програм. Постпроцесор функціонує як проміжний спеціалізований компонент між CAM-системою та обладнанням із ЧПК, забезпечуючи адаптацію траєкторних даних до кінематики конкретного верстата. Місце построцесора у загальній структурі цифрового виробництва представлено на рис. 3.1.

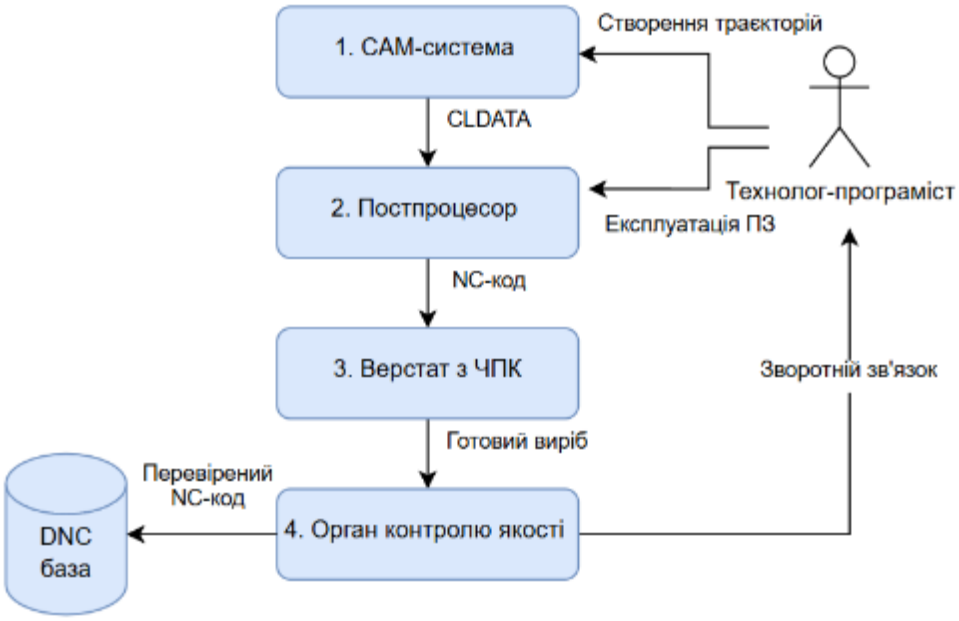

Рисунок 3.1 – Місце построцесора у загальній структурі цифрового виробництва [22]

Постпроцесор виконує роль проміжної ланки між CAM-системою і контролером ЧПК. Його архітектура передбачає прийняття даних у стандартизованому форматі, їх подальшу інтерпретацію, математичну обробку, перетворення орієнтаційних векторів у кути нахилу, застосування корекційних режимів та формування NC-коду у вигляді послідовності команд, які можуть бути безпосередньо виконані верстатом [22]. Таким чином, він з'єднує високорівневу траєкторну інформацію, створену CAM-системою, з низькорівневими інструкціями, які очікує система керування SINUMERIK.

Крім взаємодії із зовнішніми технологічними підсистемами, постпроцесор забезпечує взаємодію із користувачем – оператором або технологом-програмістом. Через графічний інтерфейс користувач визначає шлях до CLDATA-файлу та місце збереження вихідного NC-коду і його формат, а також отримує повідомлення про успішність виконання операцій або наявність помилок у вхідних даних.

Архітектура постпроцесора орієнтована на масштабованість і сумісність з існуючими виробничими системами. Завдяки відокремленню логіки обробки від графічного інтерфейсу програмний застосунок може бути інтегровано у більші системи, зокрема у PLM- або MES-платформи, у яких процес постпроцесування може виконуватися автоматично [24]. Аналогічно, ядро постпроцесора може бути використане як окремий модуль у складі інших CAM-рішень або як частина автоматизованих систем генерації NC-коду на підприємстві.

Отже постпроцесор виступає критично важливим елементом, який перетворює параметричні дані CAM-модуля на інструкції, придатні для виконання верстатом, забезпечуючи безперервність, узгодженість і технологічну цілісність у всьому ланцюгу цифрового виробництва, що робить його критично важливою ланкою ланцюга цифрового виробництва.

3.3 Вибір середовища та інструментарію розробки

3.3.1 Вибір середовища розробки

Під час розробки постпроцесора було здійснено оцінювання трьох поширених середовищ програмування: PyCharm, Visual Studio Code та Sublime Text. Порівняльну характеристику середовищ розробки наведено у табл. 3.1.

Таблиця 3.1 – Порівняльна характеристика середовищ розробки [34]-[37]

Критерій	PyCharm	VS Code	Sublime Text
Тип засобу	Повноцінне IDE	Легкий модульний редактор	Мінімалістичний редактор
Продуктивність	Високі вимоги до ресурсів	Висока швидкість на різних системах	Дуже висока швидкість
Підтримка Python	Розширена інтеграція	Можлива через плагіни	Обмежена
Налагодження	Потужні засоби	Доступні через плагіни	Мінімальні
Розширюваність	Помірна	Дуже висока	Обмежена
Підтримка Git	Вбудована	Реалізується розширеннями	Базова
Зручність для CLI-проєктів	Надлишкова складність	Оптимальна	Обмежена
Вартість	Частково платне	Безкоштовне	Платне
Загальна придатність	Підходить для великих проєктів	Найкращий баланс параметрів	Підходить для простих скриптів

Порівняльний аналіз показує, що Visual Studio Code найбільш відповідає вимогам до розробки легких, проте технічно складних програмних інструментів. Він забезпечує швидку роботу, зручні інструменти налагодження, інтеграцію з Git та можливість розширення функціоналу за допомогою великої кількості плагінів.

3.3.2 Вибір мови програмування

Для розробки постпроцесора було розглянуто три мови програмування: Python, C++ та JavaScript. Порівняльну характеристику мов програмування подано у табл. 3.2.

Таблиця 3.2 – Порівняльна характеристика мов програмування [38]-[41]

Критерій	Python	C++	JavaScript
Складність синтаксису	Низька	Висока	Низька
Швидкість розробки	Висока	Низька	Висока
Продуктивність	Середня	Дуже висока	Середня
Портативність	Відмінна	Висока	Висока
Робота з текстом	Зручна, вбудована	Потребує додаткових механізмів	Добра
Зручність реалізації алгоритмів	Висока	Складна	Обмежена
Придатність для постпроцесорів	Висока	Потребує значних зусиль	Менш оптимальна
Підтримка бібліотек	Дуже широка	Дуже широка	Обмежена

Python відзначається простою структурою, високою читабельністю та широкою підтримкою інструментів для роботи з текстовими файлами. C++ забезпечує максимальну продуктивність та контроль над ресурсами, проте є значно складнішим у розробці та супроводі. JavaScript може використовуватися для створення консольних застосунків, однак його екосистема менш орієнтована на математичні обчислення та задачі, пов’язані з генерацією NC-коду. Порівняльний аналіз демонструє, що Python є найкращим вибором для реалізації постпроцесора завдяки швидкості розробки, простоті синтаксису та високій зручності реалізації алгоритмів обробки координат і формування NC-коду. Його гнучкість і кросплатформеність роблять його найоптимальнішим рішенням для подібних програмних інструментів.

3.4 Висновки

У цьому розділі виконано аналіз вимог до постпроцесора, побудовано архітектурну модель програмного забезпечення та обґрунтовано вибір середовища розробки і мови програмування.

У якості середовища розробки обрано Visual Studio Code. У якості мови програмування – Python.

Представлені рішення забезпечують логічну структурованість системи, технічну узгодженість алгоритмів і створюють надійну основу для подальшої реалізації програмного забезпечення у наступному розділі.

4 КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Структура програми

Структура постпроцесора буде побудована на трирівневому принципі, що забезпечує логічну структурованість, гнучкість і можливість подальшого розширення. Схему цієї структури представлено на рис. 4.1.

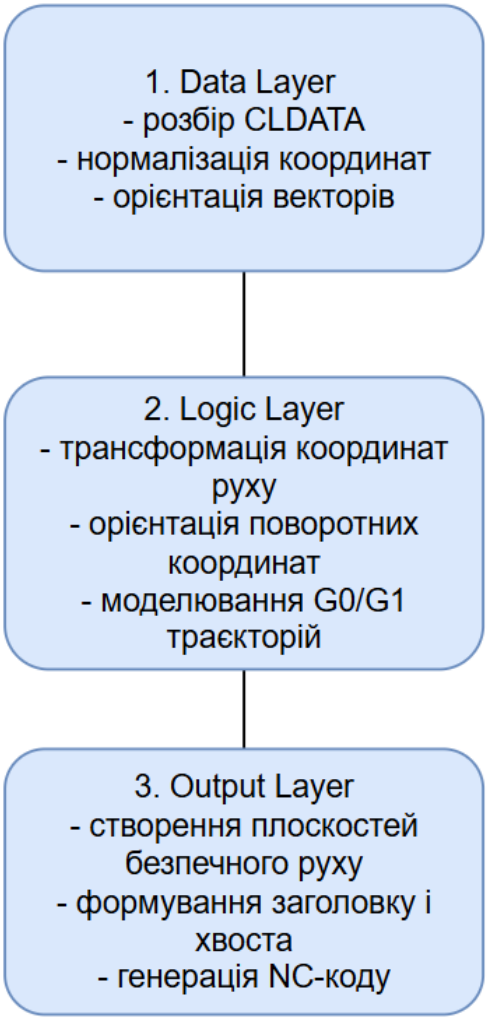

Рисунок 4.1 – Структурна схема системи

Перший рівень – відповідає за читання вихідних CLDATA-рядків і їх трансформацію у внутрішній формат. Саме тут здійснюється розбір команд, нормалізація траєкторних точок і орієнтаційних векторів, визначення технологічних параметрів та формування первинного представлення траєкторії.

Другий рівень – формує основну частину обчислювальних процесів. На цьому рівні реалізується трансформаця координат відповідно до кінематичних обмежень, обчислення кутів повороту стола, вибір модальних режимів переміщення та обробка корекцій CUTCOM із дотриманням вимог до послідовності команд. Тут також виконується оптимізація виводу, що дозволяє зменшити кількість NC-рядків, виключаючи дублювання.

Третій рівень – забезпечує побудову остаточного NC-коду. На цьому рівні створюється заголовок програми, формується безпечна траєкторія підходу, записуються робочі переміщення та генеруються завершальні команди. Таким чином, вихідний файл має структурований вигляд та відповідає вимогам контролера SINUMERIK.

Таким чином, представлена структура є модульною, що створює умови для відокремлення кожної функціональної частини в окремі модулі під час масштабування проєкту. Такий підхід полегшує налагодження, супровід та можливу адаптацію під інші CNC-системи.

4.2 Схема алгоритму функціонування програми

Функціонування постпроцесора виконується за послідовною схемою, що включає етап читання вхідних даних, їх трансформацію та формування вихідної програми. Загальний алгоритм роботи програми представлено на рис. 4.2. Схема складається з наступних блоків.

Блок 1. Початок виконання алгоритму.

Блок 2. Читання вхідного файлу з CLDATA. На цьому етапі здійснюється відкриття та покрокове зчитування CLDATA-файлу. Виконується початковий синтаксичний розбір рядків, усунення зайвих пробілів і службових символів, а також накопичення сирих записів у внутрішню структуру для подальшої обробки. Результатом є набір підготовлених рядків, структурований для наступних етапів обробки даних.

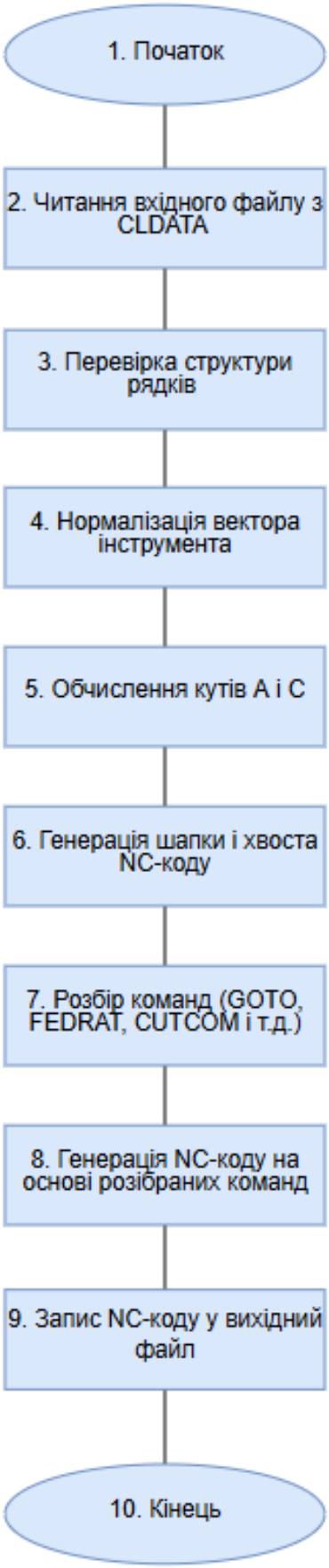

Рисунок 4.2 – Схема алгоритму роботи програми

Блок 3. Перевірка структури рядків. Виконується валідація кожного рядка CLDATA на наявність обов’язкових полів і перевірка коректності форматів чисел. Некоректні або неповні рядки позначаються та обробляються за правилами fallback з метою гарантувати, що на подальші етапи потрапляють лише коректні та повні записи.

