Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)
Навчально-науковий інститут комп'ютерних наук
та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))
Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Магістерська кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему: «Спеціалізоване програмне забезпечення для управління апаратними компонентами 3d принтера»

Виконав: студент VI курсу групи КН-62м
Спеціальності:
122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Бучко Р.О.
(прізвище та ініціали)

Керівник Думанський О.І.
(прізвище та ініціали)

Рецензент Карашецький В. К.
(прізвище та ініціали)

Львів – 2025

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра комп'ютерних наук
Рівень вищої освіти другий (магістерський)
Спеціальність 122 "Комп'ютерні науки"
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерних наук

_____ Борецька І.Б.

"10" грудня 2025 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Бучко Ростислав Орестович
(прізвище, ім'я, по батькові)

1. Тема роботи: Спеціалізоване програмне забезпечення для управління апаратними компонентами 3d принтера

Керівник роботи *доц. Думанський О.І.*,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "29" квітня 2025 року №С-288

2. Термін подання студентом роботи 10.12.2025

3. Вихідні дані до роботи: Створення спеціалізованого програмного забезпечення для управління апаратними компонентами 3d принтера

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Розділ 1. Стан проблемної області

Розділ 2. Інформаційне забезпечення

Розділ 3. Математичне забезпечення

Розділ 4. Програмне забезпечення

Розділ 5. Розроблення стартап-проєкту

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Підготовка матеріалу до доповіді

6. Дата видачі завдання 01.05.2025

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1.	Огляд літератури згідно досліджуваної теми. Збір необхідних матеріалів.	01.05.25-20.05.25	Виконано
2.	Постановка задачі і її формалізація	20.05.25-10.06.25	Виконано
3.	Виконання вхідного етапу технології	10.06.26-25.07.25	Виконано
4.	Реалізація головних алгоритмів проекту	25.07.25-12.09.25	Виконано
5.	Виконання етапу відлагодження проєкту	12.09.25-22.10.25	Виконано
6.	Виконання етапу впровадження та випуску бета-версії.	22.10.25-05.11.25	Виконано
7.	Оформлення записки до дипломного проекту.	05.11.25-10.12.25	Виконано

Студент _____ Бучко Р.О.
(підпис) (прізвище та ініціали)

Керівники роботи _____ Думанський О.І.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Магістерська робота складається із 72 сторінок пояснювальної записки, містить 28 ілюстрацій та 15 джерела літератури.

Дипломна робота присвячена розробці та створенню програмного забезпечення 3Д-друкарки , для трьох координатних осей із застосуванням алгоритмів згладжування для дуг і прямих . Для реалізації поставленої цілі за основу береться мова програмування «C/C++» із застосуванням мікроконтролера.

У роботі передбачено використання G-коду як основного інструменту керування 3D-друкаркою, що дозволяє здійснювати послідовне виконання команд для формування об'єкта відповідно до заданої моделі.

Ключові слова: CNC, G-CODE, мікропрограма, програмне забезпечення.

ABSTRACT

The master's thesis consists of 72 pages of explanatory text, includes 28 illustrations, and references 15 sources.

The thesis is devoted to the development and creation of 3D printer software for three coordinate axes using smoothing algorithms for arcs and lines. To achieve this goal, the programming language "C/C++" using a microcontroller is taken as a basis.

The work provides for the use of G-code as the main tool for controlling the 3D printer, which allows sequential execution of commands to form an object in accordance with a given model.

Keywords: CNC, G-CODE, firmware, software.

ТЕХНІЧНЕ ЗАВДАННЯ

Створити за допомогою мови програмування C/C++ основу для інтерпретції мови програмування верстатів із числовим програмним управлінням G-Code. Для створення та тестування програмного забезпечення використати віртуальний симулятор і екосистему Arduino. Програмне забезпечення має підтримувати основні команди G-Code, ввід команд здійснювати за допомогою послідовного порту.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

ПП — це послідовний порт, інтерфейс, через який дані передаються по одному біту за раз, послідовно. Це типовий спосіб зв'язку між контролером, як-от Arduino, та комп'ютером або іншим пристроєм.

ПЗ — це програмне забезпечення, набір комп'ютерних програм, процедур та документації, які виконують певні завдання.

ЧПК — це числове програмне керування. Це комплексна система, що поєднує програмне забезпечення та апаратну частину для автоматичного керування верстатами, такими як фрезерні машини або 3D-принтери. Програмна складова цієї системи відповідає за всі необхідні розрахунки, перетворення даних, масштабування значень та генерацію команд для виконавчих механізмів. Крім того, вона забезпечує взаємодію з іншими пристроями та надає користувацький інтерфейс.

УБ — універсальний кільцевий буфер. Як правило область пам’яті для тимчасового зберігання та накопичення даних. Кільцеві буфери переважно типу перший зайшов — перший вийшов (FIFO)

ВСТУП

У найзагальнішому вигляді, верстат із числовим програмним керуванням (ЧПК) є автоматизованою оброблювальною системою, функціонування якої забезпечується комп’ютерною програмою. Такий верстат керується цифровим кодом, що визначає послідовність дій, переміщення інструмента та взаємодію з оброблюваною заготівкою.

Сучасні системи ЧПК базуються на тісній інтеграції із CAD/CAM-технологіями, де CAD (Computer-Aided Design) — це комп’ютерне автоматизоване проектування, а CAM (Computer-Aided Manufacturing) — автоматизоване виготовлення виробів. Застосування таких систем забезпечує високу точність виготовлення, повторюваність продукції та скорочення термінів виробничого циклу.

Особливу актуальність у контексті цифрового виробництва набувають 3D-друкарки з числовим програмним керуванням, які дозволяють виготовляти складні об’ємні об'єкти методом пошарового наплавлення матеріалу. Управління таким обладнанням зазвичай здійснюється за допомогою G-коду — стандартної мови програмування для ЧПК-систем, яка описує координати переміщення, швидкість друку, параметри подачі тощо.

У рамках цієї роботи об'єктом дослідження є розробка програмного інтерпретатора G-коду для двокоординатної системи, з реалізацією алгоритмів згладжування траєкторій, зокрема для обробки лінійних сегментів із кутами, відмінними від прямих.

Метою роботи є створення дослідного зразка програмно-апаратного модуля, придатного для подальшої комерціалізації та використання на виробничих підприємствах або у приватному секторі.

Предмет дослідження охоплює питання налагодження ефективного процесу створення ЧПК-системи, яка здатна інтерпретувати команди G-коду та забезпечувати точне керування виконавчими механізмами.

Практичне значення полягає у впровадженні спрощених методів розробки доступних ЧПК-систем, орієнтованих на зниження вартості виробництва та пришвидшення впровадження нових технологій.

Наукова новизна полягає у створенні попередньо запрограмованого контролера, що забезпечує прискорення запуску 3D-друкарки в роботу за рахунок попередньої обробки та оптимізації команд G-коду.

Актуальність дослідження обумовлена низькою собівартістю реалізації системи та доступністю для широкого кола користувачів. Це створює потенціал для розробки та поширення модульних комплектів 3D-друкарок з ЧПК — як для аматорського застосування, так і для малого бізнесу чи дрібносерійного виробництва.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Огляд проблемної області

Машини з Числовим Програмним Управлінням (ЧПК), або CNC (Computer Numerical Control), є сучасним технологічним обладнанням, що використовує цифрові дані для автоматизованого керування рухом робочих органів та інструментів. Призначення ЧПК полягає в забезпеченні високої точності, повторюваності та ефективності виробничих процесів у різних галузях.

Цей клас обладнання охоплює широкий спектр машин. До ЧПК-машин належать лазерні різаки і різаки водним струменем високого тиску, що використовуються для прецизійного розкрою та гравірування матеріалів. Традиційніші токарні та фрезерні верстати, що обробляють деревину й метал шляхом зняття зайвого матеріалу, також широко використовують ЧПК. Крім того, дельта-роботи та механічні маніпулятори є роботизованими системами, які забезпечують швидке та точне позиціонування інструментів або об'єктів. Одним з найбільш інноваційних і швидкозростаючих сегментів ЧПК-технологій є 3D-принтери, які є яскравими представниками адитивного виробництва.[4]

Принцип функціонування ЧПК-машин полягає у перетворенні дизайнерських та інженерних рішень у послідовність точних команд, що зрозумілі для машини. Ці команди забезпечують рух робочих органів по заданих координатах за допомогою спеціальних приводів. Основними типами приводів є крокові двигуни, які забезпечують дискретне переміщення, та сервоприводи, що дозволяють більш точне керування з зворотним зв'язком. Сучасні системи також можуть використовувати лінійні двигуни. Керуючі програми для цих машин найчастіше створюються на мові G-код (G-code), яка є невід'ємною частиною систем автоматизованого проектування та виробництва (CAD/CAM). G-код містить інструкції щодо руху, швидкості, обертів інструментів та інших допоміжних функцій, дозволяючи точно контролювати весь процес виробництва.

1.2. 3D-Принтери: класифікація, принципи роботи та види друку

3D-принтери є особливим різновидом ЧПК-машин, що функціонують за принципом адитивного виробництва — тобто створення тривимірних об'єктів шляхом послідовного додавання матеріалу шар за шаром, на відміну від субтрактивних методів, де матеріал видаляється. Цей процес починається зі створення або отримання тривимірної цифрової моделі об'єкта, зазвичай у форматі CAD, яка потім програмно "нарізається" на серію тонких двовимірних шарів за допомогою спеціалізованого програмного забезпечення, так званого слайсера. Інформація про кожен такий "зріз" перетворюється на G-код, що керує рухами друкуючої головки або іншого механізму принтера по осях X, Y та Z, а також параметрами матеріалу, такими як температура та швидкість подачі.[6]

Класифікація 3D-принтерів переважно базується на технології, що використовується для затвердіння матеріалу. Одна з найдоступніших і найпоширеніших технологій – це моделювання методом наплавлення (FDM), також відоме як FFF (Fused Filament Fabrication). Тут термопластичний філамент – пластикова нитка – подається у нагрітий екструдер, де плавиться. Розплавлений матеріал потім видавлюється через сопло з контрольованою швидкістю і наноситься на робочу платформу шар за шаром. Після нанесення кожен шар швидко охолоджується і твердне, утворюючи основу для наступного. Переміщення сопла та платформи по осях X, Y, Z точно контролюється ЧПК, забезпечуючи формування об'єкта. Іншою, однією з найстаріших технологій, що забезпечує високу точність та гладку поверхню, є стереолітографія (SLA). Вона використовує ванну з фотополімерною смолою, яка твердне під впливом ультрафіолетового (УФ) лазера. Лазер послідовно "малює" контур кожного шару на поверхні смоли, що призводить до її затвердіння, після чого платформа опускається на висоту одного шару, і процес повторюється до повного формування об'єкта. Різновидом SLA є DLP (Digital Light Processing), де замість лазера використовується цифровий проектор для одночасного затвердіння цілого шару, що робить цей метод значно швидшим. Також існує метод селективного лазерного спікання (SLS), який використовує порошко-

вий матеріал, такий як поліаміди або метали. У цій технології високопотужний лазер вибірково спікає частинки порошку в заданих місцях, формуючи кожен шар. Після спікання шару, нова тонка куля порошку рівномірно розподіляється по поверхні, і процес повторюється. Важливою перевагою SLS є те, що порошок, який не був спічений, служить природною опорою для об'єкта, дозволяючи створювати дуже складні геометрії без необхідності у додаткових опорних структурах. Кожен з цих видів друку має свої унікальні переваги та обмеження, що стосуються використовуваних матеріалів, досяжної точності, швидкості виробництва та загальної вартості обладнання, роблячи їх придатними для широкого спектра застосувань, від швидкого прототипування до серійного виробництва функціональних деталей.[5]

1.3. Застосування 3D-Принтерів

Завдяки своїй здатності до швидкого та економічно ефективного створення складних тривимірних об'єктів, 3D-принтери знайшли надзвичайно широке застосування в різних галузях промисловості, медицини, освіти та навіть у повсякденному житті. В промисловості вони активно використовуються для швидкого прототипування, що дозволяє інженерам і дизайнерам оперативно тестувати нові ідеї, вносити корективи та прискорювати цикл розробки продукції. Також 3D-друк застосовується для створення інструментальних пристосувань, оснастки та шаблонів, які є важливими компонентами у виробничих лініях, а також для виготовлення функціональних кінцевих деталей у малих серіях або висококастомізованих продуктів, особливо в авіакосмічній та автомобільній галузях, де потрібні легкі та міцні компоненти складної форми.

У медицині 3D-принтери зробили революцію, дозволяючи виготовляти персоналізовані протези та ортези, що ідеально відповідають анатомії пацієнта, а також створювати хірургічні шаблони для підвищення точності операцій. Активно розвивається напрямок біодруку, де дослідники експериментують з друком живих тканин і навіть органів. У стоматології 3D-друк застосовується для виготовлення індивідуальних коронок, мостів та ортодонтичних кап. Освітня сфера також ви-

грає від 3D-друку, використовуючи принтери для створення наочних навчальних посібників, анатомічних моделей, деталей для STEM-проектів, що сприяє розвитку просторового мислення та інженерних навичок у студентів та школярів. На побутовому рівні 3D-друк дозволяє створювати індивідуальні предмети декору, іграшки, запасні частини для побутової техніки, елементи інтер'єру та різноманітні аксесуари, що відкриває нові можливості для творчості та ремонту.