Блок 4. Нормалізація вектора інструмента. Для кожного рядка з орієнтаційним вектором виконується обчислення довжини вектора та ділення компонент на його норму для отримання одиничного вектора напрямку. Перед нормалізацією проводиться перевірка на нульову довжину і застосовується захисна логіка при аномальних значеннях. Результатом є стабільне одиничне представлення напрямку інструмента для подальших перетворень.

Блок 5. Обчислення кутів A і C. Нормалізований вектор перетворюється у кути обертання відповідно до кінематичної моделі верстата: застосовуються функції atan2 для обчислення кутів A та C з урахуванням чотириквадрантної неоднозначності. На цьому етапі також проводиться корекція значень у допустимі діапазони, усунення стрибків через переповнення кутів і підготовка числового формату, сумісного з контролером SINUMERIK.

Блок 6. Генерація шапки і хвоста NC-коду. Створюються службові блоки програми: заголовок з інформацією про інструмент, систему координат та початкові установки; блоки безпечного підходу; а також завершаючі блоки з командами вимкнення шпинделя, повернення в початкову позицію і маркуванням кінця програми. Шапка і хвіст формуються у відповідності до синтаксису контролера і забезпечують безпечний запуск та завершення обробки.

Блок 7. Розбір команд (GOTO, FEDRAT, CUTCOM і т.д.). Кожний підготовлений рядок інтерпретується відповідним парсером – виділяються координати, вектори, значення подачі (FEDRAT), режими корекції (CUTCOM), команди інструменту та інші службові записи. Для кожного типу команди встановлюється внутрішній обробник, що формує уніфікований проміжний опис дії для генерації NC-коду.

Блок 8. Генерація NC-коду на основі розібраних команд. На основі проміжного опису формуються робочі рядки NC-коду: обчислені координати та кути вставляються у команди переміщень (G1/G0), встановлюються модальні команди, регулюються значення подачі та шпинделя, реалізується логіка вмикання/вимикання CUTCOM та модальна оптимізація (усунення дублювань). Формуються блоки у форматі, сумісному з SINUMERIK, з урахуванням обмежень на довжину рядка та модальності.

Блок 9. Запис NC-коду у вихідний файл. Після побудови послідовності рядків виконується остаточна верифікація, така як перевірка синтаксису, відсутність невизначених змінних, контроль меж осей, після чого послідовність записується у вихідний NC-файл. Паралельно генерується лог-файл з інформацією про помилки, попередження та статистику вихідного файлу. Результат – готовий до симуляції та завантаження на верстат NC-файл.

Блок 10. Кінець виконання алгоритму.

Ця схема забезпечує однозначну послідовність дій для всіх можливих типів команд та дозволяє масштабувати систему, додаючи нові типи CLDATA-записів за потреби.

4.3 Опис особливостей програмної реалізації

Програмна реалізація постпроцесора містить низку спеціалізованих механізмів, які забезпечують коректну обробку траєкторних даних, трансформацію координат та формування NC-коду відповідно до вимог системи SINUMERIK. Для кожного етапу використано оптимізовані алгоритми, що гарантують точність обчислень та стабільність роботи.

Однією з ключових особливостей реалізації є система детального аналізу CLDATA. Кожен рядок проходить попередню фільтрацію, під час якої видаляються зайві пробіли, службові символи та контрольні знаки. Далі виконується визначення типу запису за ключовими словами, після чого

відповідні обробники здійснюють виділення значень координат X, Y, Z та компонентів вектора напряму I, J, K. Оскільки SINUMERIK очікує кути нахилу інструмента, а вхідні дані містять орієнтацію у вигляді вектора, необхідно виконати нормалізацію та подальші кутові перетворення. Орієнтаційний вектор *v* нормалізується за формулою (4.1):

$$\|v\| = \sqrt{i^2 + j^2 + k^2}$$
$$\hat{v} = \left(\frac{i}{\|v\|}, \frac{j}{\|v\|}, \frac{k}{\|v\|}\right) \quad , \qquad (4.1)$$

де *v* – орієнтаційний вектор;

i, j, k – координати вектору з CLDATA.

Для приведення орієнтації інструмента до кінематики верстата використано математичну модель перетворення тривимірного вектора у два обертальні ступені свободи. Кути A та C обчислюються на основі нормалізованого вектора за формулою (4.2):

$$A = -\arctan 2(i_n,\ k_n)$$
$$C = \arctan 2(j_n,\ \sqrt{i_n^2 + k_n^2}) \quad , \qquad (4.2)$$

де *A* і *C* – кути повороту осей стола;

i_n, j_n, k_n – координати вектору з CLDATA у рядку n.

Такі формули враховують особливості чотириквадрантної функції atan2, що забезпечує однозначність результатів навіть при нульових компонентах вектора.

Програмний застосунок містить низку функцій, а саме.

Функція select_cldata() використовується для вибору вхідного CLDATA-файлу. Вона відкриває системний діалог вибору файлу, після чого шлях до

вибраного документа автоматично вводиться у відповідне поле інтерфейсу. Забезпечує коректне отримання шляху та попереджає введення некоректних даних.

Функція select_output() відповідає за вибір імені та каталогу для збереження вихідного NC-коду. Користувач обирає місце збереження через стандартний діалог, після чого шлях записується у поле виводу. Гарантує, що файл буде створено у доступній директорії з правильним розширенням.

Функція run_conversion() реалізує повний цикл формування керуючої програми, а саме:

– перевіряє наявність і коректність шляху до файлу;
– читає вхідний CLDATA-файл построково;
– передає вміст у функцію обробки convert_cldata_to_gcode();
– записує згенерований NC-код у вибраний користувачем файл;
– формує повідомлення про успіх або повідомлення про помилку.

Функція parse_raw_goto(s) призначена для розбору CLDATA-записів типу GOTO, а саме:

– знаходить у рядку координати X, Y, Z і компоненти орієнтаційного вектора I, J, K;
– перевіряє кількість параметрів;
– повертає список числових значень або None, якщо дані некоректні;
– забезпечує коректне трактування геометрії та виконує первинну фільтрацію помилкових рядків.

Функція map_raw_to_axes(raw) формує структуру даних із сирих значень GOTO-запису. Результатом є словник із ключами X, Y, Z, I, J, K, що уніфікує формат подальшої обробки та робить програму стійкою до нестандартних варіацій вихідних рядків.

Функція normalize_vector(i, j, k) є критичною для правильного розрахунку кутів A і C, тому функція включає перевірку помилок і відхиляє некоректні дані. Вона відповідає за нормалізацію орієнтаційного вектора, а саме:

– обчислює довжину вектора;

– перевіряє, що довжина не дорівнює нулю;

– ділить кожну компоненту на довжину.

Функція vector_to_angles(i, j, k) виконує перетворення нормалізованого вектора у кути обертання A та C за кінематичною моделлю SINUMERIK. Функція використовує:

– тригонометричні функції atan2() для уникнення неоднозначностей;

– коректну роботу у всіх чотирьох квадрантах;

– контроль значень при нульових координатах.

Функція convert_cldata_to_gcode(lines) є найважливішою частиною реалізації і виконує основну трансформацію CLDATA у G-код. Вона забезпечує:

– послідовне визначення типу кожного рядка;

– розбір GOTO і службових записів;

– нормалізацію векторів та обчислення кутів A і C;

– формування робочих переміщень G1 із координатами та кутами;

– обробку та керування режимами CUTCOM (LEFT, RIGHT, OFF);

– встановлення подачі FEDRAT і обертів шпинделя;

– модальну оптимізацію (видалення дублювання команд);

– запис службових блоків, заголовка та завершальних команд;

– побудову цілісного NC-коду у вигляді упорядкованого списку рядків;

– перевірку коректності вхідних параметрів;

– автоматичне пропускання помилкових рядків;

– fallback-режими при нестандартних форматах GOTO;

– захист від аварій при неправильному форматі даних.

Також дуже важливим у програмі є список output, що є основним контейнером, у який поступово записуються всі рядки майбутньої керуючої

програми. Завдяки покроковій побудові та модальній оптимізації забезпечується структурований, читабельний та компактний NC-код.

Загальний опис функцій створеного програмного застосунку представлено у табл. 4.1.

Таблиця 4.1 – Опис функцій створеного програмного застосунку

Фунція	Короткий опис
select_cldata()	Відкриває системний діалог вибору файлу та отримує шлях до CLDATA.
select_output()	Дозволяє вибрати каталог і назву вихідного NC-файлу через стандартний діалог.
run_conversion()	Реалізує повний цикл формування NC-коду: перевіряє шляхи, читає CLDATA, записує вихідний файл, формує повідомлення про успіх або помилки.
parse_ raw_goto(s)	Розбирає GOTO-записи CLDATA: виділяє X, Y, Z, I, J, K; перевіряє кількість параметрів; повертає список чисел або None.
map_raw _to_axes(raw)	Створює уніфіковану структуру даних (словник X, Y, Z, I, J, K) із сирих значень GOTO, забезпечуючи стабільність подальшої обробки незалежно від формату запису.
normalize _vector(i, j, k)	Нормалізує орієнтаційний вектор: обчислює довжину, перевіряє ненульове значення, ділить компоненти на довжину. Містить захист від помилок і відхиляє некоректні дані.
vector_to _angles(i, j, k)	Перетворює нормалізований вектор у кути A і C. Використовує atan2() для уникнення неоднозначностей, працює у всіх квадрантах і враховує нульові значення компонент.
convert_cldata _to_gcode(lines)	Основна функція трансформації CLDATA в NC-код. Розпізнає тип рядка, обробляє GOTO, нормалізує вектори, обчислює A і C, формує G1-переміщення, керує CUTCOM, FEDRAT, SPINDL, виконує модальну оптимізацію, створює службові блоки, заголовок і завершальні команди. Містить перевірки, fallback-логіку та фільтрацію помилок.

4.4 Висновки

У цьому розділі представлено структуру, алгоритм роботи та основні особливості реалізації постпроцесора. Послідовний розподіл функцій між окремими логічними модулями, наявність чіткої алгоритмічної схеми та продумана система обробки команд забезпечують стабільність роботи та відповідність вимогам CNC-системи SINUMERIK. Підготовлений програмний інструмент готовий до тестування та практичного впровадження.

5 ТЕСТУВАННЯ ТА ПРАКТИЧНЕ ВПРОВАДЖЕННЯ

5.1 Використання розробленого постпроцессора

Розроблений постпроцесор призначений для формування управляючих програм у форматі, сумісному з системою Siemens SINUMERIK, на основі CLDATA-файлів, що генеруються CAM-системою. Практичне використання починається з підготовки вхідних даних, тобто формування траєкторії обробки у NX або іншій CAM-системі, та експорту траєкторних записів у форматі CLDATA. Для тестування практичного використання симулюємо ланцюг цифрового виробництва. Візьмемо готову модель, представлену на рис. 5.1.

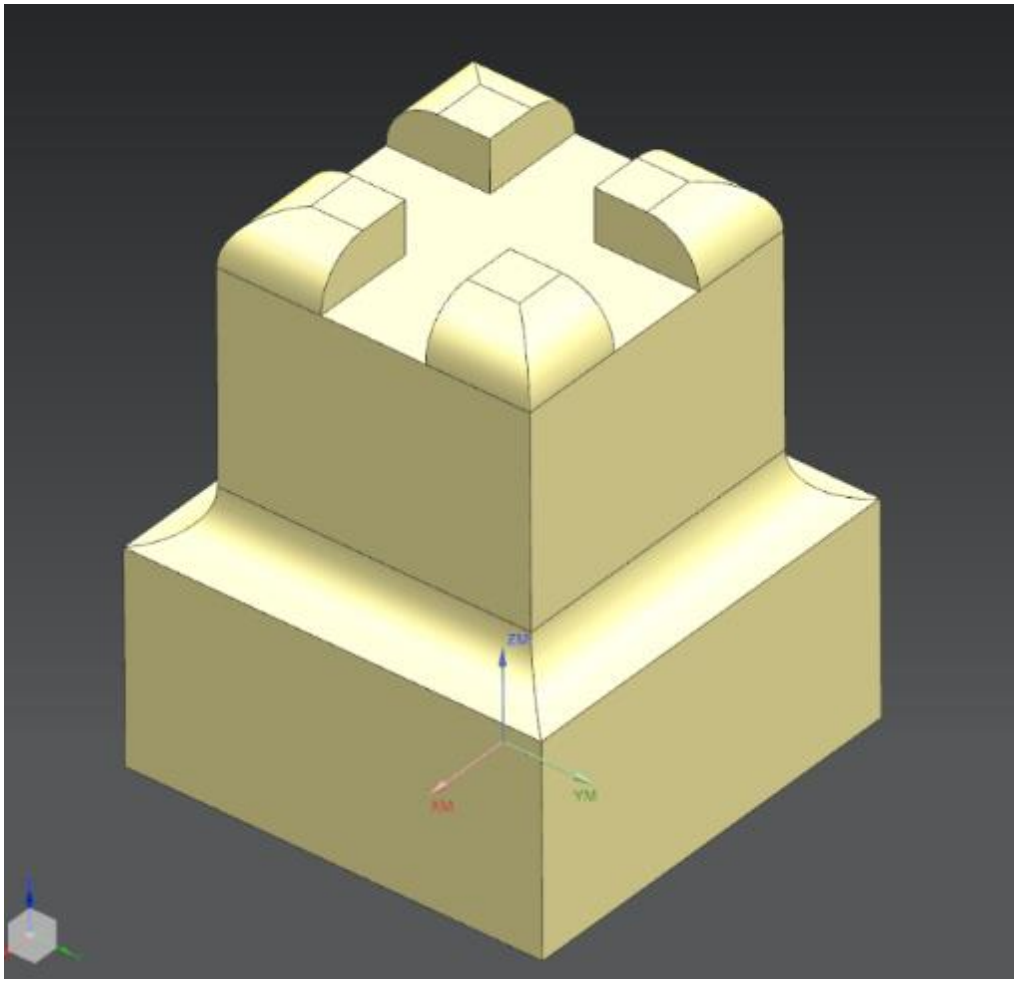

Рисунок 5.1 – Готова модель

Після цього створимо 3 найбільш розповсюджені траєкторії обробки. Першою буде загальна чорнова обробка, представлена на рис. 5.2.

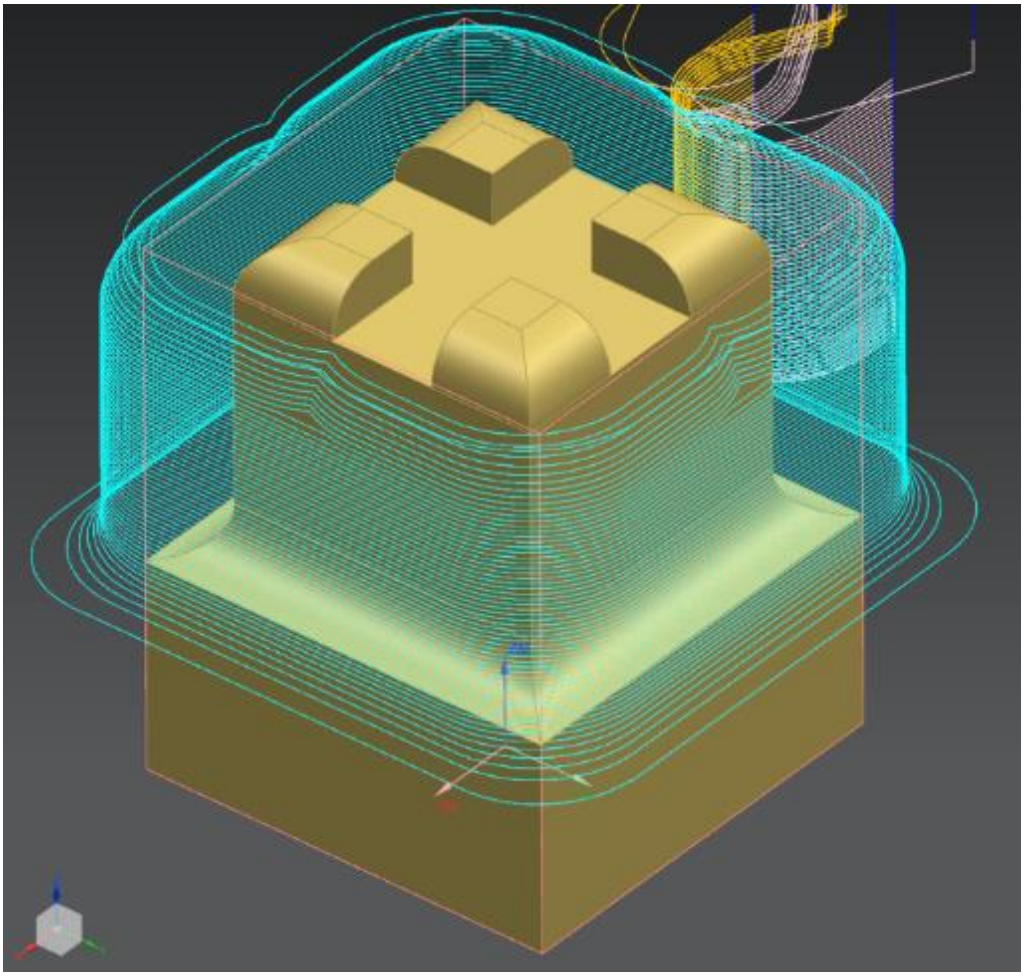

Рисунок 5.2 – Траєкторія чорнової обробки

Далі буде чистова обробка плоского контура з корекцією на діаметр інструмента, представлена на рис. 5.3. Наступна – закатка радіусів зі змінною відстанню по вісі Z, представлена на рис 5.4

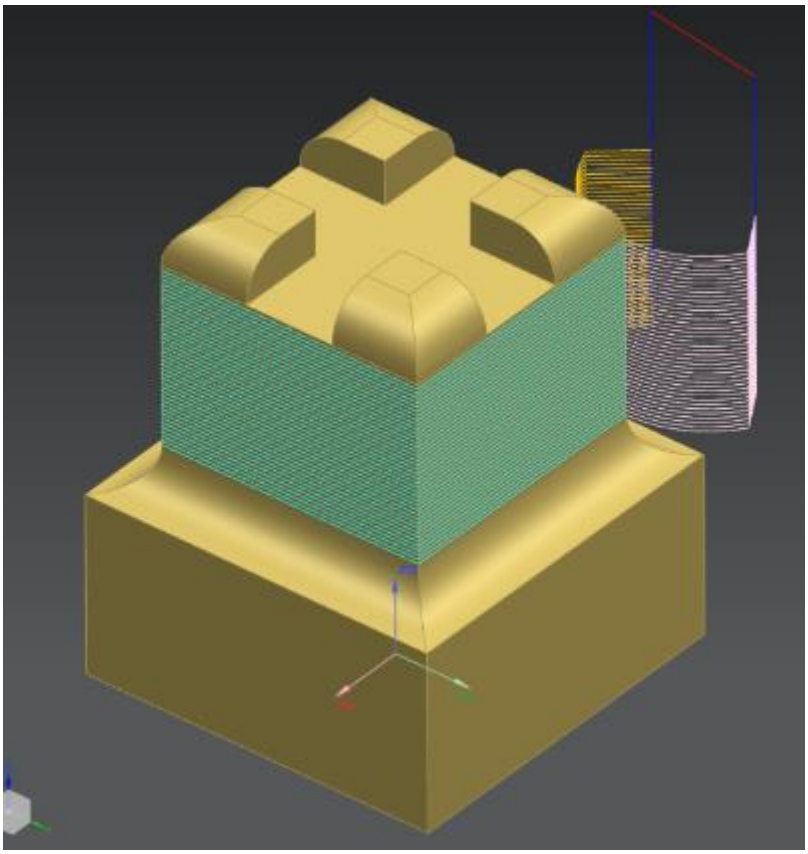

Рисунок 5.3 – Траєкторія чистової обробки з корекцією на діаметр

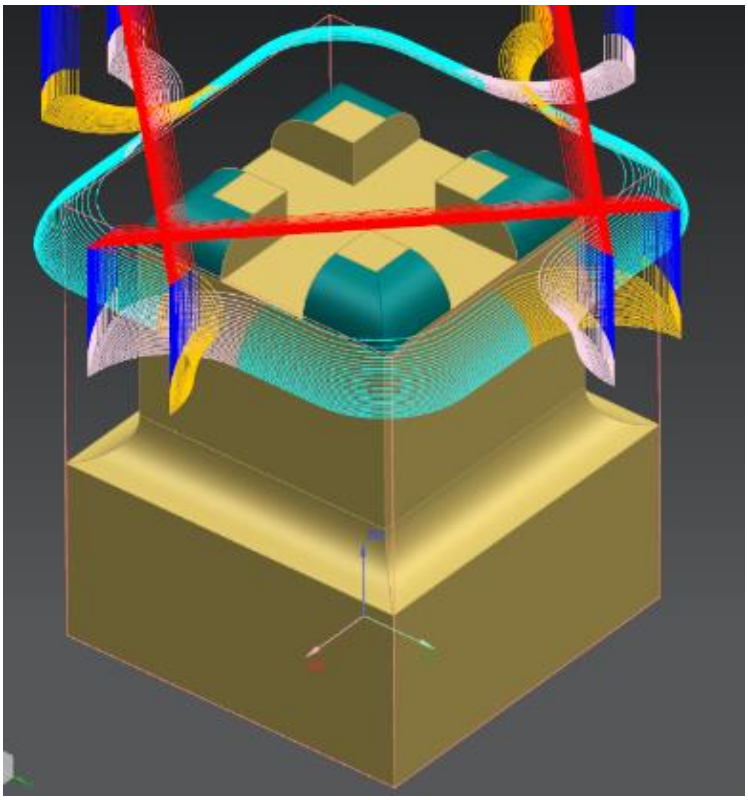

Рисунок 5.4 – Траєкторія закатки радіусів зі змінною відстанню по вісі Z

Також, для перевірки, створимо траєкторію зі змінним кутом інструмента, представлену на рис. 5.5.

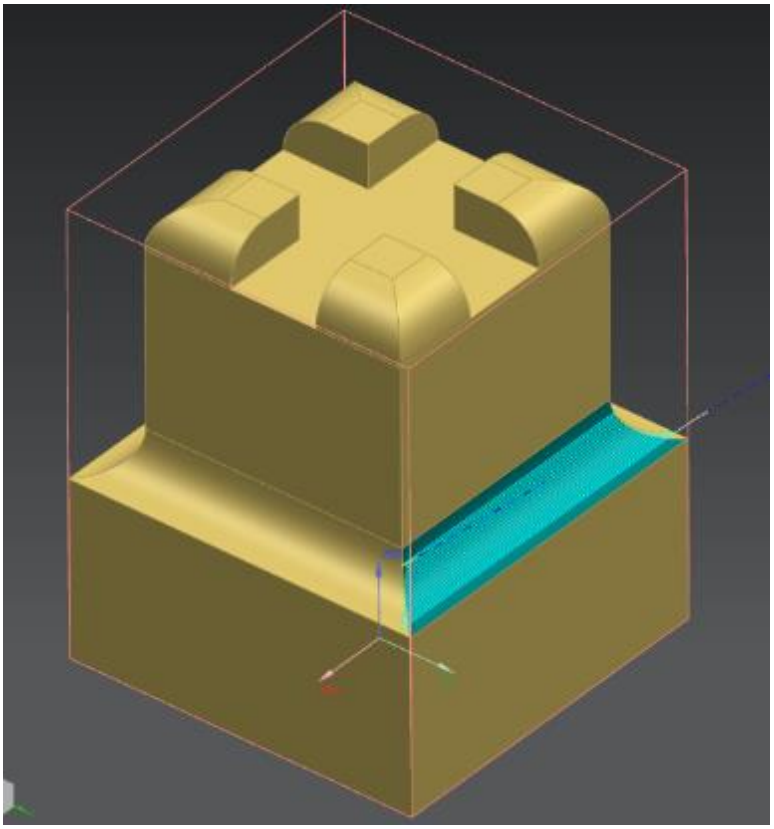

Рисунок 5.5 – Траєкторія зі змінним кутом інструмента

Після генерації всіх траєкторій, зберігаємо їх у .txt файл та запускаємо постпроцесор шляхом подвійного натискання на файл POST.exe. Відкривається вікно взаємодії з постпроцесором, представлене на рис. 5.6.