Історія ЧПК сягає корінням у початок XIX століття, коли у 18М. Жакар винайшов ткацький верстат, керований перфорованими картками, що стало раннім проявом програмного керування. Проте сучасна концепція числового керування була розроблена у середині XX століття американським інженером Джоном Парсонсом. Його розробки активно застосовувалися в авіабудівній промисловості для створення складних аеродинамічних деталей, заклавши основи для подальшого розвитку.

Сучасні ЧПК-системи являють собою комп'ютеризовані комплекси, що керують верстатним та технологічним устаткуванням. Їхня архітектура подібна до архітектури електронно-обчислювальних машин і включає запам'ятовуючі пристрої, пульти керування, дисплеї та оперативну пам'ять. Центральним елементом є контролер, який може бути реалізований як на базі спеціалізованого промислового комп'ютера, так і на базі персонального комп'ютера. Ключовою характеристикою такого контролера є кількість осей, якими він здатний керувати, що визначає його обчислювальну потужність. Завдяки значному прогресу в обчислювальній техніці, різниця у можливостях між промисловими та персональними рішеннями суттєво зменшилася, дозволяючи персональним комп'ютерам ефективно виконувати функції ЧПК-контролерів.

Застосування ЧПК-машин є фундаментальним для сучасної автоматизації виробництва. Вони забезпечують високу точність та повторюваність виготовлення деталей, що значно знижує ймовірність дефектів. ЧПК-системи мінімізують вплив людського фактора, підвищуючи продуктивність та ефективність виробничих процесів. Однією з найважливіших переваг є можливість створення деталей надзвичайно складних геометричних форм, включно з тими, що формуються за

допомогою 3D-друку, які було б неможливо або економічно недоцільно виготовити традиційними методами. Також вони забезпечують швидке переналагодження та універсальність у роботі з різними матеріалами, від металів до композитів.

1.4. Висновки до розділу

Машини з числовим програмним управлінням, включаючи 3D-принтери, є наріжним каменем сучасного виробництва, забезпечуючи безпрецедентну точність та ефективність. Їхня здатність створювати складні геометрії та автоматизувати процеси значно прискорює інновації у різноманітних галузях. Постійний розвиток цих технологій обіцяє подальшу трансформацію промисловості та відкриття нових можливостей для інженерії та дизайну.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Arduino IDE: інтегроване середовище розробки як інструмент для наукових досліджень

Arduino IDE (Integrated Development Environment) є ключовим компонентом екосистеми Arduino, що забезпечує програмний інтерфейс для розробки та завантаження програмного коду, відомого як "скетчі", на мікроконтролери плат Arduino. Цей продукт, розроблений на принципах відкритого вихідного коду, став надзвичайно популярним у науковій спільноті, серед інженерів, студентів та ентузіастів завдяки своїй простоті використання, кросплатформенній сумісності та широким можливостям розширення. Його архітектура та філософія проектування, що наголошує на доступності та функціональності, роблять його ефективним інструментом для швидкого прототипування, збору даних та автоматизації експериментів у різноманітних наукових дисциплінах.

По суті, Arduino IDE являє собою спрощене середовище програмування, що включає текстовий редактор для написання коду на базі мов C/C++, область повідомлень для відображення системної інформації, текстову консоль для виведення даних та панель інструментів з основними функціями, такими як компіляція та завантаження коду. В його основі лежить інтерпретована версія C++, яка спрощує взаємодію з апаратним забезпеченням завдяки наявності бібліотек та API, що абстрагують складні низькорівневі операції з мікроконтролером. Це дозволяє дослідникам, які не мають глибоких знань у вбудованому програмуванні, швидко реалізовувати свої ідеї та зосереджуватися на суті наукового завдання, а не на деталях взаємодії з апаратною архітектурою. Для компіляції скетчів Arduino IDE використовує компілятор avr-gcc, а для взаємодії з платами через USB-інтерфейс потрібні відповідні драйвери, які, як правило, вже інтегровані у дистрибутив середовища.

Однією з фундаментальних переваг Arduino IDE для наукових застосувань є його відкрита архітектура та можливість використання сторонніх бібліотек. Ці бібліотеки значно розширюють функціонал плат Arduino, надаючи готові функції для роботи з різноманітними датчиками, актуаторами, модулями зв'язку та інши-

ми периферійними пристроями. Дослідники можуть легко імпортувати ці бібліотеки у свої проекти, що дозволяє швидко інтегрувати нові сенсори для вимірювання фізичних параметрів, керувати виконавчими механізмами, такими як двигуни або клапани, та встановлювати зв'язок з іншими пристроями або комп'ютерами. Це особливо цінно у випадках, коли необхідно швидко модифікувати експериментальне обладнання або адаптувати його до нових умов дослідження без значних витрат часу на розробку низькорівневого коду.

Структура типового скетчу Arduino є досить стандартизованою, що сприяє легкості навчання та уніфікації підходів. Кожен скетч складається щонайменше з двох основних функцій: setup() та loop(). Функція setup() виконується лише один раз при запуску або перезавантаженні мікроконтролера і використовується для ініціалізації портів, послідовного зв'язку, підключених датчиків та інших початкових налаштувань. Функція loop() виконується циклічно, забезпечуючи безперервне виконання основного алгоритму програми, наприклад, зчитування даних із сенсорів, обробку їх та надсилання керуючих сигналів. Така проста, але ефективна структура дозволяє легко організовувати логіку програми для моніторингу процесів, збору даних у реальному часі або виконання циклічних завдань у рамках експерименту.

Важливим аспектом Arduino IDE для наукової діяльності є його кросплатформенність. Середовище доступне для операційних систем Windows, macOS та Linux, що забезпечує гнучкість у виборі робочої станції та дозволяє науковим колективам працювати над проектами незалежно від використовуваних операційних систем. Функція послідовного монітора (Serial Monitor), інтегрована в IDE, надає можливість візуалізації даних, що надсилаються з Arduino плати через послідовний порт. Це дозволяє в реальному часі відстежувати значення з датчиків, відлагоджувати код та контролювати хід експерименту, що є незамінним для досліджень, пов'язаних з вимірюваннями та аналізом динамічних процесів. Окрім текстового виведення, існують також можливості для графічного відображення даних за допомогою послідовного плотера (Serial Plotter), що значно спрощує аналіз тенденцій та взаємозв'язків у зібраних даних.

Arduino IDE, як інтегроване середовище розробки, пропонує дослідникам потужний, доступний та гнучкий інструмент для реалізації широкого спектра наукових проектів. Його простота використання, розширюваність за рахунок бібліотек, стандартизована структура коду, кросплатформенність та інтегровані засоби для відлагодження та моніторингу даних роблять його ідеальним вибором для швидкого прототипування, автоматизації експериментів, збору та попередньої обробки даних, що значно прискорює та спрощує процес наукових досліджень у багатьох дисциплінах, від фізики та біології до інженерії та робототехніки. Ця платформа демократизувала доступ до вбудованих систем, дозволивши вченим зосередитися на формулюванні гіпотез та отриманні результатів, а не на низькорівневому програмуванні та складнощах апаратної взаємодії.

2.2. Протокол STEP/DIR/ENABLE: фундаментальні принципи керування кроковими рушіями.

Протокол STEP/DIR/ENABLE (S/D/E) є стандартизованим та широко застосовуваним інтерфейсом для дискретного керування кроковими двигунами в системах автоматизації, робототехніки та, зокрема, у верстатах з числовим програмним керуванням (ЧПК). Цей протокол відзначається своєю простотою, ефективністю та високою точністю позиціонування, що робить його ідеальним для застосувань, які вимагають прецизійного контролю над кутовим положенням або лінійним переміщенням.

Протокол базується на використанні трьох окремих логічних сигналів, кожен з яких виконує специфічну функцію. Основний імпульс, що викликає дискретне переміщення крокового двигуна, передається через лінію STEP. Кожен позитивний або негативний фронт (залежно від конфігурації драйвера) на цій лінії викликає поворот ротора двигуна на один крок, що відповідає мінімальному кутовому переміщенню. Частота імпульсів на лінії STEP безпосередньо корелює зі швидкістю обертання двигуна: чим вища частота, тим швидше обертається двигун. Точність позиціонування досягається за рахунок того, що контролер може точно керувати кількістю цих імпульсів. Напрямок обертання крокового двигуна

визначає бінарний логічний сигнал DIR. Як правило, високий логічний рівень (HIGH) може означати один напрямок обертання, а низький (LOW) – протилежний. Зміна стану лінії DIR повинна відбуватися до подачі імпульсів STEP, щоб гарантувати коректний напрямок руху. Третій сигнал, ENABLE, використовується для активації або деактивації драйвера крокового двигуна. Коли драйвер активований (зазвичай низьким логічним рівнем, LOW), він подає струм на обмотки двигуна, утримуючи його в поточному положенні або дозволяючи йому рухатися згідно з сигналами STEP/DIR. Деактивація драйвера (зазвичай високим логічним рівнем, HIGH) припиняє подачу струму, що дозволяє вільно переміщувати ротор двигуна без опору; це може бути корисним для ручного позиціонування або для економії енергії, коли двигун не використовується.[3]

Протокол має низку ключових переваг, що сприяли його широкому поширенню. Його простота реалізації вимагає мінімальної кількості виводів мікроконтролера (три на кожен двигун), що значно спрощує апаратну частину системи. Завдяки дискретному характеру керування кроками, протокол забезпечує високу точність та повторюваність позиціонування, що є критично важливим для багатьох прецизійних застосувань. Гнучкість у керуванні швидкістю досягається прямим контролем частоти імпульсів STEP, дозволяючи динамічно змінювати швидкість під час роботи. Крім того, протокол демонструє високу сумісність з широким спектром драйверів крокових двигунів, від простих до високопродуктивних, що забезпечує гнучкість у виборі компонентів. Застосування протоколу S/D/E охоплює широкий спектр галузей, включаючи керування осями X, Y, Z та екструдером у 3D-принтерах, забезпечення прецизійного позиціонування інструмента у ЧПК-верстатах (фрезерних, лазерних, токарних), керування суглобами роботів, де потрібне точне кутове переміщення, а також використання в автоматизованих системах позиціонування для оптичних систем, медичного обладнання та вимірювальних приладів.

Незважаючи на значні переваги, протокол має певні обмеження, головним з яких є відсутність зворотного зв'язку щодо фактичного положення двигуна. Це означає, що контролер не "знає", чи дійсно двигун виконав усі кроки або чи не

було пропущено кроків через перевантаження або зовнішні перешкоди. Для застосувань, де пропуск кроків є неприпустимим, протокол S/D/E часто доповнюється використанням енкодерів або інших датчиків зворотного зв'язку, що дозволяють реалізувати замкнений контур керування. Крім того, для досягнення дуже плавного руху та зменшення вібрацій, багато сучасних драйверів підтримують мікрокроковий режим (microstepping), де один фізичний крок двигуна розділяється на декілька мікрокроків. Це дозволяє підвищити роздільну здатність позиціонування та зменшити шум, хоча й може збільшити кількість імпульсів STEP, необхідних для повного оберту.[4]

2.3. Висновки до розділу

Протокол STEP/DIR/ENABLE є надійним та широко розповсюдженим рішенням для дискретного керування кроковими двигунами. Його простота в реалізації, висока точність позиціонування та гнучкість роблять його невід'ємною частиною багатьох сучасних автоматизованих систем. Хоча він і не надає вбудованого зворотного зв'язку, його інтеграція з додатковими сенсорами може забезпечити високонадійний замкнений контур керування, що розширює сферу його застосування у найвимогливіших проектах.

РОЗДІЛ 3. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Алгоритм Брезенхема.

Алгоритм Брезенхема — це основний і надзвичайно ефективний метод у комп'ютерній графіці, розроблений Джеком Елтоном Брезенхемом у 1962 році в IBM для малювання ліній та кіл на растрових дисплеях. Його головна перевага полягає у використанні виключно цілочисельної арифметики, що дозволяє уникнути повільних операцій з плаваючою комою та накопичення помилок округлення, забезпечуючи високу швидкість і точність.[1]

Принцип роботи алгоритму

Для малювання ліній алгоритм Брезенхема послідовно вибирає найближчий до ідеальної лінії піксель. Він починає з визначення, яка вісь (X чи Y) має більший приріст, і ітерує по ній. Ключовим елементом є так званий параметр рішення (або "error term") — цілочисельне значення, що відображає "відстань" між поточним пікселем та ідеальною лінією. Якщо цей параметр менше певного порогу (зазвичай нуля), це означає, що ідеальна лінія ближче до пікселя, що лежить на тій самій горизонтальній (або вертикальній) лінії, що й попередній. У протилежному випадку, якщо параметр перевищує поріг, ідеальна лінія ближче до пікселя, який потребує зміщення по діагоналі. На кожному кроці алгоритм малює обраний піксель і оновлює параметр рішення, "крокуючи" від початкової до кінцевої точки і вибираючи найближчі до ідеальної лінії пікселі.[3]

Переваги та застосування

Така реалізація забезпечує алгоритму Брезенхема кілька значних переваг: швидкість, завдяки цілочисельній арифметиці, що критично для графіки в реальному часі; точність, оскільки відсутність операцій з плаваючою комою виключає проблеми округлення; та простота реалізації, що робить його доступним для розуміння та впровадження. Алгоритм Брезенхема знайшов широке застосування не лише в комп'ютерній графіці для малювання базових примітивів, таких як лінії,

кола та еліпси, але й у пристроях для друку зображень, робототехніці для планування траєкторій руху та в різних алгоритмах обробки зображень, де потрібна попіксельна точність. Існують також узагальнення цього алгоритму для побудови інших геометричних фігур, що робить його наріжним каменем у галузі комп'ютерної графіки.

Алгоритм Брезенгема для дискретизації відрізка у тривимірному просторі

Вихідні дані

Нехай задано дві точки:

$$P_0 = (x_0, y_0, z_0),$$
$$P_1 = (X_1, y_1, z_1),$$
$$x_0, y_0, z_0, x_1, y_1, z_1 \in Z$$

Визначимо прирости координат:

$$\Delta_x = |x_1 - x_0|,$$
$$\Delta_y = |y_1 - y_0|,$$
$$\Delta_z = |z_1 - z_0|.$$

Напрямки руху для кожної координати задаються як:

$$s_x = sign(x_1 - x_0),$$
$$s_y = sign(y_1 - y_0),$$
$$s_z = sign(z_1 - z_0), sz = \text{sign}(z1 - z0),$$

де

$sign(a) = \{1, \text{якщо} a > 0, 0, \text{якщо} a = 0, -1, \text{якщо} a < 0\}$

Вибір провідної осі

Провідною вважається та координата, для якої приріст є найбільшим:

Якщо $\Delta_x \geq \Delta_y i \Delta_x \geq \Delta_z$, то головна вісь — x.

Якщо$\Delta_y \geq \Delta_x i \Delta_y \geq \Delta_z$, то головна вісь — y.

Інакше — головна вісь z.

Нехай для прикладу головна вісь — x. Тоді обчислюються дві змінні похибки:

$$e_x = 2\Delta_y - \Delta_x, e_y = 2\Delta_y - \Delta_x, e_z = 2\delta_z - \Delta_x.$$

Ітеративно змінюємо координати x,y,z у напрямку від P0 до P1 , на кожному кроці:

$$x := x + s_x$$

Якщо $e_y \geq 0$, тоді $y := y + s_y i e_y := e_y - 2\Delta_x$

Якщо $e_z \geq 0$, тоді $z := z + s_z i e_z := e_z - 2\Delta_x$

Оновлюємо похибки:

$$e_y := e_y + 2\Delta_Y,$$

$$e_z := e_z + 2\Delta_z$$

Аналогічно алгоритм перебудовується для провідної осі y або z

У Результаті алгоритм генерує впорядковану послідовність дискретних точок:

$$\{x_i, y_i, z_i\}_{i\ =\ 0}^{N},$$

що апроксимують відрізок між P0 і P1 , де $N = max(\Delta_x, \Delta_y, \Delta_z)$.[3]

Дискретизація відрізка у тривимірному просторі за допомогою алгоритму Брезенгема показана на рис 1. Червоним кольором позначено вокселі, які проходить відрізок між початковою і кінцевою точками у кубічній сітці. Координатні осі X,Y і Z показано для просторової орієнтації.

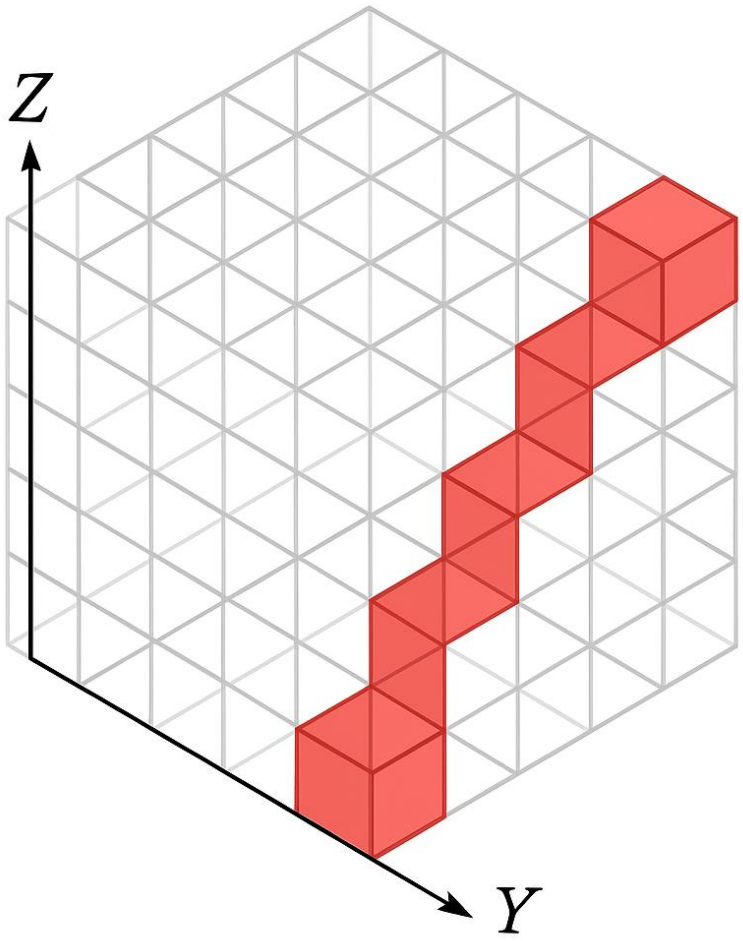


Рисунок 3.1. Дискретизація відрізка у тривимірному просторі.

3.2. Загальні положення для кіл і еліпсів.

Алгоритм Брезенгема, спочатку розроблений для дискретизації відрізків у двовимірному растровому середовищі, був розширений для побудови кіл та еліпсів з використанням виключно цілочисельних обчислень. Ці модифікації дозволяють апроксимувати гладкі криві у вигляді набору вокселів, які найкраще відповідають їхній геометричній формі.

У тривимірному просторі побудова кіл та еліпсів розглядається як дискретизація кривих, що лежать у площині, розташованій у 3D-просторі. Найчастіше ви-

користовується ортогональне проєктування кривої у площинах XY, XZ або YZ, однак загальний випадок допускає довільну орієнтацію.

Побудова кола у 3D-просторі (в площині XY)

Вихідні дані:

Центр кола:

$$C = (x_0, y_0, z_0)$$

Радіус:

$$r \in N$$

Площина побудови: XY (тобто $z = z_0 = const$)

Алгоритм для побудови кола використовує симетрію відносно осей і діагоналей. Достатньо згенерувати одну восьму дуги кола, а потім відобразити її в інші частини.

Початкові значення:

$$x = 0, y = r, d = 3 - 2r$$

На кожному кроці, залежно від значення d, оновлюються координати та значення дискримінанта:

Якщо $d > 0$,:

$d := d + 4x + 6$

Інакше:

$d := d + 4(x - y) + 10$,

$y := y - 1$

У будь-якому випадку:

$x := x + 1$

Кожна обчислена точка (x,y) використовується для генерації восьми симетричних точок на площині XY:

$$(x_0 \mp x, y_0 \mp y, z_0),$$

Цей процес повторюється, доки

$$x \leq y.$$

Генералізація у довільній площині:

Щоб побудувати коло в площині, не обов'язково ортогональній координатним осям, потрібно застосувати перетворення координат (наприклад, обертання або матрицю повороту).[1]

Побудова еліпса у 3D-просторі (в площині XY)

Вихідні дані:

Центр еліпса:

$C = (x_0, y_0, z_0) C$=($x0$,$y0$,$z0$)

Напівосі:

a (по осі X),

b (по осі Y)

Площина побудови: XY (тобто $z = z_0$)

Побудова еліпса виконується в двох регіонах, які відрізняються тим, яка координата є домінуючою.

Початкові значення:

$$x=0, y=d_1 = d^2 - a^2 \cdot b + \frac{1}{4}a^2$$

Регіон 1 ($2b^2 \cdot x \geq 2a^2 \cdot y$):

Поки умова виконується:

Якщо $d1<0$:

$$d_1 := d_1 + b^2(2x + 3)$$

Інакше:

$$d_1 := d_1 + b^2(2x + 3) + a^2(-2y + 2), y := y - 1$$

У будь-якому випадку:

$$x := x + 1$$

Регіон 2 ($2b^2 \cdot x \geq 2a^2 \cdot y$)

Переходимо до:

$$d2 = b^2\left(x + \frac{1}{2}\right)^2 + a^2(y - 1)^2 - a^2$$

Поки $y \geq 0$

Якщо $d_2 \geq 0$,:

$$d_2 := d_2 + a^2(-2y + 3)$$

Інакше:

$$d_2 := d_2 + b^2(2x + 2) + a^2(-2y + 3), x := x + 1$$

У будь-якому випадку: $y := y - 1$

Симетрія, кожна точка відображається у чотирьох напрямках:

$$(x_0 \mp x, y_0 \mp y, z_0)$$

Побудова кіл/еліпсів у площинах XZ або YZ залишається ідентичним, однак координата z змінюється замість y або x. Наприклад, для кола в площині XZ:

$$(x_0 \mp x, y_0, z_0 \mp z)$$

Для еліпса з півосями

a по осі X і b по осі Z:

$$(x_0 \mp x, y_0, z_0 \mp y)$$

Побудова в довільній площині

Для побудови кола чи еліпса у довільно орієнтованій площині слід:

Побудувати криву у стандартній площині (наприклад, XY),

Застосувати до координат кожної точки перетворення повороту:

$$\overrightarrow{P_{\text{нове}}} = R\vec{p}XY + \vec{C}\ ,$$

де

R— матриця обертання,

C — центр у 3D.

3.3. Матриця повороту у тривимірному просторі

Для реалізації побудови геометричних об'єктів, таких як прямі, кола чи еліпси, в довільно орієнтованих площинах тривимірного простору необхідно використовувати перетворення координат на основі матриць повороту.[3] Поворот точки у просторі навколо однієї з координатних осей реалізується множенням її координат на відповідну матрицю повороту розміру 3×3

Поворот на кут θ навколо осі X описується матрицею:

$$R_x(\theta) = \begin{bmatrix} 100 \\ 0cos\theta sin\theta \\ 0 - sin\theta cos\theta \end{bmatrix}$$

Поворот на кут ϕ навколо осі Y задається наступною матрицею:

$$R_y(\phi) = \begin{bmatrix} cos\phi 0sin\phi \\ 010 \\ -sin\phi 0cos\phi \end{bmatrix}$$

Аналогічно, поворот навколо осі Z на кут ψ описується матрицею:

$$R_z(\psi) = \begin{bmatrix} cos\psi - sin\psi 0 \\ sin\psi cos\psi 0 \\ 001 \end{bmatrix}$$

У випадку повороту в довільну орієнтацію у просторі, застосовується композиція обертань у певному порядку. Типово використовується наступна композиція:

$$R = R_z(\psi) \cdot R_y(\phi) \cdot R_x(\theta),$$

де множення матриць виконується справа наліво відповідно до порядку застосування поворотів. Важливо зазначити, що множення матриць не є комутативним, тобто

$$R1R2 \neq R2R1.$$

Застосування матриці повороту до точки $p = (x, y, z)^T$ з урахуванням центру обертання $C = (x_0, y_0, z_0)^T$ реалізується за формулою:

$p_{\text{нова}} = R \cdot (p - C) + C$

У загальному випадку повороту навколо довільної осі $u = (u_x, u_y, u_z)$, що є одиничним вектором, використовується формула Родріґеса:

$R(\theta, u) = cos\theta \cdot I + sin\theta \cdot [u]_\times + (1 - cos\theta) \cdot (u \otimes u)$

де

I — одинична матриця,$[u]_\times$ — кососиметрична матриця векторного добутку, а

$u \otimes u$ — тензорний (зовнішній) добуток вектора u на себе.

Ці перетворення дозволяють реалізовувати геометричні побудови у довільних площинах тривимірного простору, зокрема для побудови кривих методом Брезенгема після застосування відповідного обертання до кожної точки дискретизації.

3.4. Висновки до розділу

У розділі 3 проаналізовано алгоритмічне забезпечення геометричних побудов у 3D-просторі, головним чином на основі Алгоритму Брезенхема. Головна перевага алгоритму полягає у використанні виключно цілочисельної арифметики, що забезпечує високу швидкість і точність дискретизації. Ця методика успішно узагальнена для:

Дискретизації відрізків у 3D за принципом провідної осі та двох змінних похибок.

Побудови кіл та еліпсів у стандартних (XY, XZ, YZ) площинах, використовуючи симетрію та цілочисельні дискримінанти.

Для забезпечення побудови об'єктів у довільно орієнтованих площинах використовується апарат матриць повороту (Rx,Ry,Rz). Це дозволяє трансформувати координати дискретизованих точок (отриманих методом Брезенхема у стандартній площині) у потрібну орієнтацію у 3D.