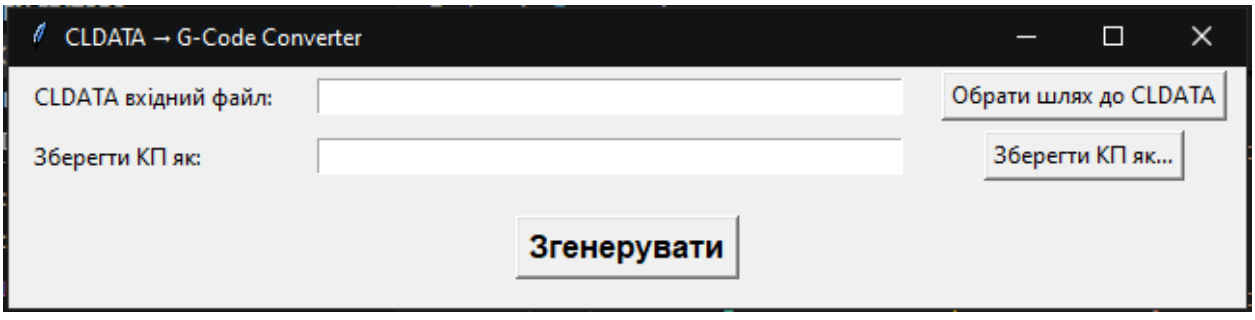

Рисунок 5.6 – Вікно взаємодії з постпроцесором

Якщо користувач спробує сгенерувати КП без вказання шляху до CLDATA, то отримає повідомлення про помилку (рис. 5.7).

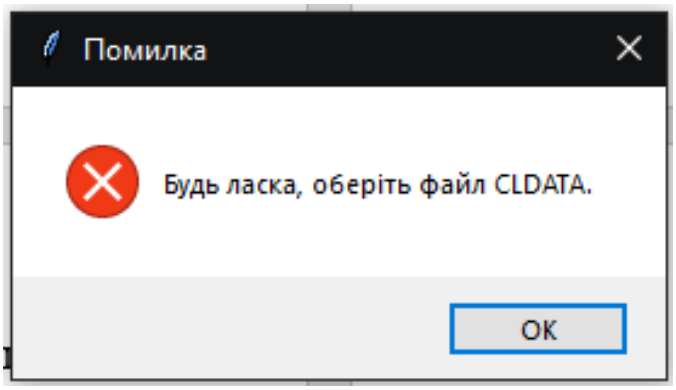

Рисунок 5.7 – Повідомлення про невказаний шлях до CLDATA

Якщо користувач спробує сгенерувати КП без вказання шляху для збереження керуючої програми, отримає повідомлення про помилку (рис. 5.8).

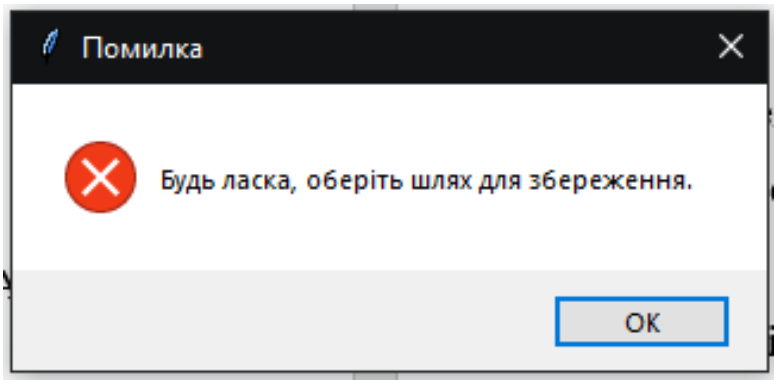

Рисунок 5.8 – Повідомлення про невказаний шлях для збереження КП

Після того як користувач обрав шлях до CLDATA та назвав вихідний файл, у який буде збережено КП, він може натиснути кнопку «Сгенерувати». Якщо генерація пройшла успішно, користувач отримає файл з КП та відповідне повідомлення (рис. 5.9).

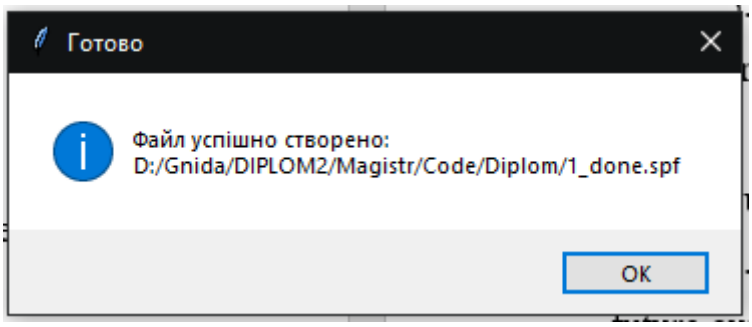

Рисунок 5.9 – Повідомлення про успішну генерацію керуючої програми

Варто зазначити, що у процесі використання програма не потребує додаткових налаштувань – усі необхідні параметри формуються автоматично на основі CLDATA. Загалом, завдяки оптимізації, обсяг вихідного файлу зменшується, що робить програму більш читабельною для оператора та спрощує налагодження на верстаті.

5.2 Симуляція та перевірка програми

Перевірка правильності роботи постпроцесора є критичною складовою впровадження. Цю перевірку можна виконати шляхом симуляції. Основна мета симуляції – переконатися, що траєкторія інструмента після постпроцесінгу залишається коректною, а всі режими обробки відповідають вимогам реального верстата. Для перевірки можна використовувати будь-яке середовище симуляції, для прикладу використано сайт https://gcodetutor.com/cnc-program-simulator.html. Симульовані траєкторії представлено на рис. 5.10.

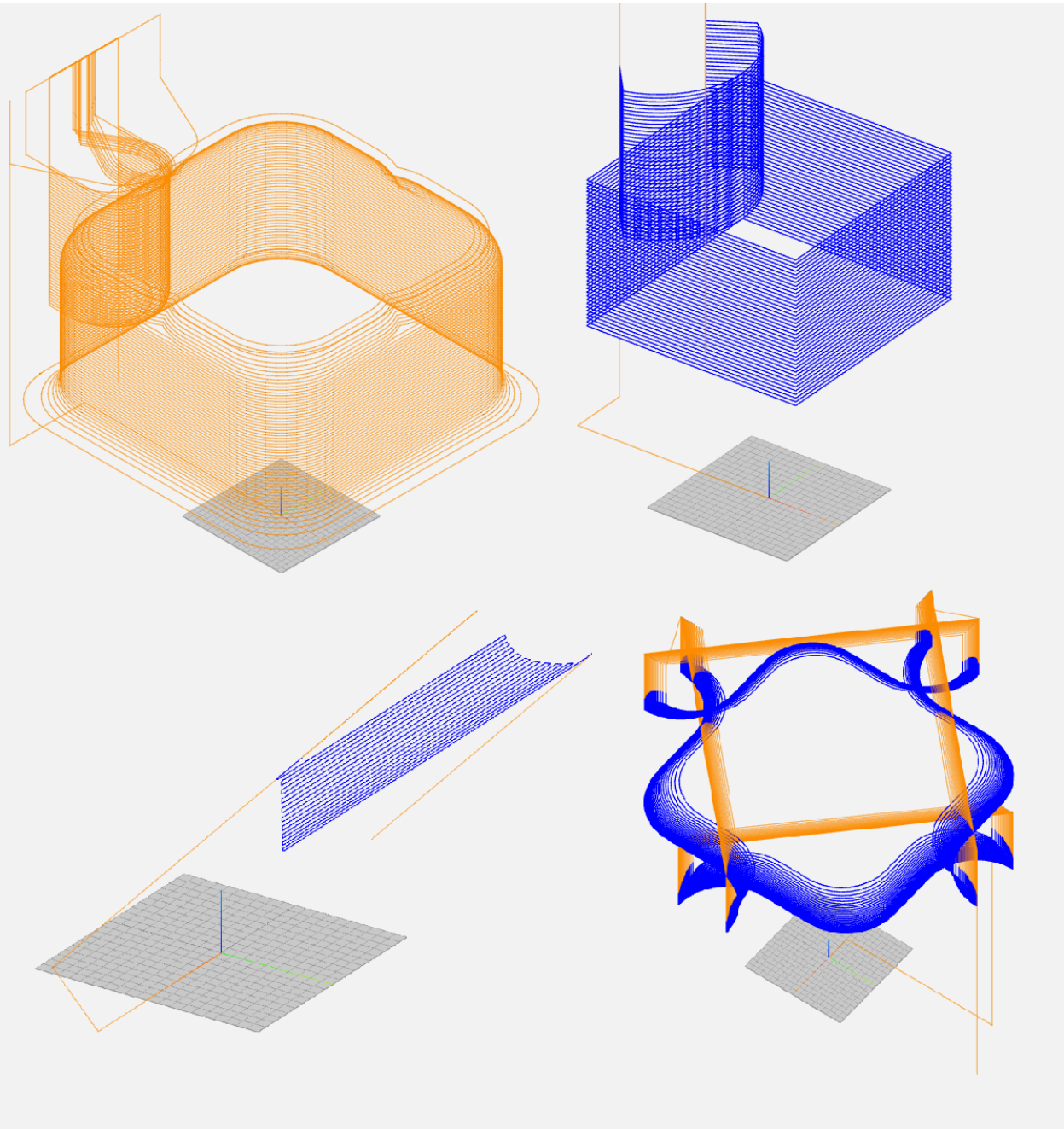

Рисунок 5.10 – Симуляція отриманих в результаті роботи застосунку КП

Особливу увагу під час перевірки необхідно приділити положенню системи координат, переходам між траєкторіями та методам врізання/виходу. Аналізуючи рис. 5.10, можна зробити висновок, що КП повністю відтворюють створені в CAM-застосунку траєкторії, що вказує на їх правильність.

5.3 Розробка рекомендацій щодо використання результатів

Для забезпечення надійного та безпечного використання результатів роботи постпроцесора необхідно дотримуватися цілісного комплексу рекомендацій, що охоплюють усі етапи – від підготовки траєкторії у CAM-системі до архівування кінцевого NC-коду. Можна виділити узагальнений перелік рекомендацій.

По-перше, перед експортом CLDATA необхідно перевірити правильність побудованої траєкторії в CAM-системі, звертаючи увагу на плавність переходів, відсутність розривів, неприродних змін орієнтаційного вектора та відповідність траєкторії кінематичним можливостям реального верстата. Рекомендовано виконувати візуальний контроль орієнтаційних векторів інструмента, особливо на складних поверхнях, оскільки різкі стрибки можуть спричинити некоректне обчислення кутів осей A та C під час постпроцесінгу.

По-друге, перед передачею CLDATA у постпроцесор слід переконатися, що файл містить повний набір параметрів: координати, орієнтаційні вектори, подачі, режими шпинделя, команди COROT/CUTCOM та інші службові записи, необхідні для коректного формування NC-коду. Також у випадку наявності пошкоджених або неповних рядків їх потрібно виправити або виключити, оскільки такі записи можуть призвести до некоректної траєкторії або неправильної роботи функцій корекції інструмента.

Після формування NC-коду бажано виконати покадрову симуляцію програми у будь-якому симуляційному застосунку, приділяючи особливу увагу рухам на ділянках із різкими змінами кута нахилу, оскільки тут найчастіше проявляються неточності у вихідних CLDATA.

По-третє, під час симуляції слід оцінити плавність зміни кутів A та C та переконатися у відсутності «стрибків» або небажаних обертів інструмента, що можуть виникати через помилки в орієнтаційних векторах або похибки у вихідних даних.

Також необхідно детально перевіряти моменти включення та виключення CUTCOM, оскільки корекція інструмента є одним із найчутливіших елементів траєкторії, здатним спричинити різкі зміщення у площині обробки за умови неправильного формування компенсаторної траєкторії. При обробці складних поверхонь бажано мінімізувати кількість переходів між режимами CUTCOM LEFT/RIGHT/OFF, оскільки часті зміни корекції ускладнюють траєкторію й підвищують ризик некоректного поведення інструмента.

Перед завантаженням NC-коду на верстат необхідно виконати генеральну перевірку перших блоків програми, переконуючись у правильності початкових установок, системи координат, висоти безпечного підходу та коректності встановленого інструмента.

Також рекомендується для кожної деталі або проєкту документувати параметри обробки: тип інструмента, режим корекції, подачі, швидкості, кути нахилу, особливості переходів та зауваження, виявлені під час симуляції. Це забезпечує технологічну спадковість і полегшує подальші доопрацювання. Це дозволить у разі необхідності повторного виготовлення тієї ж деталі повністю відтворити процес постпроцесінгу, зберігши при цьому технологічні параметри та уніфікований підхід до формування NC-коду.

5.4 Висновки

У цьому розділі описано процес використання постпроцесора, наведено результати симуляційної перевірки та сформовано рекомендації щодо його практичного застосування. Тестування підтвердило коректність роботи алгоритмів, точність геометричних обчислень і відповідність вимогам системи SINUMERIK.

ВИСНОВКИ

У дипломній роботі проведено аналіз особливостей сучасного цифрового виробництва, існуючих методів генерації керуючих програм та вимог до постпроцесорів для системи Siemens SINUMERIK. Досліджено структуру даних CLDATA, архітектуру постпроцесора та методи обробки траєкторій для багатокоординатної обробки. На основі аналізу сформовано вимоги до програмного забезпечення та розроблено архітектурне рішення, що забезпечує коректне перетворення даних щодо траєкторії у NC-код.

Розроблено постпроцесор, який підтримує зчитування вихідних CLDATA-записів, виконує перетворення координат з урахуванням кінематики верстата, обчислює кути осей A та C, формує службові блоки програми та забезпечує коректну обробку режимів CUTCOM. Виконано симуляцію обробки та перевірку сформованого NC-коду у середовищах верифікації, що підтвердило працездатність та правильність реалізованого алгоритму.

Наукова новизна роботи полягає в тому, що запропонований підхід включає механізм нормалізації орієнтаційних векторів, оптимізацію модальних команд та перевірку вхідних даних, що забезпечує перетворення даних щодо траєкторії у коректний NC-код.

Практична цінність роботи полягає в тому, що розроблений адаптивний постпроцесор може бути застосований в умовах сучасного цифрового виробництва, а за потреби модифікований та розширений під інші типи верстатів та контролерів, що сприятиме підвищенню ефективності роботи обладнання з ЧПК.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. ISO programming and G-codes [Electronic resource]. – Access mode: https://cncofcourse.com/lesson/iso-programming-and-g-codes.

2. International Organization for Standardization. ISO 6983-1:2009. Numerical control of machines – Program format and definitions of address words. – Geneva: ISO, 2009. – 38 p.

3. Altintas, Y. Manufacturing automation: metal cutting mechanics, machine tool vibrations, and CNC design / Y. Altintas. – Cambridge: Cambridge University Press, 2012. – 382 p.

4. Rosso, R. S. U. The adoption of STEP-NC for the manufacture of asymmetric rotational components / R. S. U. Rosso Jr., S. T. Newman, S. Rahimifard // Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture. – 2004. – Vol. 218, Issue 11. – P. 1639–1644.

5. Cao, X. Digital Twin–oriented real-time cutting simulation for intelligent CNC machining / X. Cao, G. Zhao, W. Xiao // Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture. – 2020. – Vol. 236, Issue 1. – P. 1–10.

6. Kusiak, A. Predictive models in digital manufacturing: research, applications and future outlook / A. Kusiak // International Journal of Production Research. – 2023. – Vol. 61, Issue 17. – P. 6052-6062.

7. Research on intelligent decision method of computer-aided manufacturing numerical control parameters based on model-based definition and back propagation neural networks / [N. Wang, S. Zhang, N. Wang, et al.] // Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture. – 2022. – Vol. 237, Issue 10. – P. 1596-1607.

8. Gui, W. J. Research and application of CAD/CAM technology in NC machining of parts with complex surface / W. J. Gui, D. Wei // Applied Mechanics and Materials. – 2014. – Vol. 602–605. – P. 111–116.

9. Accurate prediction of machining feedrate and cycle time considering interpolator dynamics / R. Ward, B. Sencer, B. Jones, E. Ozturk // Electrical Engineering and Systems Science. Systems and Control. – 2021. – P. 1-37.

10. Digital Twin-driven smart manufacturing: Connotation, reference model, applications and research issues / [Y. Lu, C. Liu, K. I-K. Wang, et al.] // Robotics and Computer-Integrated Manufacturing. – 2020. – Vol. 61. – P. 1-14.

11. FANUC. Manual Guide *i* [Electronic resource]. – Access mode: https://www.fanucamerica.com/products/cnc/cnc-software/programming-simulation-software/manual-guide-i.

12. Siemens. SINUMERIK Fundamentals [Electronic resource]. – Access mode: https://itscnc.com/pub/media/documents/fadal_manuals/siemansmanuals/Fundamentals.pdf.

13. Lavernhe, S. Model for performance prediction in multi-axis machining / S. Lavernhe, C. Tournier, C. Lartigue // High Performance Cutting: 2nd International Conference, Vancouver, Canada, 12 – 13 June 2006: proceedings. – 2006. – P.1-9.

14. TNC 640. User's Manual. Klartext programming [Electronic resource]. – Access mode: https://content.heidenhain.de/doku/tnc_guide/pdf_files/TNC640/34059x-11/bhb/892903-29.pdf.

15. Marek, J. Geometric Accuracy, Volumetric Accuracy and Compensation of CNC Machine Tools / J. Marek, M. Holub, T. Marek, P. Blecha // Machine Tools - Design, Research, Application. – 2020. – P. 1-45.

16. The Development Trends of Computer Numerical Control (CNC) Machine Tool Technology / K.-C. Yao, D.-C. Chen, C.-H. Pan, C.-L. Lin // Mathematics. – 2024. – Vol. 12, Issue 13. – P. 1-33.

17. Optimization of Tool Path Planning on CNC Machine Performance in Time-Efficient Machining / A. Pajaziti, O. Tafilaj, A. Gjelaj, B. Berisha // Machines, Vol. 13, no 1. – 2025. – P. 1-20.

18. Siemens AG. SINUMERIK 840D/840Di/810D/FM-NC. Programming Guide [Electronic resource] / Siemens AG. – Access mode: https://support.industry.siemens.com/cs/attachments/109439500/PGZ_0400_en.pdf.

19. DI FA PMA APC Team. Virtual Commissioning & Simulation Application Examples for machine building [Electronic resource] / DI FA PMA APC Team. – Siemens, 2025. – Access mode: https://cache.industry.siemens.com/dl/files/165/109777165/att_1338411/v2/2025_08_VirtualCommissioning_Simulation_ApplicationExamples_Internet.pdf.

20. Fatriyana, M. CNC Program and Programming of CNC Machine / M. Fatriyana // Journal of Mechanical Science and Engineering. – 2020. – Vol. 7, Issue 2. – P. 19–23.

21. Zou, Q. Brief on tool path generation/optimization methods for multi-axis CNC machining / Q. Zou // Computer Science. – 2022. – P. 1-12.

22. Perkasa, L. Automated generation of CNC programs for manufacturing: a review / L. Perkasa // Journal of Mechanical Science and Engineering. – 2019. – Vol. 6, Issue 2. – P. 35-39.

23. Liao, J. Data model-based toolpath generation techniques for CNC milling machines / J. Liao, Z. Huang // Frontiers in Mechanical Engineering. – 2024. – Vol. 10. – P. 1-12.

24. Petrakov, Y. Technology for programming contour milling on a CNC machine / Y. Petrakov, V. Korenkov, A. Myhovych // Eastern-European Journal of Enterprise Technologies. – 2022. – Vol. 2, No. 1(116). – P. 55–61.

25. Rahulkrishnan, R. A. Advancements in CNC Machinery: A Comprehensive Review in Industry 4.0 / [R. A. Rahulkrishnan, S. Nijin, S. P. Nikhil, et al.] // Journal of Emerging Technologies and Innovative Research. – 2024. – Vol. 11, Issue 4. – P. 1397–1403.

26. VMT. CNC Machining Programming: Definition, Process, Types, and Tips [Electronic resource] / VMT. – 2025. – Access mode: https://www.machining-custom.com/blog/cnc-machining-programming-tips.html.

27. Katz, R. Prototype Design and Manufacturing Manual / R. Katz, M. Lewis. – Department of Mechanical Engineering, University of Victoria. – P. 1-64.

28. Zhu, X. R. Research on automatic programming methods of CNC machining parameters of gear / X. R. Zhu // Electrical, Automation and Mechanical Engineering: International Conference, Phuket, Thailand, 26-27 July 2015: proceedings. – 2015. – P. 113–115.

29. Alharbi, M. Automatic Code Generation Techniques: A Systematic Literature Review / M. Alharbi, M. Alshayeb //Automated Software Engineering. – 2025. – Vol. 33, Issue. 4. – P. 1-96.

30. Optimization of CNC Milling Machining Time through Variation of Machine Parameters and Toolpath Strategy in Various Cross-sectional Shape on Tool Steels and Die Steels Materials / [W. Sumbodo, K. Kriswanto, T. S. Allam et al.] // Machines. – 2025. – Vol. 13, Issue 1. – P. 1-20.

31. Jee, S. Adaptive fuzzy logic controller for feed drives of a CNC machine tool / S. Jee, Y. Koren // Mechatronics. – 2004. – Vol. 14, Issue 3. – P. 299–326.

32. Feng, S. C. Manufacturing Planning and Predictive Process Models Integration Using Software Agents / S. C. Feng, K. A. Stouffer, K. K. Jurrens // Advanced Engineering Informatics. – 2005. – Vol. 19, Issue 2. – P. 135-142.

33. Fang, X. Intelligent Numerical Control Programming System Based on Knowledge Graph / X. Fang, J. Su, D. Cheng // Machines. – 2024. – Vol. 12, Issue 12. – P. 1-28.

34. Meyerovich, L. Empirical analysis of programming language adoption / L. Meyerovich, A. Rabkin // Object oriented programming systems languages & applications: ACM SIGPLAN international conference, Indianapolis Indiana USA, 29 - 31 October 2013: proceedings. – 2013. – P. 1–18.

35. Alomari, Z. Comparative studies of six programming languages / Z. Alomari, O. El Halimi, K. Sivaprasad, C. Pandit // Computer Science. Programming Languages. – 2015. – P. 1-71.

36. Abdulkareem, S. A. Evaluating Python, C++, JavaScript and Java Programming Languages Based on Software Complexity Calculator (Halstead

Metrics) / S. A. Abdulkareem, A. J. Abboud // IOP Conference Series: Materials Science and Engineering. – 2021. – Vol. 1076, Issue 1. – P. 1-10.

37. Onyango, K. A. Comparative analysis on the evaluation of complexity of C, C++, Java, PHP and Python based on Halstead metrics / K. A. Onyango, G. W. Mariga // International Journal of Computer and Information Technology. – 2023. – Vol. 12, No. 1. – P. 8-19.

38. A comparative analysis of programming languages used in microservices / [H. Saad, A. Safa, A. Gamal et al.] // International Journal of Computational and Experimental Science and Engineering. – 2025. – Vol. 11, no 1. – P. 1203-1213.

39. Productivity, portability, performance: data-centric Python / [A. Ziogas, T. Schneider, T. Ben-Nun et al.] // IEEE Transactions on Parallel and Distributed Systems. 2025. – Vol. 36, no 5. – P. 804-820.

40. Bridging HPC communities through the Julia programming language / [V. Churavy, W. Godoy, C. Bauer et al.] // Computer Science. Distributed, Parallel, and Cluster Computing. – 2022. – P. 1-20.

41. Blumfield, A. A Generator for Creating Adaptive Post Processors / A. Blumfield, M. Shpitalni, E. Lenz // CIRP Annals. – 1992. – Vol. 41, Issue 1. – P. 527–530.