Таким чином, математичне забезпечення, представлене в розділі, формує надійну, ефективну та гнучку основу для програмної реалізації точного відображення базових геометричних примітивів у будь-якій орієнтації тривимірного простору.

РОЗДІЛ 4. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

4.1. Визначення констант

Ініціалізація програмного середовища починається з декларації констант, які слугують для конфігурації ключових параметрів системи. Константа VERSION ("0.01") позначає ітерацію розробки прошивки, що є критично важливим для контролю версій та ідентифікації програмного забезпечення. Швидкість послідовної передачі даних, необхідна для комунікації з хост-пристроєм, визначена як BAUD_RATE (9600 біт/с), що є стандартним для багатьох інтерфейсів UART. Максимальна ємність буфера для вхідних команд, що надходять через послідовний порт, встановлена константою MAX_BUFER (128 байт), обмежуючи довжину одного командного рядка. Параметр STEPS_PER_TURN (400) задає кількість кроків, які повинен виконати кроковий двигун для здійснення одного повного оберту, що є фундаментальним для перетворення лінійних переміщень у дискретні імпульси двигуна. Мінімальна часова затримка між послідовними кроками двигуна, виражена в мікросекундах, фіксується константою MIN_STEP_DELAY (50.0 мкс), що, у свою чергу, визначає теоретичну максимальну швидкість обертання. На основі цього значення розраховується MAX_FEED_RATE, що представляє максимальну швидкість подачі (лінійного переміщення) у кроках за секунду, тоді як MIN_FEED_RATE (0.01 кроків/с) встановлює нижню межу швидкості. Для однозначного визначення напрямку дугових переміщень використовуються константи ARC_CW (1) для обертання за годинниковою стрілкою та ARC_CCW (-1) для обертання проти годинникової стрілки. Деталізація апроксимації дуг лінійними сегментами контролюється константою MM_PER_SEGMENT (10), що вказує на кількість сегментів на міліметр довжини дуги, тим самим впливаючи на точність траєкторії. Код показано на рис.2

```
/* Константи */
#define VERSION "0.01"       // Версія прошивки
#define BAUD_RATE 9600       // Швидкість UART
#define MAX_BUFER 128        // Максимальний розмір буфера команд
#define STEPS_PER_TURN 400   // Кроків двигуна на один повний оберт
#define MIN_STEP_DELAY 50.0  // Мінімальна затримка між кроками (мкс)
#define MAX_FEED_RATE (1000000.0/MIN_STEP_DELAY) // Максимальна подача (швидкість руху)
#define MIN_FEED_RATE 0.01   // Мінімальна подача

// Напрямок обертання для дуг
#define ARC_CW 1     // за годинниковою стрілкою
#define ARC_CCW -1   // проти годинникової стрілки

// Скільки сегментів на мм при розбитті дуг
#define MM_PER_SEGMENT 10
```

Рисунок 4.1. Декларації констант для конфігурації ключових параметрів системи

4.2. Призначення виводів та глобальні змінні

Секція визначення апаратних інтерфейсів та глобальних змінних є критичною для функціонування контролера. Кожен вивід Arduino, відповідальний за керування двигунами, чітко асоційований з його функцією: для осі X це MOTOR_X_STEP (2), MOTOR_X_DIR (5) та MOTOR_X_ENA (8); аналогічно для осі Y: MOTOR_Y_STEP (3), MOTOR_Y_DIR (6), MOTOR_Y_ENA (8); і для осі Z: MOTOR_Z_STEP (4), MOTOR_Z_DIR (7), MOTOR_Z_ENA (8). Важливо зазначити, що у поточному дизайні вивід увімкнення двигуна (ENA) є спільним для всіх осей (пін 8), що спрощує схему підключення. Додатково, визначено виводи для кінцевих вимикачів: SWITCH_1 (9) та SWITCH_2 (10), хоча їх програмна реалізація у наданому фрагменті коду відсутня. На рис.3 показано фрагмент коду із визначення апаратних інтерфейсів.

```
/* Піни двигунів і енкодерів */
#define MOTOR_X_STEP 2
#define MOTOR_X_DIR 5
#define MOTOR_X_ENA 8

#define MOTOR_Y_STEP 3
#define MOTOR_Y_DIR 6
#define MOTOR_Y_ENA 8

#define MOTOR_Z_STEP 4
#define MOTOR_Z_DIR 7
#define MOTOR_Z_ENA 8

#define SWITCH_1 9
#define SWITCH_2 10
```

Рисунок 4.2. Секція із визначенням апаратних інтерфейсів

Серед глобальних змінних Serial_Buffer є масивом символів розміром MAX_BUFER, що слугує як тимчасове сховище для вхідних команд, отриманих через послідовний порт. Змінна symqnty динамічно відстежує поточну кількість символів, записаних у Serial_Buffer. Поточна позиція інструмента у тривимірному просторі зберігається у змінних з плаваючою комою px, py, pz, які представляють координати за осями X, Y та Z відповідно в міліметрах. Поточна швидкість подачі, виражена в кроках за секунду, акумулюється у змінній fr. Величина step_delay, розрахована на основі fr, визначає необхідну мікросекундну затримку між послідовними кроками двигунів. Бінарна змінна mode_abs функціонує як прапорець, що визначає поточний режим інтерпретації координат: значення 1 вказує на абсолютний режим (команди інтерпретуються як абсолютні позиції), тоді як 0 вказує на відносний режим (команди інтерпретуються як зміщення від поточної позиції). Секція коду показана на рис. 4

```
/* Глобальні змінні для координат та подачі */
char Serial_Buffer [MAX_BUFER]; // Буфер для отримання команд
int symqnty;                    // Кількість символів в буфері

float px = 0, py = 0, pz = 0;  // Поточна позиція у mm
float fr = 0;                  // Поточна швидкість подачі в кроках/сек

long step_delay;               // Затримка між кроками (мкс)
char mode_abs=1;               // Режим координат: 1 - абсолютний, 0 - відносний
```

Рисунок 4.3. Секція коду із глобальними змінними

4.3. Функціональні модулі керування рухом

Програмне забезпечення побудовано на модульному принципі, де кожна функція виконує специфічне завдання. Фундаментальною для руху є функція Motor_X_Step(int dir), яка ініціює виконання одного кроку двигуна осі X. Її архітектура передбачає активацію двигуна, встановлення напрямку руху (dir: 1 або -1) та генерацію імпульсу кроку, що забезпечує дискретне переміщення. Тут має бути ілюстрація функції Motor_X_Step. Аналогічно, функції Motor_Y_Step(int dir) та Motor_Z_Step(int dir) забезпечують ідентичний механізм керування для осей Y та Z відповідно. Тут має бути ілюстрація функції Motor_Y_Step. Код проілюстровано на рис.5

```
/* Зробити один крок по осі X з напрямком dir (1 чи -1) */
void Motor_X_Step(int dir) {
    digitalWrite(MOTOR_X_ENA, LOW); // Вмикаємо двигун
    digitalWrite(MOTOR_X_DIR, dir == 1 ? HIGH : LOW); // Встановлюємо напрямок
    digitalWrite(MOTOR_X_STEP, HIGH); // Сигнал кроку
    digitalWrite(MOTOR_X_STEP, LOW);
}

/* Зробити один крок по осі Y */
void Motor_Y_Step(int dir) {
    digitalWrite(MOTOR_Y_ENA, LOW);
    digitalWrite(MOTOR_Y_DIR, dir == 1 ? HIGH : LOW);
    digitalWrite(MOTOR_Y_STEP, HIGH);
    digitalWrite(MOTOR_Y_STEP, LOW);
}

/* Зробити один крок по осі Z */
void Motor_Z_Step(int dir) {
    digitalWrite(MOTOR_Z_ENA, LOW);
    digitalWrite(MOTOR_Z_DIR, dir == 1 ? HIGH : LOW);
    digitalWrite(MOTOR_Z_STEP, HIGH);
    digitalWrite(MOTOR_Z_STEP, LOW);
}
```

Рисунок 4.4. Функції для здійснення руху рушіями.

Функція Disable() рис.6 реалізує механізм аварійного або програмного вимкнення всіх крокових двигунів шляхом встановлення високого логічного рівня на відповідних виводах EN (Enable). Це необхідно для забезпечення безпеки оператора, зменшення енергоспоживання у стані спокою або для можливості ручного позиціонування механічних компонентів.

```
/* Вимкнути усі двигуни (для безпеки чи збереження енергії) */
void Disable() {
    digitalWrite(MOTOR_X_ENA, HIGH);
    digitalWrite(MOTOR_Y_ENA, HIGH);
    digitalWrite(MOTOR_Z_ENA, HIGH);
}
```

Рисунок 4.5. Секція коду із функцією Disable().

Початкова конфігурація виводів мікроконтролера здійснюється функцією Setup_Controller() рис.7 , яка ініціалізує всі піни, що використовуються для керування двигунами (STEP, DIR, ENA), як вихідні. Це дозволяє мікроконтролеру формувати необхідні керуючі сигнали для драйверів двигунів.

```
/* Налаштування пінів Arduino */
void Setup_Controller() {
    pinMode(MOTOR_X_ENA, OUTPUT);
    pinMode(MOTOR_Y_ENA, OUTPUT);
    pinMode(MOTOR_Z_ENA, OUTPUT);
    pinMode(MOTOR_X_STEP, OUTPUT);
    pinMode(MOTOR_Y_STEP, OUTPUT);
    pinMode(MOTOR_Z_STEP, OUTPUT);
    pinMode(MOTOR_X_DIR, OUTPUT);
    pinMode(MOTOR_Y_DIR, OUTPUT);
    pinMode(MOTOR_Z_DIR, OUTPUT);
}
```

Рисунок 4.6. Фрагмент коду із реалізацією функції Setup_Controller().

Функція Pause(long us) рис.8 забезпечує точне часове затримання виконання програми, що є критично важливим для контролю швидкості руху двигунів. Вона поділяє загальний інтервал затримки (us, в мікросекундах) на мілісекундні та мікросекундні компоненти, використовуючи вбудовані функції delay() та delayMicroseconds() відповідно.

```
/* Затримка з поділом на мс та мкс */
void Pause(long us) {
    delay(us / 1000);          // мілісекунди
    delayMicroseconds(us % 1000); // мікросекунди
}
```

Рисунок 4.7. Секція коду із функцією Pause().

Динамічне керування швидкістю переміщення реалізується функцією Feed_Rate(float nfr) рис. 9. Вона приймає бажану швидкість подачі nfr (у кроках за секунду), валідує її в межах встановлених констант (MIN_FEED_RATE та MAX_FEED_RATE), та, у випадку валідності, конвертує її у затримку між кроками (step_delay у мікросекундах), яка використовується алгоритмами руху.

```
/* Встановлення швидкості подачі (feed rate)
   nfr - швидкість у кроках/сек */
void Feed_Rate(float nfr) {
    if (fr == nfr) return; // Якщо не змінилось - не змінюємо
    if (nfr > MAX_FEED_RATE || nfr < MIN_FEED_RATE) {
        Serial.print(F("Помилка: швидкість поза межами: "));
        Serial.println(nfr);
        return;
    }
    step_delay = 1000000.0 / nfr; // Переводимо швидкість у затримку між кроками (мкс)
    fr = nfr;
}
```

Рисунок 4.8. Фрагмент коду із функцією Feed_Rate().

Функція Position(float npx, float npy, float npz=0) рис. 10 виконує оновлення внутрішніх змінних, що представляють поточні координати інструмента (px, py, pz), без ініціювання фізичного переміщення двигунів. Це дозволяє програмно встановлювати нульову або іншу початкову точку.

```
/* Встановлення нової позиції */
void Position(float npx, float npy, float npz=0) {
    px = npx;
    py = npy;
    pz = npz;
}
```

Риунок 4.9 Ілюстрація реалізації функції Position().

4.5. Алгоритми траєкторного планування

Фундаментом для реалізації лінійного руху є функція TD_Line(float newx, float newy, float newz). Цей модуль застосовує адаптований алгоритм Брезенхема для 3D, що дозволяє забезпечити синхронізоване та плавне лінійне переміщення інструмента від поточної позиції до заданих кінцевих координат (newx, newy, newz). Алгоритм розраховує прирости та напрямки для кожної осі, визначаючи "головну" вісь руху. Потім він ітеративно генерує кроки, підтримуючи співвідношення між рухами осей та застосовуючи step_delay для контролю швидкості. Після завершення руху, глобальні змінні позиції оновлюються. Код функції показаний на рис.11

```
void TD_Line(float newx, float newy, float newz) {
    long i;
    long dx = abs(newx - px);
    long dy = abs(newy - py);
    long dz = abs(newz - pz);

    int dirx = (newx > px) ? 1 : -1;
    int diry = (newy > py) ? 1 : -1;
    int dirz = (newz > pz) ? 1 : -1;

    long max_delta = max(dx, max(dy, dz)); // Визначаємо головну вісь руху
    long over_x = max_delta / 2;
    long over_y = max_delta / 2;
    long over_z = max_delta / 2;

    for (i = 0; i < max_delta; i++) {
        over_x += dx;
        over_y += dy;
        over_z += dz;

        if (over_x >= max_delta) {
            over_x -= max_delta;
            Motor_X_Step(dirx);
        }
        if (over_y >= max_delta) {
            over_y -= max_delta;
            Motor_Y_Step(diry);
        }
        if (over_z >= max_delta) {
            over_z -= max_delta;
            Motor_Z_Step(dirz);
        }
        Pause(step_delay);
    }
    // Оновлюємо координати позиції після руху
    px = newx;
    py = newy;
    pz = newz;
}
```

Рисунок 4.10. Фрагмент фз кодом функції TD_Line().

Математична функція atan3(float dy, float dx) рис.12 є розширенням стандартної функції atan2(). Її призначення — обчислення кута у радіанах від 0 до 2π (від 0 до 360 градусів) на основі компонентів вектора dy та dx. Це забезпечує коректну обробку кутів для всіх квадрантів, що є критично важливим для точного планування дугових траєкторій.

```
/* Функція, що повертає кут у діапазоні 0...2*PI (0 - 360 градусів) */
float atan3(float dy, float dx) {
    float a = atan2(dy, dx);
    if (a < 0) a += 2.0 * PI;
    return a;
}
```

Рисунок 4.11. Фрагмент коду із реалізацією математичної функція atan3.

Складний рух по дузі, інтегрований з інтерполяцією по осі Z, реалізується функцією TD_Arc(float cx, float cy, float startz, float x, float y, float endz, float dir). Показано на рис.13. Цей модуль приймає координати центру дуги (cx, cy), початкову (startz) та кінцеву (endz) координати по осі Z, а також кінцеві координати дуги в площині XY (x, y) та її напрямок (ARC_CW або ARC_CCW). Функція розраховує радіус дуги, початковий та кінцевий кути, а також загальну довжину. Для забезпечення плавного руху, дуга розбивається на численні дрібні лінійні сегменти, кількість яких визначається константою MM_PER_SEGMENT. Для кожного сегмента обчислюються проміжні координати, включаючи лінійну інтерполяцію по осі Z, і потім для виконання цього сегмента викликається функція TD_Line(). Фінальний сегмент забезпечує точне позиціонування на кінцевій точці дуги.

```
/* --- Рух по дузі у площині XY з інтерполяцією Z --- */
/* cx, cy - центр дуги */
/* startz, endz - початковий і кінцевий Z */
/* x, y, z - кінцева точка */
/* dir - напрямок руху: ARC_CW або ARC_CCW */
void TD_Arc(float cx, float cy, float startz, float x, float y, float endz, float dir) {
    float dx = px - cx;
    float dy = py - cy;
    float radius = sqrt(dx*dx + dy*dy);

    float angle1 = atan3(dy, dx);
    float angle2 = atan3(y - cy, x - cx);
    float theta = angle2 - angle1;

    // Корекція кута для правильного напряму руху
    if (dir > 0 && theta < 0) angle2 += 2 * PI;
    else if (dir < 0 && theta > 0) angle1 += 2 * PI;
    theta = angle2 - angle1;

    float arc_length = abs(theta) * radius;
    int segments = ceil(arc_length * MM_PER_SEGMENT);

    for (int i = 0; i < segments; ++i) {
        float scale = (float)i / segments;
        float angle = angle1 + theta * scale;
        float nx = cx + cos(angle) * radius;
        float ny = cy + sin(angle) * radius;
        float nz = startz + (endz - startz) * scale; // Інтерполяція Z

        TD_Line(nx, ny, nz); // Рух по сегменту дуги
    }
    // Заключний рух до кінцевої точки дуги
    TD_Line(x, y, endz);
}
```

Рисунок 4.12. Фрагмент коду із реалізацією складного руху по дузі. Функція TD_Arc().

4.5. Модулі парсингу та взаємодії з користувачем

Функція Parse_Number(char code, float val) є ключовим компонентом для інтерпретації числових значень з вхідного командного буфера (Serial_Buffer). Вона виконує пошук заданого символьного code (наприклад, 'X', 'Y', 'Z', 'F', 'I', 'J') у буфері. У разі знаходження code, функція використовує atof (ASCII to Float) для конвертації наступної послідовності символів у число з плаваючою комою. Якщо code не знайдено, повертається значення val, яке слугує як значення за замовчуванням. Це забезпечує гнучке вилучення параметрів з G-команд. Ілюстрація функції Parse_Number показана на рис.14

```
/* --- Функція парсингу числового значення по коду (X, Y, Z, F, I, J тощо) --- */
/* Якщо не знайдено відповідного коду, повертає val (поточне значення) */
float Parse_Number(char code, float val) {
    char *ptr = Serial_Buffer;
    while ((long)ptr > 1 && (*ptr) && (long)ptr < (long)Serial_Buffer + symqnty) {
        if (*ptr == code) {
            return atof(ptr + 1);
        }
        ptr = strchr(ptr, ' ');
        if (!ptr) break;
        ptr++;
    }
    return val;
}
```

Рисунок 4.13. ілюстрація функції Parse_Number

Для стандартизованого виведення інформації через послідовний порт використовується допоміжна функція OutPut(const char *code, float val), показана на рис.15. Вона форматує вихідні дані у вигляді "КОД ЗНАЧЕННЯ" (наприклад, "X10.5"), що спрощує відлагодження та моніторинг стану системи.

```
/* --- Вивід позиції або повідомлень --- */
void OutPut(const char *code, float val) {
    Serial.print(code);
    Serial.println(val);
}
```

Рисунок 4.14. Фрагмент коду. Код функції OutPut().

Функція Current_Position() надає користувачеві актуальну інформацію про поточний стан контролера, виводячи поточні координати інструмента (X, Y, Z) та швидкість подачі (F), а також інформуючи про активний режим координат (абсолютний або відносний). Код функції зображений на рис.16

```
/* Вивід поточної позиції та подачі */
void Current_Position() {
    OutPut("X", px);
    OutPut("Y", py);
    OutPut("Z", pz);
    OutPut("F", fr);
    Serial.println(mode_abs ? "Режим: Абсолютний" : "Режим: Відносний");
}
```

Рисунок 4.15. Ілюстрація коду функції Current_Position().

Модуль Help() призначений для надання інтерактивної довідки користувачеві. Код модуля показаний на рис.17. Він виводить на послідовний порт список підтримуваних G-кодів та M-кодів з коротким описом їх синтаксису. Використання макросу F() дозволяє зберігати ці рядки у Flash-пам'яті мікроконтролера, оптимізуючи використання оперативної пам'яті.

```
/* --- Довідка --- */
void Help() {
    Serial.println(F("Довідка з команд G-коду:"));
    Serial.println(F("G00 [X] [Y] [Z] [F] - лінійний рух"));
    Serial.println(F("G01 [X] [Y] [Z] [F] - лінійний рух"));
    Serial.println(F("G02 [X] [Y] [Z] [I] [J] [F] - дуга за годинниковою"));
    Serial.println(F("G03 [X] [Y] [Z] [I] [J] [F] - дуга проти годинникової"));
    Serial.println(F("G04 P[сек] - пауза"));
    Serial.println(F("G90 - абсолютний режим"));
    Serial.println(F("G91 - відносний режим"));
    Serial.println(F("G92 [X] [Y] [Z] - задати позицію"));
    Serial.println(F("M18 - вимкнути приводи"));
    Serial.println(F("M100 - допомога"));
    Serial.println(F("M114 - показати поточну позицію"));
    Serial.println(F("Команди мають вводитися великими літерами"));
}
```

Рисунок 4.16. Код модуля для організації довідки.

Обробка спеціалізованих M-команд здійснюється функцією Handle_M_Command(int mcode). Вона виконує дії, специфічні для кожної M-команди: M18 викликає Disable() для вимкнення двигунів; M100 активує Help() для виведення довідки; M114 викликає Current_Position() для відображення поточної позиції. У випадку отримання нерозпізнаної M-команди, на послідовний порт виводиться відповідне повідомлення про помилку. Код функції обробки M-команд показаний на рис.18.

```
/* --- Обробка M-команд */
void Handle_M_Command(int mcode) {
    switch (mcode) {
        case 18:  // Вимкнути двигуни
            Disable();
            Serial.println(F("Приводи вимкнені"));
            break;
        case 100: // Довідка
            Help();
            break;
        case 114: // Показати позицію
            Current_Position();
            break;
        default:
            Serial.print(F("Невідома M-команда: "));
            Serial.println(mcode);
    }
}
```

Рисунок 4.17. Код функції обробки M-команд

4.6. Основна логіка керування та цикли виконання

Функція Execute_Command() що показана на рис. 19, є ключовим модулем, який інтерпретує та виконує отримані G-команди з Serial_Buffer. Вона спочатку намагається виділити числовий G-код за допомогою Parse_Number('G', -1). Логіка виконання команд структурована за допомогою оператора switch:

G00 та G01 (лінійний рух): Оновлюється швидкість подачі, парсяться координати X, Y, Z (з урахуванням абсолютного/відносного режиму), і потім викликається TD_Line() для фізичного переміщення.

G02 та G03 (рух по дузі): Оновлюється швидкість подачі, парсяться кінцеві координати X, Y, Z, а також параметри центру дуги I та J. Після цього викликається TD_Arc() з відповідним напрямком обертання.

G04 (пауза): Парситься параметр P (у секундах) і викликається Pause() для затримки виконання програми.

G90: Встановлює режим абсолютних координат (mode_abs = 1).

G91: Встановлює режим відносних координат (mode_abs = 0).

G92: Виконує команду встановлення поточної позиції без фізичного переміщення. Координати px, py, pz оновлюються відповідно до значень X, Y, Z, отриманих з команди.

Якщо G-код не знайдено (cmd == -1), функція перевіряє, чи починається команда з символу 'M'. У такому випадку вона намагається виділити M-код та передати його для обробки функції Handle_M_Command(). Якщо команда не відповідає жодному відомому формату, виводиться повідомлення про невідому команду. Будь-які інші невідомі G-команди обробляються через default гілку, виводячи повідомлення про помилку.

```
/* --- Головна функція виконання команди G-коду --- */
void Execute_Command() {
    int cmd = (int)Parse_Number('G', -1);  // Читаємо номер команди G

    switch (cmd) {
        case 0:
        case 1: {
            // Лінійний рух (G00 та G01)
            Feed_Rate(Parse_Number('F', fr));  // Оновлюємо швидкість (якщо є)
            float newX = Parse_Number('X', mode_abs ? px : 0) + (mode_abs ? 0 : px);
            float newY = Parse_Number('Y', mode_abs ? py : 0) + (mode_abs ? 0 : py);
            float newZ = Parse_Number('Z', mode_abs ? pz : 0) + (mode_abs ? 0 : pz);
            TD_Line(newX, newY, newZ);
            break;
        }
        case 2:
        case 3: {
            // Дуга (G02 - за годинниковою, G03 - проти)
            Feed_Rate(Parse_Number('F', fr));
            float cx = Parse_Number('I', mode_abs ? px : 0) + (mode_abs ? 0 : px);
            float cy = Parse_Number('J', mode_abs ? py : 0) + (mode_abs ? 0 : py);
            float newX = Parse_Number('X', mode_abs ? px : 0) + (mode_abs ? 0 : px);
            float newY = Parse_Number('Y', mode_abs ? py : 0) + (mode_abs ? 0 : py);
            float newZ = Parse_Number('Z', mode_abs ? pz : 0) + (mode_abs ? 0 : pz);
            TD_Arc(cx, cy, pz, newX, newY, newZ, (cmd == 2) ? -1 : 1);
            break;
        }
```

Рисунок 4.18. Фрагмент коду функції Execute_Command ().

Системна функція Ready(), що показана на рис.20 виконує підготовку буфера для прийому нових даних та сигналізує хост-пристрою про готовність контролера. Вона обнуляє лічильник символів symqnty, ефективно очищаючи

Serial_Buffer, та виводить символ > на послідовний порт, що є протокольним маркером готовності.

```
/**
* Підготуємо вхідний буфер для отримання нового повідомлення
* та повідомимо пристрій, який підключений через
56* послідовний порт, що він готовий
* до прийому наступного повідомлення.
**/
void Ready() {
symqnty=0; // очищуємо буфер
Serial.print(F(">")); // подаємо сигнал готовності до прийому
}
```

Рисунок 4.19. Фрагмент коду із функцією ready().

Ініціалізація програмного забезпечення здійснюється в системній функції setup(), яка виконується лише один раз при завантаженні мікроконтролера. Її послідовність дій включає: ініціалізацію послідовного порту (Serial.begin(BAUD_RATE)), конфігурацію виводів керування двигунами (Setup_Controller()), встановлення початкової позиції (Position(0,0)), встановлення швидкості подачі за замовчуванням (Feed_Rate((MAX_FEED_RATE + MIN_FEED_RATE)/2)), виведення довідкової інформації (Help()) та сигналізацію готовності до прийому першої команди (Ready()). Секцію із кодом функції показано на рис.21.

```
/**
* Початкові налаштування контролера
*/
void setup() {
Serial.begin(BAUD_RATE); // ініціалізація послідовного порту
Setup_Controller();
Position(0,0); // встановлюємо початкову позицію
Feed_Rate((MAX_FEED_RATE + MIN_FEED_RATE)/2); // Встановлюємо швидкість переміщення (подачі) за замовчуванням
Help(); // Привітаємось!:)
Ready();
}
/**
```

Рисунок 4.20. Фрагмент коду із реалізацією функції setup().