ДОДАТОК А

Текст програми

A.1 gui.py

```
import tkinter as tk
from tkinter import filedialog, messagebox
import os
import test   # импортируем твой файл (test.py)

def select_cldata():
    path = filedialog.askopenfilename(
        title="Обрати шлях до CLDATA",
        filetypes=[("CLDATA files", "*.cldata *.txt *.dat"), ("All files",
"*.*")]
    )
    if path:
        entry_in.delete(0, tk.END)
        entry_in.insert(0, path)

def select_output():
    path = filedialog.asksaveasfilename(
        title="Зберегти КП як...",
        defaultextension=".nc",
        filetypes=[("G-code", "*.nc"), ("Text files", "*.txt"), ("All files",
"*.*")]
    )
    if path:
        entry_out.delete(0, tk.END)
        entry_out.insert(0, path)

def run_conversion():
```

```
        infile = entry_in.get().strip()
        outfile = entry_out.get().strip()

        if not infile or not os.path.exists(infile):
            messagebox.showerror("Помилка", "Будь ласка, оберіть файл CLDATA.")
            return

        if not outfile:
            messagebox.showerror("Помилка", "Будь ласка, оберіть шлях для збереження.")
            return

        try:
            with open(infile, "r", encoding="utf-8", errors="ignore") as f:
                lines = f.readlines()

            result = test.convert_cldata_to_gcode(lines)

            with open(outfile, "w", encoding="utf-8") as f:
                f.write("\n".join(result))

            messagebox.showinfo("Готово", f"Файл успішно створено:\n{outfile}")

        except Exception as e:
            messagebox.showerror("Помилка виконання", str(e))
```

```
    # ------------------ GUI ------------------
    root = tk.Tk()
    root.title("CLDATA → G-Code Converter")

    tk.Label(root, text="CLDATA вхідний файл:").grid(row=0, column=0, padx=10, pady=5, sticky="w")
    entry_in = tk.Entry(root, width=50)
    entry_in.grid(row=0, column=1, padx=10)
    tk.Button(root, text="Обрати шлях до CLDATA", command=select_cldata).grid(row=0, column=2, padx=10)

    tk.Label(root, text="Зберегти КП як:").grid(row=1, column=0, padx=10, pady=5, sticky="w")
    entry_out = tk.Entry(root, width=50)
    entry_out.grid(row=1, column=1, padx=10)
    tk.Button(root, text="Зберегти КП як...", command=select_output).grid(row=1, column=2, padx=10)

    tk.Button(root, text="Згенерувати", font=("Arial", 12, "bold"), command=run_conversion).grid(
        row=2, column=0, columnspan=3, pady=15
    )

root.mainloop()
```

A.2 test.py

```
    #!/usr/bin/env python3
    import re
```

```
import argparse
from math import isclose, atan2, degrees, sqrt

# ---------- helpers ----------
def parse_raw_goto(s):
    parts = s.split('/', 1)
    if len(parts) < 2:
        return []
    vals = parts[1].split(',')
    out = []
    for v in vals:
        v = v.strip()
        if v == '':
            out.append(None)
            continue
        try:
            out.append(float(v))
        except:
            out.append(None)
    return out

def map_raw_to_axes(raw):
    out = {}
    if len(raw) > 0 and raw[0] is not None:
        out['X'] = raw[0]
    if len(raw) > 1 and raw[1] is not None:
        out['Z'] = raw[1]
    if len(raw) > 2 and raw[2] is not None:
        out['Y'] = – raw[2]
```

```
        # keep orientation components available as raw keys (do NOT overwrite axis names)
        if len(raw) > 3 and raw[3] is not None:
            out['OR_X'] = raw[3]
        if len(raw) > 4 and raw[4] is not None:
            out['OR_Y'] = raw[4]
        if len(raw) > 5 and raw[5] is not None:
            out['OR_Z'] = raw[5]
        return out

    def fmt_num(x):
        if x is None:
            return None
        s = ('{:.3f}'.format(x)).rstrip('0').rstrip('.')
        if s in ('-0','-0.0'):
            s = '0'
        return s

    def compact_token(coords, axes):
        parts = []
        for a in axes:
            if a in coords:
                parts.append(f'{a}{fmt_num(coords[a])}')
        return ''.join(parts)

    def orientation_vector_to_angles(vx, vy, vz):
        # guard for None
        if vx is None or vy is None or vz is None:
            return None, None
```

```
        # compute A
        rxy = sqrt(vx*vx + vy*vy)
        A = degrees(atan2(rxy, vz))
        # compute raw C
        # if both vx and vy nearly zero, azimuth is undefined -> return 0
        if abs(vx) < 1e-12 and abs(vy) < 1e-12:
            C = 0.0
        else:
            C = degrees(atan2(vy, vx)) – 90.0
            # normalize C to -180..180
            while C <= -180.0:
                C += 360.0
            while C > 180.0:
                C -= 360.0
            # prefer +180 instead of -180
            if isclose(C, -180.0, abs_tol=1e-9):
                C = 180.0
        # round tiny near-zero values to 0
        if abs(A) < 1e-9: A = 0.0
        if abs(C) < 1e-9: C = 0.0
        return A, C


    # ---------- main converter ----------
    def convert_cldata_to_gcode(lines):
        # FIRST PASS: collect safe_z, first spindle, first tool, first orientation candidate
        raw_z_vals = []
        first_spindle = None
        first_tool = None
```

```
        first_orient_candidate = None
        first_goto_raw = None
        invert_x = False
        invert_z = False
        swap_yz = False

        for ln in lines:
            s = ln.strip()
            us = s.upper()
            if us.startswith('GOTO/'):
                raw = parse_raw_goto(s)
                # capture first GOTO raw (to be used as safe approach + orientation if present)
                if first_goto_raw is None:
                    first_goto_raw = raw
                if len(raw) > 1 and raw[1] is not None:
                    raw_z_vals.append(raw[1])
                # if full orientation vector present, store it
                if first_orient_candidate is None and len(raw) >= 6:
                    a = raw[3]; b = raw[4]; c = raw[5]
                    if a is not None or b is not None or c is not None:
                        first_orient_candidate = (a, b, c)
            if us.startswith('SPINDL/RPM') and first_spindle is None:
                parts = s.split(',')
                try:
                    rpm = int(float(parts[1]))
                except:
                    rpm = None
                dflag = parts[2].strip().upper() if len(parts) > 2 else ''
```

```
            if rpm is not None:
                first_spindle = (rpm, dflag)
        if us.startswith('LOAD/TOOL') and first_tool is None:
            m = re.search(r'LOAD/TOOL\s*,\s*(\d+)', us)
            if not m:
                m = re.search(r'LOAD/TOOL/(\d+)', us)
            if m:
                first_tool = int(m.group(1))

    if first_goto_raw is not None and len(first_goto_raw) > 1 and first_goto_raw[1] is not None:
        safe_z = first_goto_raw[1]
    else:
        safe_z = max(raw_z_vals) if raw_z_vals else 0.0

    out = []
    n = 1
    def emit(line):
        nonlocal n
        out.append(f'N{n} {line}')
        n += 1

    # HEADER: Tool, M6 first
    tool = first_tool if first_tool is not None else 1
    emit(f'T{tool}D1')
    emit('M6')

    emit('G0G54G64G90G94')
    emit('FGROUP(X,Y,Z,A,C)')
```

```
        # SPINDLE (single emit)
        spindle_emitted = False
        if first_spindle is not None:
            rpm, dflag = first_spindle
            spindle_emitted = True
            if 'CLW' in dflag or 'CW' in dflag:
                emit(f'S{rpm}M3')
            else:
                emit(f'S{rpm}M4')

        emit('M27M29')

        # HEADER ORIENTATION: per request swap A/C mapping: use full orientation vector if available
        last_out = {'X':None,'Y':None,'Z':None,'A':None,'C':None}
        last_token = ''

        if first_orient_candidate is None and first_goto_raw is not None and len(first_goto_raw) >= 6:
            first_orient_candidate = (first_goto_raw[3], first_goto_raw[4], first_goto_raw[5])

        if first_orient_candidate is not None:
            vx, vy, vz = first_orient_candidate
            a_ang, c_ang = orientation_vector_to_angles(vx, vy, vz)

            if c_ang is not None and c_ang > 90.0:
                invert_x = True
                invert_z = True
```

```
            if a_ang is not None and a_ang >= 0.0 and a_ang < 90.0:
                swap_yz = True
                invert_z = True

            # normalize None -> 0.0
            a_ang = 0.0 if a_ang is None else a_ang
            c_ang = 0.0 if c_ang is None else c_ang
            emit(f'G0A{fmt_num(a_ang)}C={fmt_num(c_ang)}+R24')
            last_out['A'] = a_ang
            last_out['C'] = c_ang
            last_token = f'G0A{fmt_num(a_ang)}C={fmt_num(c_ang)}+R24'

        # apply Y/Z swap to safe_z itself (but DO NOT swap first GOTO coords!)
        if swap_yz:
            # safe_z originally stored the CLDATA Z, which after swap becomes Y
            # The new Z for safe approach must be previous Y of the first GOTO
            if first_goto_raw is not None and len(first_goto_raw) > 2:
                original_y = -first_goto_raw[2]  # raw -> Y = -raw[2]
                safe_z = original_y

        if invert_z:
            safe_z = -safe_z

        last_mode = 'G0'
        emit('M26M28')
```

```
            # STATE
            last_pos = {'X':None,'Y':None,'Z':None,'A':None,'C':None}
            last_feed = None
            last_feed_sent = None
            last_token = last_token
            safe_sequence_done = False
            coolant_on = False

            # CUTCOM state
            comp_state = 'OFF'
            comp_pending_on = False
            comp_pending_off = False

            # POST-CUTCOM-OFF special sequence handler
            post_cutoff_sequence = []  # list of tuples (axis, include_feed_bool)
            in_post_cutoff = False

            for raw in lines:
                line = raw.strip()
                if not line:
                    continue
                us = line.upper()

                # skip comments/service lines
                if us.startswith('(') or 'END-OF-PATH' in us or
us.startswith('TOOL PATH') or us.startswith('TLDATA') or
us.startswith('PAINT/'):
                    continue
```

```
# LOAD/TOOL mid-file
if us.startswith('LOAD/TOOL'):
    m = re.search(r'LOAD/TOOL\s*,\s*(\d+)', us)
    if not m:
        m = re.search(r'LOAD/TOOL/(\d+)', us)
    if m:
        tnum = int(m.group(1))
        if tnum != tool:
            emit(f'T{tnum}D1'); emit('M6'); tool = tnum
    continue

# SPINDL/RPM mid-file
if us.startswith('SPINDL/RPM'):
    if spindle_emitted:
        continue
    parts = line.split(',')
    try:
        rpm = int(float(parts[1]))
    except:
        rpm = None
    dflag = parts[2].strip().upper() if len(parts) > 2 else ''
    if rpm is not None:
        spindle_emitted = True
        if 'CLW' in dflag or 'CW' in dflag:
            emit(f'S{rpm}M3')
        else:
            emit(f'S{rpm}M4')
    continue
```

```
                # FEDRAT
                if us.startswith('FEDRAT'):
                    nums = re.findall(r'[-+]?[0-9]*\.?[0-9]+', line)
                    if nums:
                        last_feed = float(nums[-1])
                    continue

                # CUTCOM possibly combined with GOTO
                if us.startswith('CUTCOM/'):
                    parts = line.split(None,1)