Безперервна робота контролера забезпечується в головному циклі loop(). Фрагмент з кодом функції показаний на рис.22. Ця функція постійно моніторить наявність вхідних даних у послідовному порту (Serial.available() > 0). При вияв-

ленні даних вона зчитує символи по одному (Serial.read()), відправляє їх назад як ехо-відповідь (Serial.print(c)), що є важливим для верифікації зв'язку. Прочитані символи накопичуються у Serial_Buffer до тих пір, поки не буде досягнуто максимального розміру буфера або не буде отриманий символ нового рядка (\n) або повернення каретки (\r), що сигналізує про завершення вхідної команди. Після отримання повної команди буфер завершується нульовим символом, виводиться новий рядок, і викликається Execute_Command() для її обробки. Після завершення виконання команди знову викликається Ready(), що завершує цикл обробки однієї команди та підготовлює контролер до наступної.

```
void loop() {
// Очікуємо команди на вході послідовного порту
while(Serial.available() > 0) { // якщо щось надійшло
char c=Serial.read(); // отримуємо
Serial.print(c); // Надсилаємо зворотнє ехо, слугує для перевірки обладнання
if(symqnty<MAX_BUFER-1) Serial_Buffer[symqnty++]=c; // зберігаємо отримане
if((c=='\n') || (c == '\r')) {
// вхідна команда отримана
Serial_Buffer[symqnty]=0;
Serial.print(F("\r\n"));
Execute_Command();
Ready();
} }}
```

Рисунок 4.21. Секція коду із реалізацією функції loop().

4.7. Тестування за допомогою он-лайн симулятора

Тестування програмного забезпечення будемо проводити в он_лайн симуляторі. Wokwi являє собою високофункціональний, повністю онлайновий симулятор, створений для спрощення та прискорення розробки проєктів на базі мікроконтролерів, таких як Arduino, ESP32 та Raspberry Pi Pico. Його основна ідея полягає у наданні розробникам і початківцям віртуального середовища, де можна швидко збирати електронні схеми, писати код (підтримується Arduino C/C++ та MicroPython) та миттєво запускати симуляцію, не потребуючи фізичного обладнання. Платформа працює безпосередньо у веббраузері, що усуває необхідність у

складному налаштуванні програмного забезпечення та драйверів, роблячи процес навчання та прототипування максимально доступним.

Ключові переваги Wokwi охоплюють декілька аспектів. Перш за все, це значна економія часу та коштів, оскільки користувачеві не потрібно купувати, чекати на доставку чи фізично підключати реальні компоненти. Це ідеальний інструмент для швидкого прототипування та тестування ідей, де можна без ризику «спалити» плату експериментувати з підключеннями та кодом. Крім того, Wokwi чудово підтримує віддалену роботу та навчання завдяки функції миттєвого поширення проєкту через унікальне посилання; це дозволяє викладачам легко ділитися завданнями, а учням — демонструвати результати. Для IoT-розробки важлива можливість симуляції мережевих підключень (HTTP, MQTT), що дозволяє тестувати хмарну логіку ще до перенесення коду на реальний пристрій, а наявність потужних інструментів, таких як віртуальний логічний аналізатор та інтеграція з GDB, задовольняє потреби навіть досвідчених інженерів у налагодженні.

Однак, як і будь-який симулятор, Wokwi має певні недоліки та обмеження, про які варто пам'ятати. Основний мінус полягає у тому, що симуляція ніколи не буде ідентична реальному світові; вона не враховує всіх нюансів фізичної електроніки, таких як шуми, точні часові затримки, проблеми з живленням або фізичні перешкоди, які можуть виникнути у реальній схемі. Крім того, не всі бібліотеки та рідкісні компоненти можуть бути підтримані симулятором, що іноді змушує досвідчених розробників звертатися до фізичного тестування. Хоча більшість базових функцій Wokwi доступна безкоштовно, деякі професійні можливості (наприклад, розширена підтримка налагодження чи приватні шлюзи для IoT) можуть вимагати платної ліцензії. Утім, ці обмеження є типовими для онлайн-симуляторів і не зменшують його цінності як навчального та прототипного інструменту.

Загальний вигляд інтерфейса і симуляції показаний на рис.23

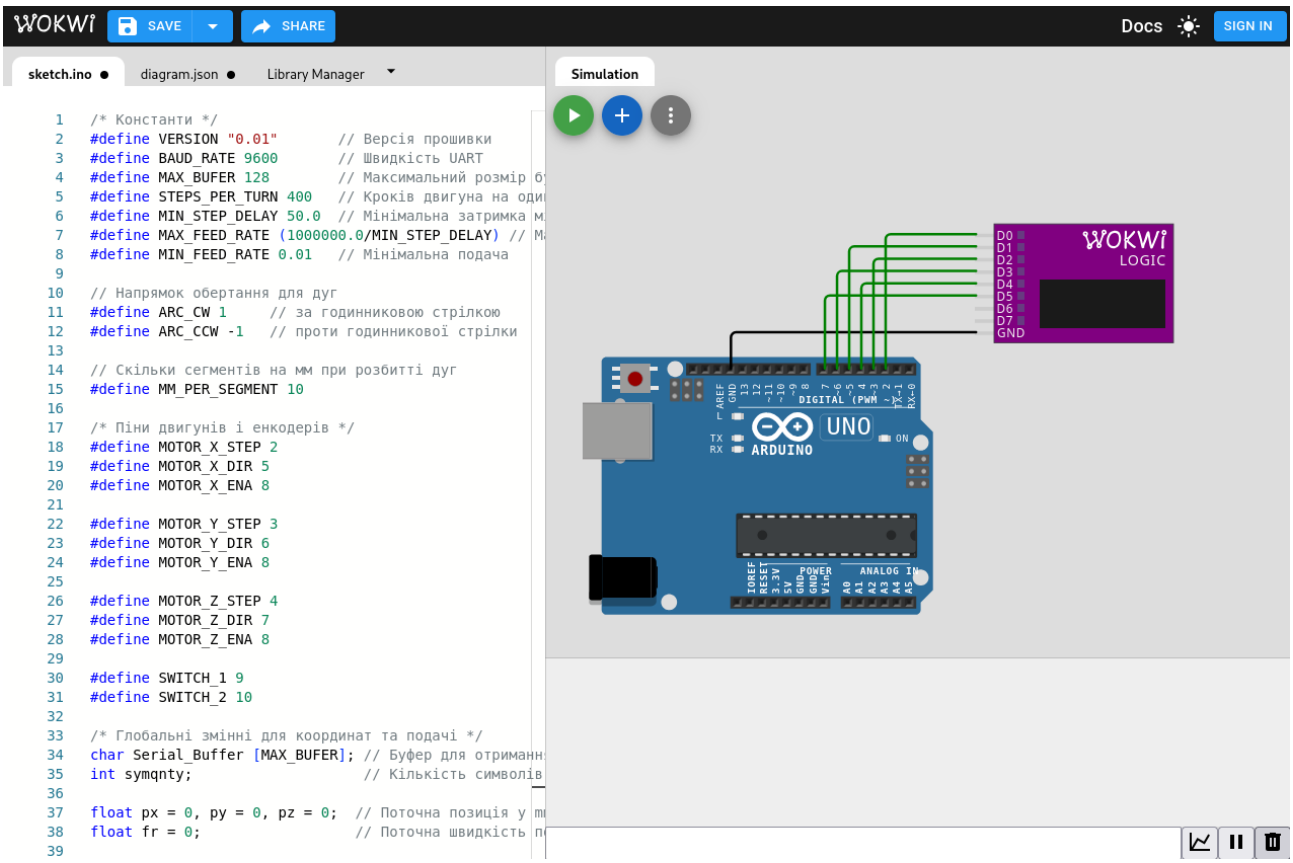


Рисунок 4.22. Загальний вигляд он-лайн симулятора wokwi.

Вбудований Логічний аналізатор є одним із найпотужніших інструментів симулятора Wokwi, який виводить процес налагодження мікроконтролерних проєктів на новий рівень. Він дозволяє користувачу віртуально підключатися до будь-якого цифрового виводу (піна) мікроконтролера чи компонента у схемі та записувати послідовність керуючих сигналів. Результатом роботи аналізатора є файл, який можна відкрити у стандартних програмах для аналізу цифрових сигналів, наприклад PulseView. Це дає можливість детально вивчати протоколи обміну даними (такі як I2C, SPI, UART), відстежувати точні часові затримки та перевіряти коректність формування імпульсів, що є критично важливим для роботи з датчиками, дисплеями та іншими периферійними пристроями.

Ключова перевага використання Логічного аналізатора полягає в тому, що для дослідження та перевірки керуючих сигналів часто не обов'язково симулювати всю складну периферійну схему. Наприклад, якщо розробнику потрібно переконатися, що мікроконтролер ESP32 правильно генерує команду для включення реле

або відправляє коректний байт даних через шину I2C, достатньо просто "підключити" аналізатор до відповідного виводу. Таким чином, можна ізолювати програмну логіку від апаратної частини, швидко перевіряючи епюри (часові діаграми) сигналів прямо на виводах мікроконтролера. Цей підхід економить ресурси симулятора, значно прискорює процес налагодження та дозволяє розробнику зосередитися виключно на коректності програмної реалізації протоколів зв'язку. Схема підключення логічного аналізатора показана на рис.24.

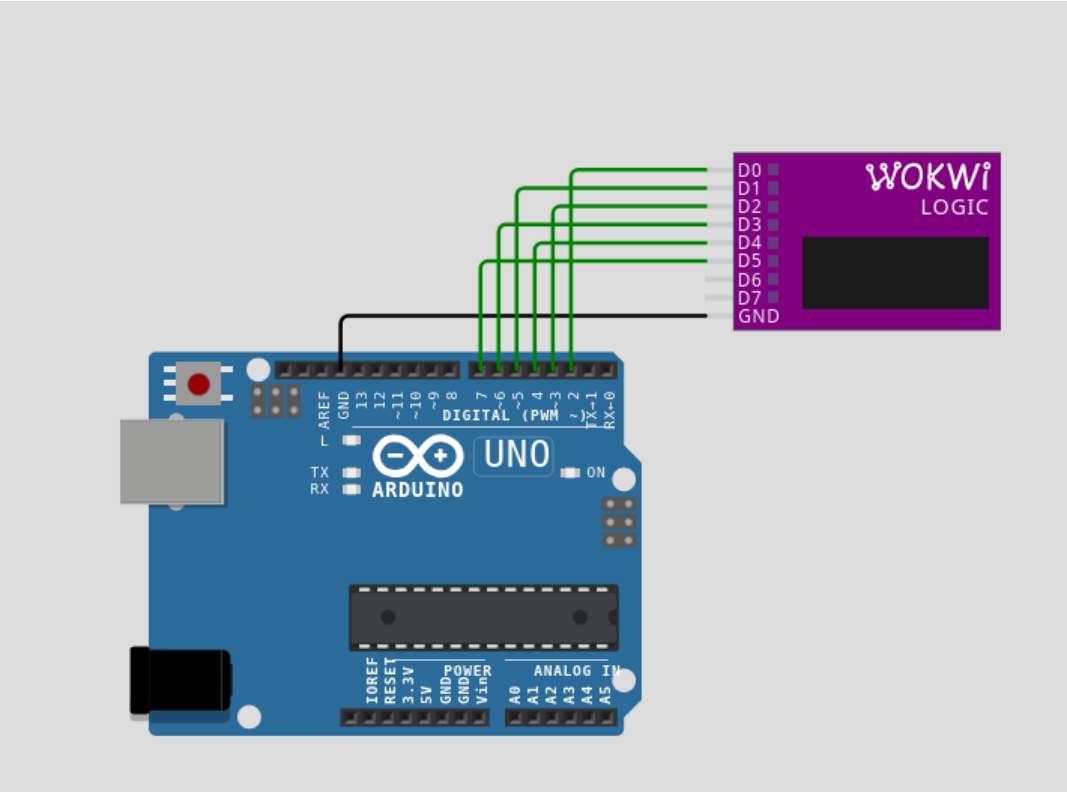
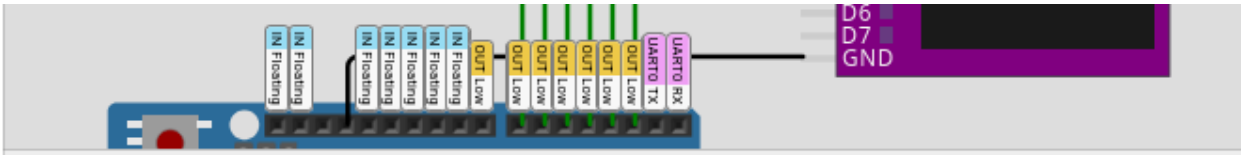


Рисунок 4.23. Віртуальна схема підключення логічного аналізатора.

Для запуску симуляції копіюємо код у поле для програм, що знаходиться у лівій частині інтерфейсу, при компіляції коду у симуляторі необхідні бібліотеки будуть підключені автоматично. Після запуску симуляції у правій нижній частині буде показаний вивід термінала, що його показано на рис.25.




```
Довідка з команд G-коду:
G00 [X] [Y] [Z] [F] - лінійний рух
G01 [X] [Y] [Z] [F] - лінійний рух
G02 [X] [Y] [Z] [I] [J] [F] - дуга за годинниковою
G03 [X] [Y] [Z] [I] [J] [F] - дуга проти годинникової
G04 P[сек] - пауза
G90 - абсолютний режим
G91 - відносний режим
G92 [X] [Y] [Z] - задати позицію
M18 - вимкнути приводи
M100 - допомога
M114 - показати поточну позицію
Команди мають вводитися великими літерами
>
```

Рисунок 4.24. Вивід довідки про підтримувані команди у термінал.

Введемо команду G00 X5 Y5 Z5 як показано на рис.26, команда G0 відповідає за переміщення верстата у задані координати без застосування інструменту.

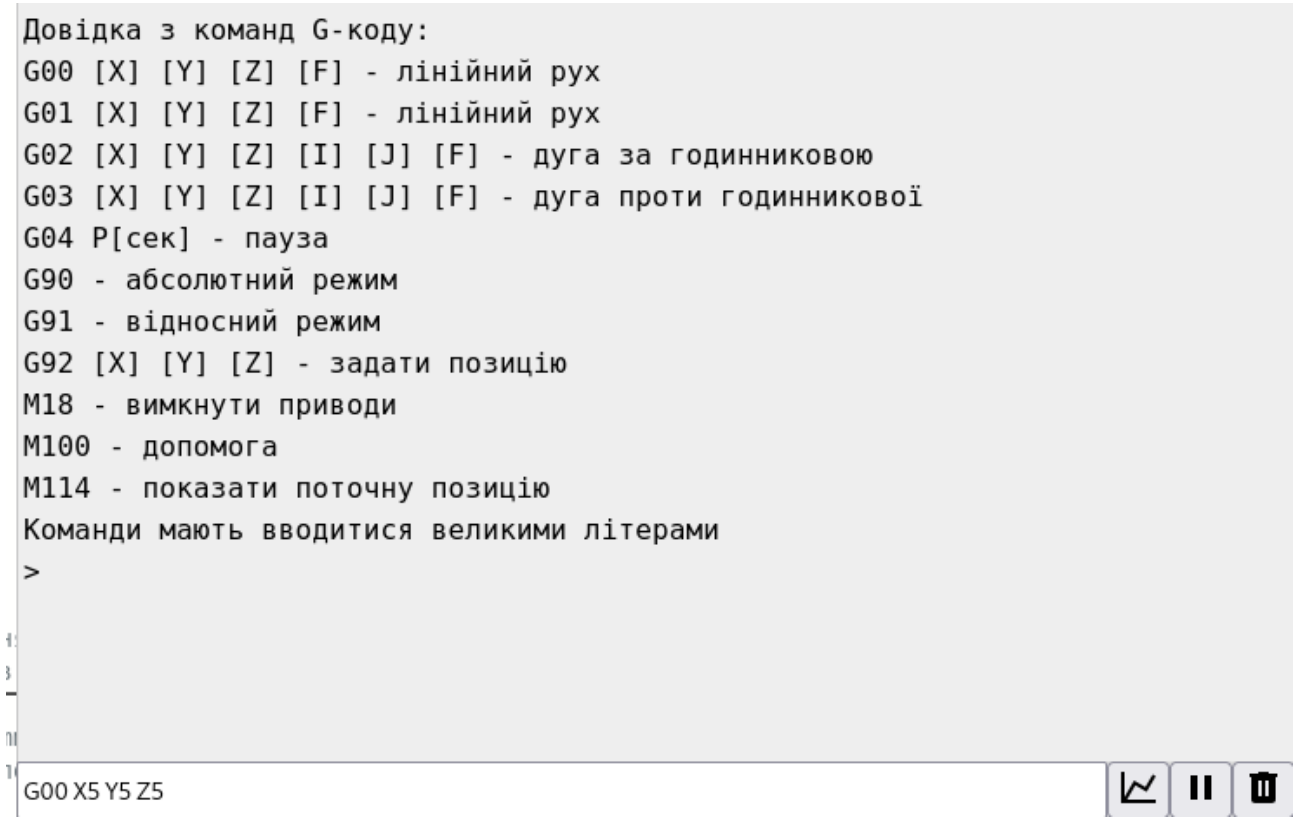


Рисунок 4.25. Ввід команди у терміналі симулятора.

Після виконання команди зупинимо симуляцію. Он-лайн симулятор надішле нам файл із даними, що їх було зібрано за допомогою логічного аналізатора. Дані зберігаються у форматі VCD. Формат Value Change Dump (VCD) є ключовим елементом, який забезпечує функціональність Логічного аналізатора в Wokwi та в більшості інструментів для симуляції цифрової логіки. VCD — це стандартизований, текстовий (ASCII) формат файлу, який використовується для запису змін значень вибраних сигналів у проєкті протягом часу симуляції. Назва "Dump" (скидання, звалище) вказує на те, що файл реєструє подію лише тоді, коли значення сигналу змінюється (наприклад, з 0 на 1 або навпаки), що робить його дуже компактним і ефективним для фіксації логічної активності. Файл VCD містить заголовок з метаданими, визначення змінних (їхні імена та ідентифікатори) і, власне, часові мітки разом із новими значеннями сигналів, дозволяючи точно відтворити поведінку цифрової схеми після завершення симуляції.

Після запуску симуляції з підключеним Логічним аналізатором, результати записуються саме у формат VCD. Цей універсальний формат дозволяє перенести дані з вебсимулятора у потужні зовнішні програми для аналізу осцилограм, такі як PulseView або інші інструменти для вивчення логічних схем. У цих програмах текстова інформація з VCD перетворюється на інтуїтивно зрозумілі графічні часові діаграми (епюри), що відображають зміни станів сигналів із високою роздільною здатністю. Таким чином, VCD слугує мостом між симулятором і професійними аналітичними засобами, надаючи розробнику можливість проводити глибокий, детальний та точний аналіз комунікаційних протоколів і синхронізації між компонентами віртуальної схеми. Для перегляду даних у графічному вигляді будемо використовувати програму GTKWave. Програма GTKWave є де-факто стандартом серед інженерів та розробників вбудованих систем для візуалізації та аналізу даних, збережених у форматі VCD (Value Change Dump). Це безкоштовний, кросплатформний переглядач осцилограм з відкритим вихідним кодом, розроблений спеціально для роботи з файлами, що генеруються симуляторами цифрової електроніки, включаючи Wokwi. Основна функція GTKWave — перетворити сирі, текстові дані з VCD-файлу, що містять часові мітки та зміни логічних станів, на зручні для читання та інтерпретації часові діаграми. Завдяки інтуїтивно зрозумілому графічному інтерфейсу, користувач може легко масштабувати графіки, переходити до конкретних часових інтервалів, групувати пов'язані сигнали та швидко ідентифікувати проблеми в синхронізації або логіці роботи протоколів, таких як SPI, I2C або UART.

Таким чином, GTKWave утворює логічний ланцюжок з Wokwi, забезпечуючи повноцінний цикл тестування та налагодження. Якщо Логічний аналізатор Wokwi фіксує послідовність керуючих сигналів, то GTKWave служить потужним мікроскопом для їхнього вивчення, надаючи розробнику глибинний інструментарій для пост-симуляційного аналізу. Користувач Wokwi завантажує VCD-файл, отриманий під час віртуальної роботи мікроконтролера, у GTKWave, де може візу-

ально порівнювати сигнали, визначати точні тривалості імпульсів і, що найважливіше, підтверджувати, що його програмна логіка генерує саме ті цифрові послідовності, які очікуються для коректної роботи зовнішньої периферії. Це робить GTKWave незамінним інструментом для точного верифікації роботи коду до його завантаження на фізичний пристрій. Загальний вигляд програми GTKWave показаниць на рис.26

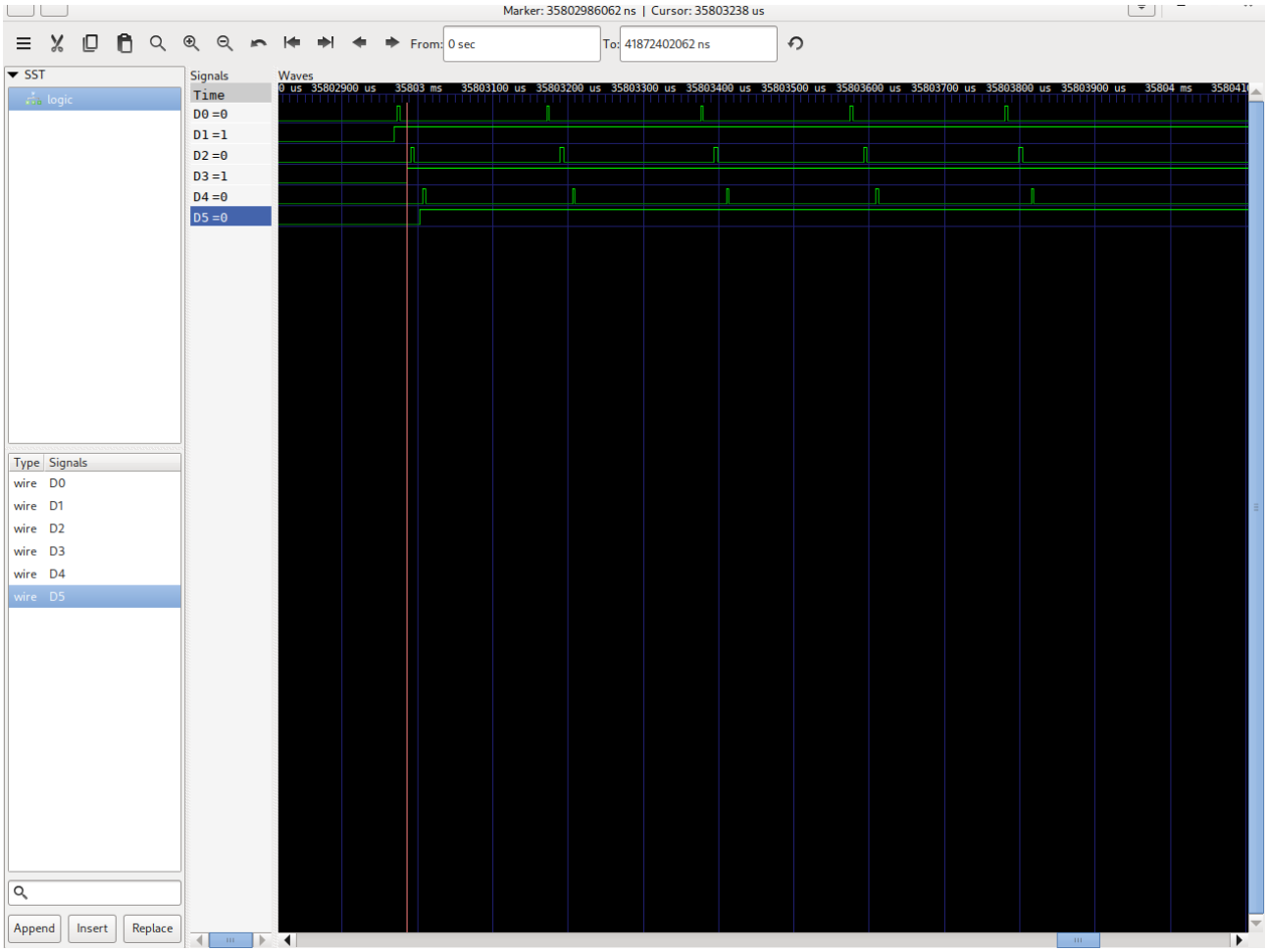


Рисунок 4.26 Загальний вигляд інтерфейса програми GTKWave.

Епюри сигналів управління кроковими рушіями під час симуляції показані на рис.27. Сигнали D0, D2, D4 відповідають сигналам на виводах мікроконтролера 2,3,4 відповідно описаними у макросах (рис.3). На цих виводах генеруються імпульси кроків. В свою чергу сигнали D1, D3, D5 відповідають сигналам на виводах 5,6,7 і генерують сигнали напрямку руху. Епюри сигналів на виводах мікрокконтролера покахано на рис.27.

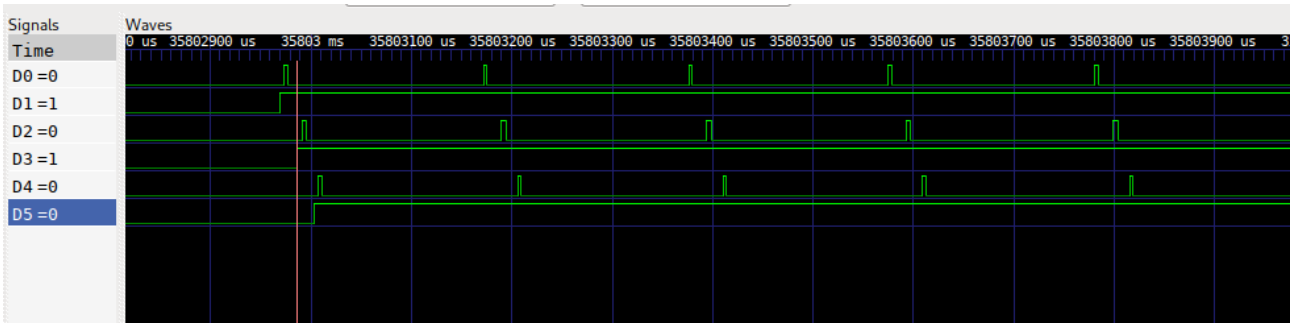


Рисунок 4.27. Епюри сигналів на виводах мікроконтролера.