                    cmd = parts[0].upper()
                    if 'LEFT' in cmd:
                        comp_state = 'LEFT'; comp_pending_on = True; comp_pending_off = False
                    elif 'RIGHT' in cmd:
                        comp_state = 'RIGHT'; comp_pending_on = True; comp_pending_off = False
                    else:
                        comp_state = 'OFF'; comp_pending_on = False; comp_pending_off = True
                    # if combined with GOTO, fall through to GOTO processing
                    if len(parts) > 1 and parts[1].strip().upper().startswith('GOTO/'):
                        line = parts[1].strip(); us = line.upper()
                    else:
                        continue

                # GOTO handling (including combined CUTCOM/... GOTO/...)
                if us.startswith('GOTO/'):
                    raw_vals = parse_raw_goto(line)
```

```
                coords = map_raw_to_axes(raw_vals)

                if swap_yz:
                    y_val = coords.get('Y')
                    z_val = coords.get('Z')
                    if y_val is not None or z_val is not None:
                        coords['Y'], coords['Z'] = z_val, y_val

                if invert_x and 'X' in coords and coords['X'] is not None:
                    coords['X'] = -coords['X']

                if invert_z and 'Z' in coords and coords['Z'] is not None:
                    coords['Z'] = -coords['Z']

                # process full orientation vector if present (raw_vals[3..5])
                if len(raw_vals) >= 6 and any(v is not None for v in
raw_vals[3:6]):
                    vx = raw_vals[3]; vy = raw_vals[4]; vz = raw_vals[5]
                    a_ang, c_ang = orientation_vector_to_angles(vx, vy, vz)
                    # normalize None -> 0.0
                    if a_ang is None: a_ang = 0.0
                    if c_ang is None: c_ang = 0.0

                    # emit orientation if changed
                    if last_out.get('A') != a_ang or last_out.get('C') != c_ang:
                        orient                                                =
f'G0A{fmt_num(a_ang)}C={fmt_num(c_ang)}+R24'
                        if orient != last_token:
                            emit(orient); last_token = orient
```

```
                    last_out['A'] = a_ang; last_out['C'] = c_ang
                last_mode = 'G0'

            # decide mode
            if 'Z' in coords and isclose(coords['Z'], safe_z, rel_tol=1e-6, abs_tol=1e-6):
                mode = 'G0'
            else:
                if last_pos['Z'] is not None and 'Z' in coords and coords['Z'] < last_pos['Z']:
                    mode = 'G1'
                elif last_feed is not None:
                    mode = 'G1'
                else:
                    mode = 'G0'

            # SAFE APPROACH: once, for the first safe_z G0 we emit X, Y, Z separate tokens then M8
            if mode == 'G0' and not safe_sequence_done and 'Z' in coords and isclose(coords['Z'], safe_z, rel_tol=1e-6, abs_tol=1e-6):
                seq = []
                for ax in ('X','Y','Z'):
                    if ax in coords and coords[ax] != last_out.get(ax):
                        seq.append({ax: coords[ax]})
                first = True
                for item in seq:
                    token = compact_token(item, ('X','Y','Z'))
                    if token == '': continue
                    if first:
```

```
            if last_mode == 'G0':
                emit(token)
            else:
                emit('G0' + token)
            first = False
            last_mode = 'G0'
        else:
            emit(token)
        for k,v in item.items():
            last_out[k] = v; last_pos[k] = v
        last_token = token
    if not coolant_on:
        emit('M8'); coolant_on = True
    safe_sequence_done = True
    continue

# Handle pending CUTCOM ON/OFF when X and Y present
has_xy = ('X' in coords) and ('Y' in coords)
if comp_pending_on and has_xy:
    # enable correction now with G41/G42
    code = 'G41' if comp_state == 'LEFT' else 'G42'
    xy = {'X': coords['X'], 'Y': coords['Y']}
    token = compact_token(xy, ('X','Y'))
    if mode != last_mode:
        emit(mode + code + token); last_mode = mode
    else:
        emit(code + token)
    last_token = code + token
    comp_pending_on = False
```

```
                    coords.pop('X', None); coords.pop('Y', None)
                    last_out['X'] = xy['X']; last_out['Y'] = xy['Y']
                    last_pos['X'] = xy['X']; last_pos['Y'] = xy['Y']
                elif comp_pending_off and has_xy:
                    # emit G40X...Y... and enter special post-cutoff sequence handler
                    xy = {'X': coords['X'], 'Y': coords['Y']}
                    token = compact_token(xy, ('X','Y'))
                    if mode != last_mode:
                        emit(mode + 'G40' + token); last_mode = mode
                    else:
                        emit('G40' + token)
                    last_token = 'G40' + token
                    comp_pending_off = False
                    coords.pop('X', None); coords.pop('Y', None)
                    last_out['X'] = xy['X']; last_out['Y'] = xy['Y']
                    last_pos['X'] = xy['X']; last_pos['Y'] = xy['Y']
                    # prepare expected post-cutoff sequence (axes and whether to include feed)
                    # per universal pattern: Z_small, Z_safe+feed, X,Y, Z_small, Z_cut+feed
                    post_cutoff_sequence = [('Z', False), ('Z', True), ('XY', False), ('Z', False), ('Z', True)]
                    in_post_cutoff = True
                    continue  # continue to next input lines which will be handled by post-cutoff logic

                # If we are in post-cutoff special handling, process those expected steps
```

```
                if 'in_post_cutoff' in locals() and in_post_cutoff:
                    if post_cutoff_sequence:
                        expected_axis, expect_feed = post_cutoff_sequence.pop(0)
                        # build token depending on expected_axis
                        if expected_axis == 'Z':
                            if 'Z' in coords:
                                # emit Z (no G0/G1 prefix, no mode forcing), include
feed only if expect_feed True
                                z_token = compact_token({'Z': coords['Z']}, ('Z',))
                                if expect_feed and last_feed is not None:
                                    z_token = z_token + f'F{fmt_num(last_feed)}'
                                # emit without mode prefix (match etalon)
                                emit(z_token)
                                last_token = z_token
                                last_out['Z'] = coords['Z']; last_pos['Z'] = coords['Z']
                                # remove Z from coords
                                coords.pop('Z', None)
                                # if this was last expected, and sequence empty => exit
post_cutoff
                                if not post_cutoff_sequence:
                                    in_post_cutoff = False
                                continue
                            else:
                                # if Z not present while expected, just skip (tolerate)
                                if not post_cutoff_sequence:
                                    in_post_cutoff = False
                                continue
                        elif expected_axis == 'XY':
                            # expect both X and Y in coords
```

```
                    if ('X' in coords) and ('Y' in coords):
                        xy_token = compact_token({'X': coords['X'], 'Y': coords['Y']}, ('X','Y'))
                        emit(xy_token)
                        last_token = xy_token
                        last_out['X'] = coords['X']; last_out['Y'] = coords['Y']
                        last_pos['X'] = coords['X']; last_pos['Y'] = coords['Y']
                        coords.pop('X', None); coords.pop('Y', None)
                        if not post_cutoff_sequence:
                            in_post_cutoff = False
                        continue
                    else:
                        # missing XY, skip
                        if not post_cutoff_sequence:
                            in_post_cutoff = False
                        continue
                else:
                    in_post_cutoff = False
                # if we reach here, fall through to normal processing for remaining coords

            # Normal movement emission: filter unchanged axes
            filtered = {}
            for ax in ('X','Y','Z'):
                if ax in coords and coords[ax] != last_out.get(ax):
                    filtered[ax] = coords[ax]

            coord_token = compact_token(filtered, ('X','Y','Z'))
            feed_token = ''
```

```
                if mode == 'G1' and last_feed is not None:
                    if last_feed != last_feed_sent or mode != last_mode:
                        feed_token = f'F{fmt_num(last_feed)}'; last_feed_sent =
last_feed

                token = (coord_token + feed_token) if (coord_token or
feed_token) else ''

                if token:
                    if mode != last_mode:
                        emit(mode + token); last_mode = mode
                    else:
                        # omit mode prefix; avoid duplicates
                        if token != last_token:
                            emit(token)
                    last_token = token
                    for ax in filtered:
                        last_out[ax] = filtered[ax]

                # update last_pos for provided coords
                for ax in ('X','Y','Z','A','C'):
                    if ax in coords:
                        last_pos[ax] = coords[ax]

                continue

            # ignore other lines
            continue
```

```
            # TAIL: if safe Z already at last_out['Z'] and last_feed_sent corresponds, skip emitting duplicate safe-Z.
            tail_feed = last_feed if last_feed is not None else 3000.0
            safe_z_already_emitted = (last_out.get('Z') is not None and isclose(last_out.get('Z'), safe_z, rel_tol=1e-6, abs_tol=1e-6))
            # Additionally ensure we didn't already emit the Z with same feed at end of post-cutoff
            if not safe_z_already_emitted:
                emit(f'Z{fmt_num(safe_z)}F{fmt_num(tail_feed)}')
            # else: do not emit duplicate safe-Z line

            emit('G54')
            emit('G75Z0M5M9')
            emit('M17')

            return out

        # ---------- CLI ----------
        def main():
            ap = argparse.ArgumentParser(description='CLDATA -> G-code')
            ap.add_argument('--infile', required=True)
            ap.add_argument('--outfile', required=True)
            args = ap.parse_args()

            with open(args.infile, 'r', encoding='utf-8', errors='ignore') as f:
                lines = f.readlines()

            out_lines = convert_cldata_to_gcode(lines)
```

```
        with open(args.outfile, 'w', encoding='utf-8') as f:
            for ln in out_lines:
                f.write(ln + '\n')

        print(f'Wrote {len(out_lines)} lines to {args.outfile}')

    if __name__ == '__main__':
main()
```

ДОДАТОК Б

Слайди презентації

Рисунок Б.1 – Слайд 1

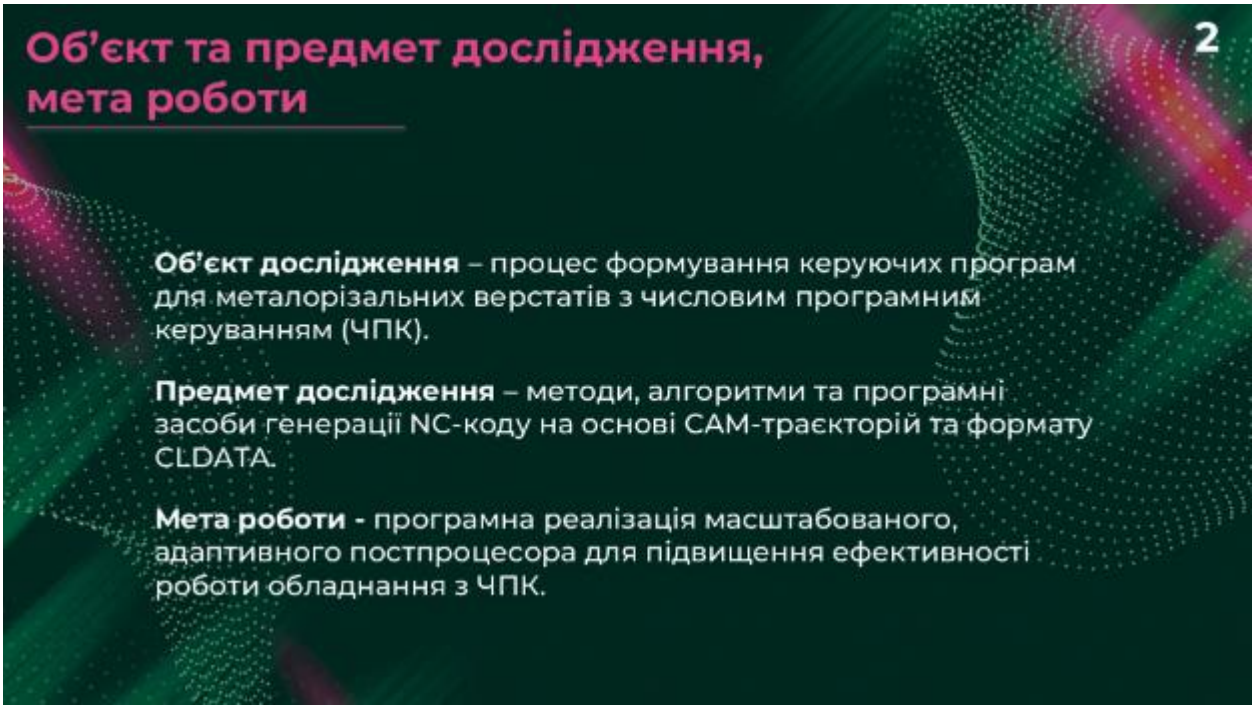

Рисунок Б.2 – Слайд 2

Рисунок Б.3 – Слайд 3

Рисунок Б.4 – Слайд 4

Рисунок Б.5 – Слайд 5

Рисунок Б.6 – Слайд 6

Рисунок Б.7 – Слайд 7

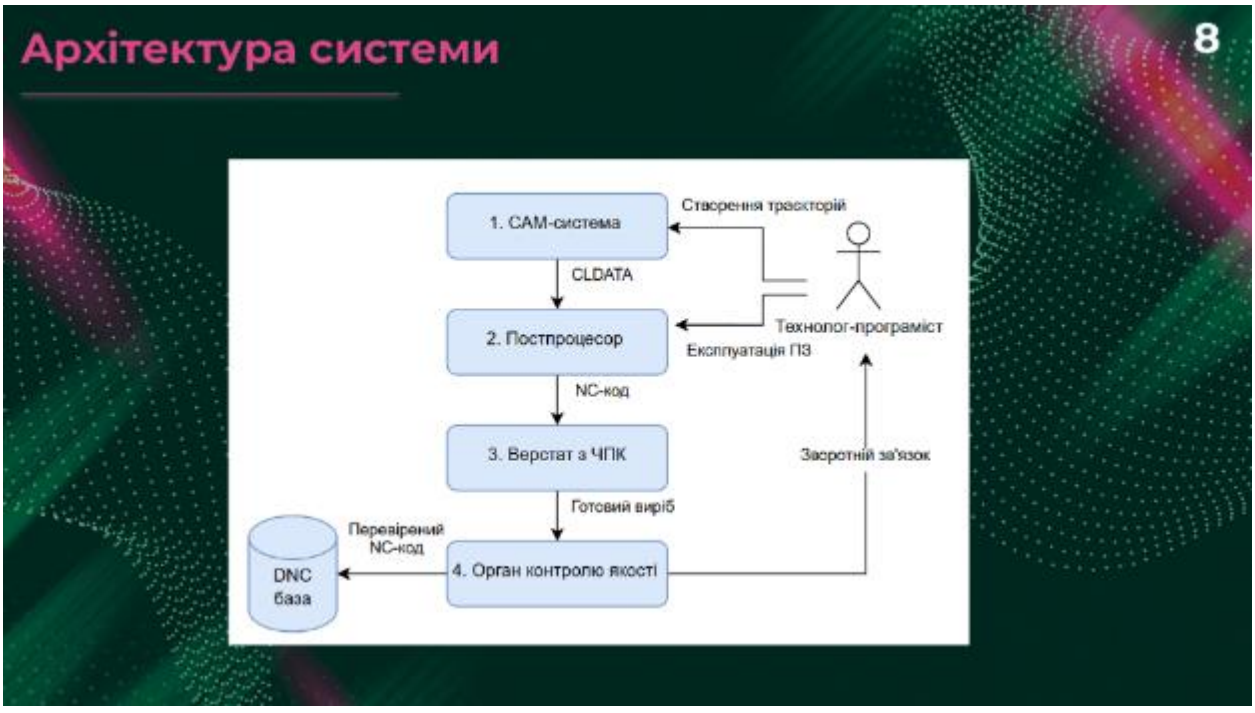

Рисунок Б.8 – Слайд 8

Рисунок Б.9 – Слайд 9

Рисунок Б.10 – Слайд 10

Рисунок Б.11 – Слайд 11

Рисунок Б.12 – Слайд 12

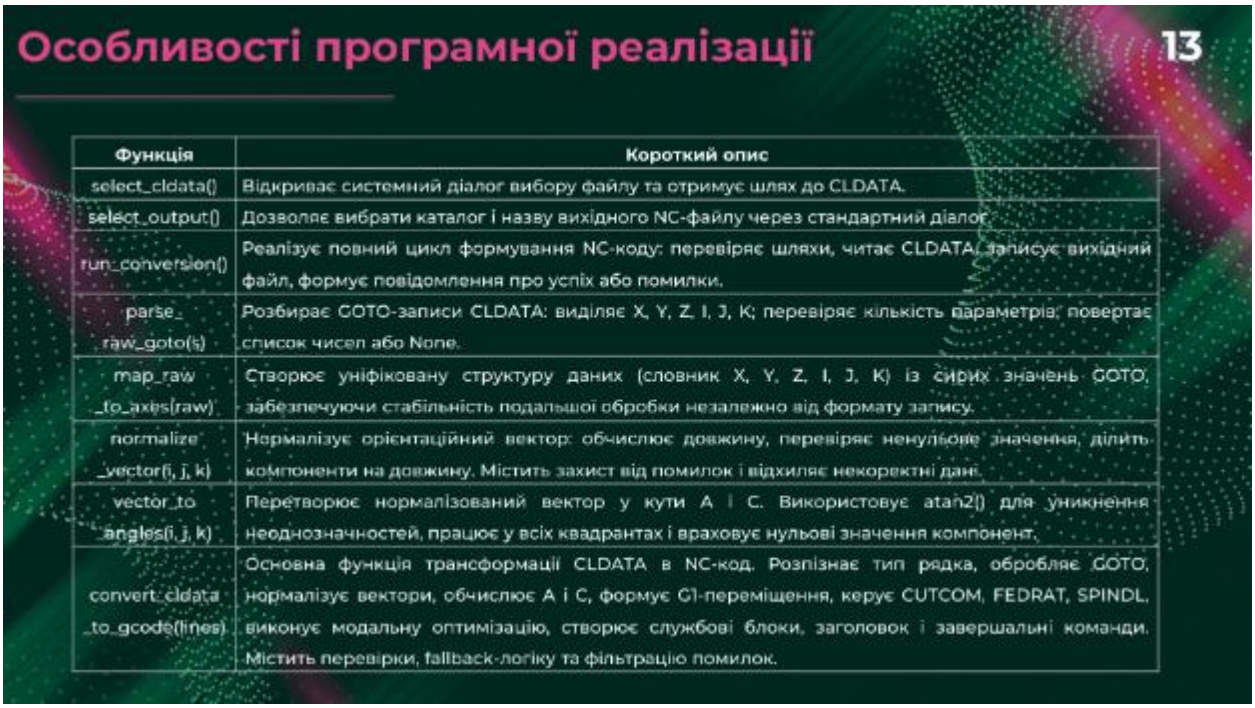

Рисунок Б.13 – Слайд 13

Рисунок Б.14 – Слайд 14

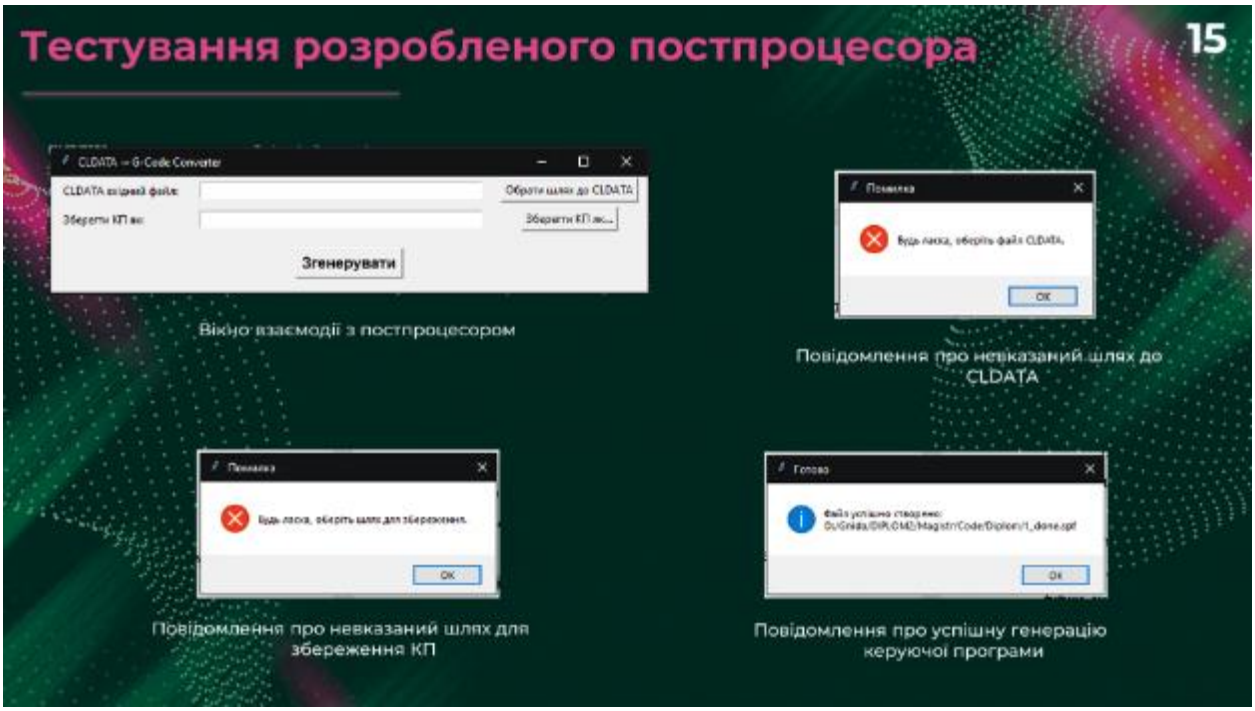

Рисунок Б.15 – Слайд 15

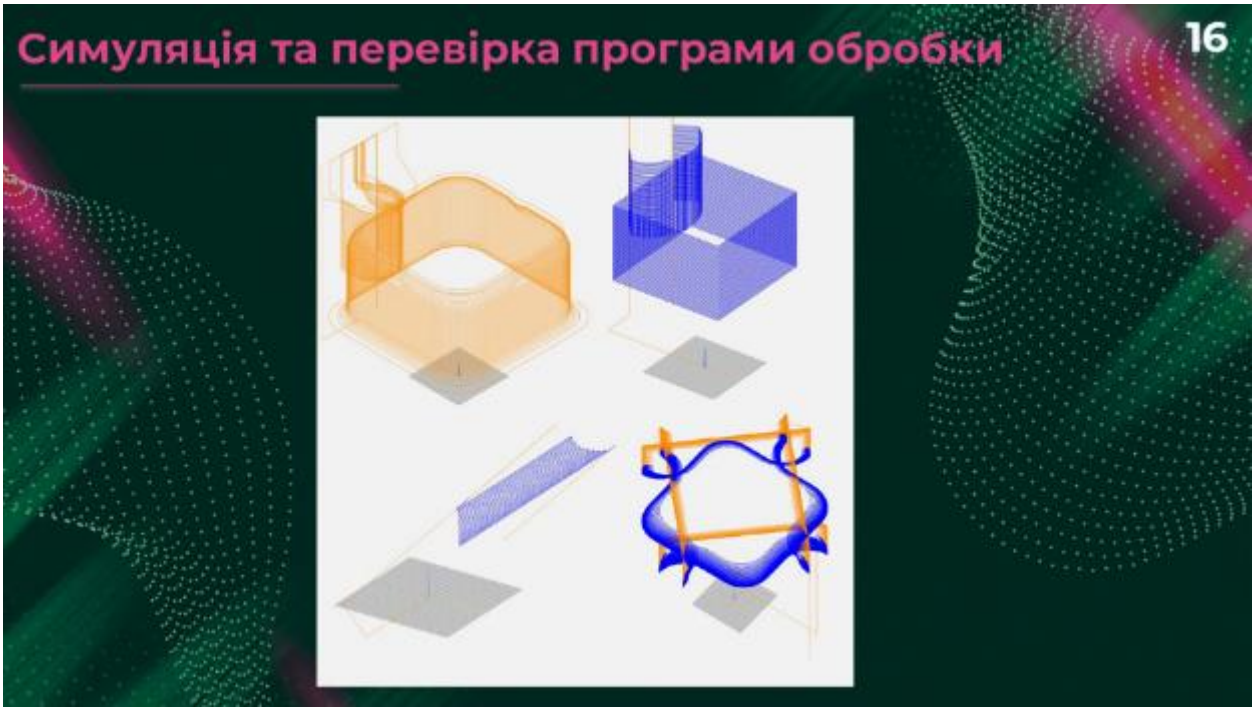

Рисунок Б.16 – Слайд 16

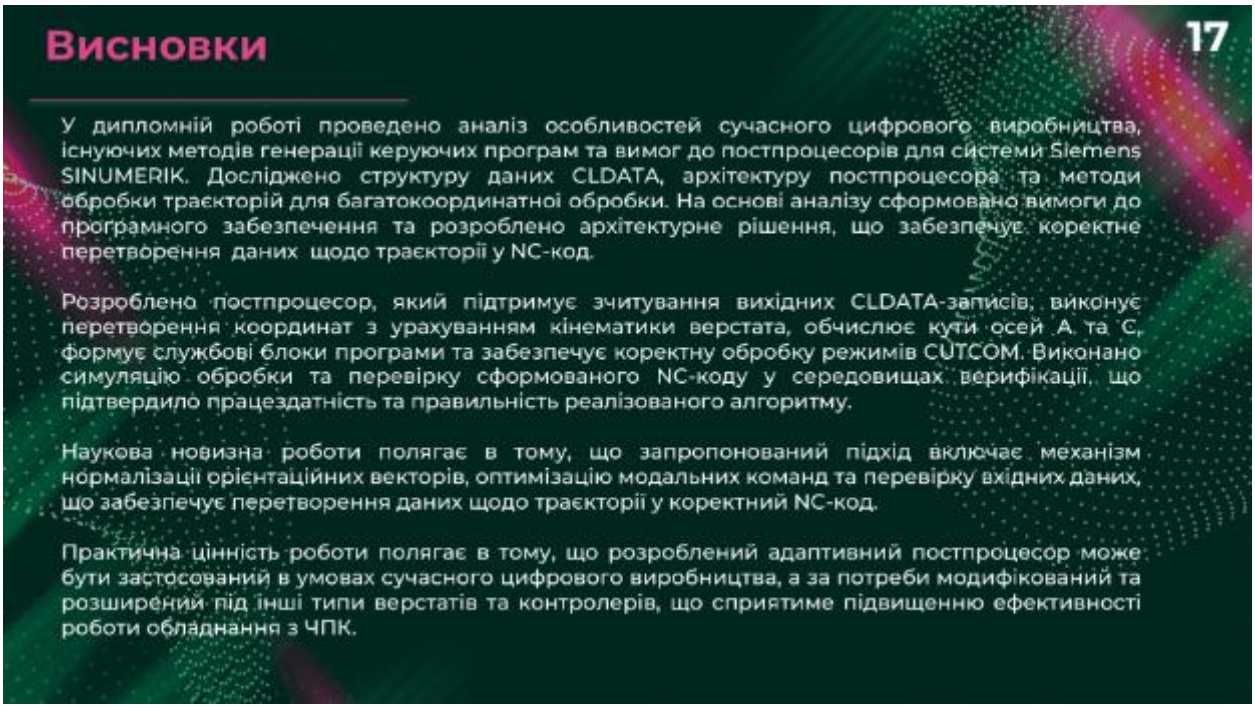

Рисунок Б.17 – Слайд 17