Оскільки була введена команда для переміщення верстата на 5 одиниць по трьом осям, ми спостерігаємо по 5 імпульсів кроку для кожної вісі X,Y,Z. З чого можна зробити висновок що програмне забезпечення коректно аналізує команди і інтерпретує їх.

4.8. Висновки до розділу

Програмне забезпечення контролера реалізовано як модульна та високоефективна система, призначена для точної інтерпретації G/M-кодів та керування тривісним верстатом. В основі архітектури лежить модульний принцип, де функції, такі як Motor_X_Step() та Feed_Rate(), забезпечують дискретне керування двигунами та динамічне регулювання швидкості, а структурована конфігурація через константи підвищує гнучкість системи. Центральним елементом є функція Execute_Command(), яка забезпечує гнучку інтерпретацію вхідних команд. Для планування траєкторій використовується вбудований математичний апарат: цілочисельні алгоритми, зокрема адаптований 3D-алгоритм Брезенхема у функції TD_Line(), забезпечують точне лінійне переміщення, тоді як дуговий рух у TD_Arc() реалізується через розбиття на дрібні лінійні сегменти, що підтверджує надійність інтеграції програмної та математичної логіки.

Для верифікації та налагодження програмної логіки був застосований онлайн-симулятор Wokwi. Незважаючи на те, що симулятор не є досконалим та не враховує всіх фізичних нюансів, він надав критично важливу інженерну можливість для швидкої оцінки та підтвердження прийнятих архітектурних рішень. Го-

ловним інструментом верифікації став Віртуальний Логічний Аналізатор, який фіксував керуючі сигнали у форматі VCD. Подальший аналіз VCD-файлів за допомогою професійної програми GTKWave підтвердив коректність роботи коду. Зокрема, було зафіксовано, що введення команди G00 X5 Y5 Z5 викликало 5 синхронізованих імпульсів кроку на виводах кожної осі, що безпосередньо доводить правильність логіки інтерпретації команд та точність генерації керуючих сигналів.

Таким чином, реалізоване програмне забезпечення демонструє високий рівень інженерної продуманості, поєднуючи ефективні математичні алгоритми, модульну структуру керування та надійну систему інтерпретації G/M-команд. Методологія тестування, що включає використання Wokwi як симуляційного середовища та GTKWave для глибокого аналізу сигналів, підтверджує, що програмний код є коректним, функціональним та придатним для надійного керування механікою верстата.

РОЗДІЛ 5. РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ

5.1. Опис ідеї проєкту

Сучасний світ пропонує безмежні можливості для реалізації творчих ідей та підприємницьких починань. Часто особисте захоплення чи хобі, особливо у технічній сфері, переростає у щось більше – постійну діяльність або навіть повноцінний малий бізнес. У такі моменти виникає гостра потреба в якісному, але доступному обладнанні.

Наш стартап-проєкт "Майстерня Майбутнього" має на меті демократизувати доступ до передових технологій 3D-друку та обробки матеріалів за допомогою ЧПК (числового програмного керування). Ми розробляємо інноваційні, прості у збірці та економічно вигідні набори для конструювання власних верстатів із ЧПК та 3D-принтерів. Це дозволить кожному, хто має ідею – чи то майстер-ремісник, дизайнер, інженер-початківець або власник малого виробництва – без зайвих труднощів створити інструмент для втілення своїх задумів. Показане нами програмне забезпечення стане основою для створення верстатів, здатних реалізувати будь-які бажання та проєкти.

5.2. Актуальні тенденції на ринку 3D-друку

Ринок 3D-друку, або адитивного виробництва, переживає експоненційний ріст. За даними Market Research Future (MRFR) та інших авторитетних джерел, прогнозується, що середньорічний темп зростання (CAGR) складе від 18.5% до 23.5% до 2030-2034 років. Це значно вищі темпи зростання, ніж для ринку верстатів із ЧПК, що безумовно підкреслює величезну перспективність інтеграції 3D-друку до проєкту.

Важливо відзначити, що апаратна частина, тобто самі 3D-друкарки, займає значну частку цього зростаючого ринку. Хоча промислові 3D-друкарки генерують основний дохід, сегмент настільних (desktop) 3D-друкарок демонструє найшвидший CAGR, досягаючи 21-25% щорічно. Це є ключовим моментом для ідеї "наборів для збірки" та концепції "низького порогу входу", оскільки зростаюча доступ-

ність та простота використання цих друкарок є визначальним трендом для невеликих бізнесів та ентузіастів-хобістів.

Крім того, попри відсутність великої кількості прямих звітів MRFR саме щодо "DIY 3D printer kits", активні обговорення в спільнотах 3D-друку чітко показують, що інтерес до DIY-збірок, таких як Prusa або Voron, зберігається. Це обумовлено можливостями кастомізації, апгрейду та глибшого розуміння технології, що чудово резонує з вашою ідеєю. Постійний розвиток нових матеріалів, включно з полімерами, металами та композитами, також значно розширює можливості 3D-друку, роблячи його привабливим для все більшої кількості застосувань. Нарешті, варто підкреслити, що 3D-друк все частіше використовується не лише для швидкого прототипування, а й для безпосереднього створення функціональних кінцевих деталей та дрібносерійного виробництва, що відкриває нові горизонти для бізнес-моделей.

Швидкий розвиток індустріалізації та національні програми розвитку у багатьох країнах забезпечують сприятливе середовище для зростання промисловості фрезерних верстатів із ЧПУ. Це не тільки підвищує заробітну плату фахівців у цій галузі, але й створює нові робочі місця. Отже, інвестування у цей напрямок є стратегічно вигідним.

5.3. Тенденції та високий ступінь автоматизації

Попит на автоматизовані робочі процеси стрімко зростає, особливо у країнах, що розвиваються, що відкриває нові перспективи для виробників верстатів із ЧПК. Пандемія COVID-19 яскраво продемонструвала критичну важливість основного обладнання, а виробники ЧПК є невід’ємною частиною базових послуг. Компанії промислового дизайну все більше покладаються на верстати з ЧПК для виготовлення модульних кухонь, меблів, а також освітнього та розважального обладнання.

Високоавтоматизований режим обробки фрезерного верстата з ЧПК забезпечує стабільну та безперебійну роботу навіть в умовах нестабільності, подібної до пандемії, що дозволяє підприємствам продовжувати генерувати прибуток.

5.4. Переваги розробки: доступність та ефективність

Ключові переваги нашого стартапу полягають у:

Низькій собівартості та низькому порогу входу: Ми пропонуємо доступні комплекти для збірки верстатів із ЧПК та 3D-принтерів, що робить їх привабливими як для хобі-проєктів, так і для малого бізнесу чи навіть невеликих виробництв.

Пришвидшеному запуску: Реалізація попередньо запрограмованого контролера значно спрощує та прискорює введення верстата в експлуатацію, мінімізуючи час на налаштування та навчання.

5.5. Висновки до розділу

У розділі 5 розглянуто концепцію стартап-проєкту "Майстерня Майбутнього", який має на меті демократизувати доступ до технологій числового програмного керування (ЧПК) та 3D-друку. Ідея полягає у розробці та просуванні інноваційних, економічно вигідних наборів (DIY-кітів) для конструювання власних верстатів із ЧПК та 3D-принтерів. Це дозволить майстрам, дизайнерам, інженерам-початківцям та малому бізнесу отримати необхідне обладнання для реалізації своїх творчих ідей та виробничих потреб, використовуючи, зокрема, програмне забезпечення, розроблене в попередніх розділах.

Актуальність проєкту підтверджується експоненційним ростом ринку адитивного виробництва, де середньорічний темп зростання (CAGR) прогнозується на рівні 18.5%–23.5% до 2030–2034 років. При цьому сегмент настільних 3D-друкарок демонструє найшвидший ріст, що ідеально узгоджується з концепцією "наборів для збірки" та низького порогу входу. Збереження високого інтересу до DIY-збірок (як-от Prusa та Voron) підтверджує ринковий попит на кастомізацію та глибоке розуміння технології.

Ключові переваги стартапу полягають у забезпеченні низької собівартості та доступності обладнання. Особливий акцент робиться на пришвидшеному запуску завдяки реалізації попередньо запрограмованого контролера. Таким чином, "Майстерня Майбутнього" позиціонується як стратегічно вигідний проєкт, який відпо-

відає сучасним ринковим трендам, пропонуючи ефективне та доступне рішення для входження у сферу цифрового виробництва.

ВИСНОВКИ

У межах виконаної роботи було реалізовано комплексний підхід дослідження та проєктування систем 3D-друку, що охопив усі ключові етапи — від теоретичного вивчення до розробки програмного забезпечення та закладення основ для комерціалізації. На початковому етапі було ґрунтовно розглянуто різні технології та методи 3D-друку, що дозволило визначити оптимальні інженерні рішення для подальшої розробки. Серцем проєкту стала програмна реалізація, зокрема написання власного забезпечення, призначеного для аналізу та безпосереднього виконання G-коду. Ця частина роботи вимагала детального опрацювання математичного апарату, а саме: було успішно реалізовано алгоритм Брезенгема для трьох осей, що є фундаментальною основою для точного позиціонування друкувальної головки. Для ефективної розробки та налагодження програмного коду використовувалися відкриті та доступні платформи, зокрема інтегроване середовище Arduino IDE, а також потужний онлайн-симулятор Wokwi, який дозволив проводити віртуальне тестування логіки мікроконтролера. Більш того, для глибокого аналізу керуючих сигналів, отриманих під час симуляції, застосовувалося вільне програмне забезпечення GTKWave, що дало змогу візуалізувати часові діаграми (епюри) і верифікувати коректність реалізації протоколів обміну даними.

Отримані результати демонструють високий рівень інтеграції між апаратною логікою, математичною моделлю та програмним забезпеченням. Успішна реалізація ключових функціональних блоків, від алгоритму Брезенгема до власного парсера G-коду, підтверджує можливість створення повністю функціональної та керованої системи 3D-друку на основі вільно доступних інструментів. При цьому, обрана методологія з використанням Wokwi та GTKWave значно мінімізує ризики, пов'язані з фізичним налагодженням, та прискорює цикл розробки. З огляду на глибину опрацювання технічних рішень і використання відкритого програмного забезпечення, що знижує вхідні бар'єри, у роботі було закладено міцні основи для подальшого розвитку проєкту у форматі стартапу.

Таким чином, цей проєкт є не лише успішним технічним дослідженням, але й має високий потенціал для комерціалізації як інноваційне та доступне рішення у сфері адитивних технологій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. http://www.cg.tuwien.ac.at/courses/cg2 – сайт Інституту комп'ютерної графіки і алгоритмів Віденського технічного університету.
2. http://graphics.cs.ucdavis.edu – сайт з КГ інституту аналізу даних і візуалізації Каліфорнійського університету.
3. Анісімов В.А., Терещенко В.М., Кравченко І.В. Основні алгоритми обчислювальної геометрії: Навч. посібн. – К.: Київський університет, 2002.
4. GD&T: Application and Interpretation Seventh Edition, Revised, by Bruce A. Wilson, November 1, 2019
5. CNC Programming Handbook, Third Edition by Peter Smid, November 26, 2007 Machining Fundamentals Tenth Edition, by John R. Walker , Bob Dixon. January 25, 2018
6. Петренко В. П. Числове програмне керування верстатами: Підручник. — Київ: Кондор, 2016.
7. Гриньов В. М. Програмування верстатів з числовим програмним керуванням. — Харків: НТУ "ХПІ", 2018.
8. Peter Smid — CNC Programming Handbook. — Industrial Press, 3rd Edition, 2007.
9. Wokwi. Welcome to Wokwi! URL: https://docs.wokwi.com/. Загальне джерело
10. Іванченко С. В. Основи програмування на C++: Підручник. — Львів: Новий Світ, 2020.
11. Гончаренко Т. Ю. Алгоритми та структури даних у C++: Навчальний посібник. — Київ: Либідь, 2019.
12. Мельник А. І. Системне програмування на C: Підручник. — Одеса: Астропринт, 2017.
13. Романенко В. С. Об'єктно-орієнтоване програмування на C++: Навчальний посібник. — Дніпро: Універсум, 2021.
14. Петренко В. П. Числове програмне керування верстатами: Підручник. — Київ: Кондор, 2016.

15. Шевченко Н. М. Практикум з програмування мовами С та С++: Навчальний посібник. — Чернівці: Букрек, 2022.

ДОДАТКИ

```
/* Константи */
#define VERSION "0.01"       // Версія прошивки
#define BAUD_RATE 9600       // Швидкість UART
#define MAX_BUFER 128        // Максимальний розмір буфера команд
#define STEPS_PER_TURN 400   // Кроків двигуна на один повний оберт
#define MIN_STEP_DELAY 50.0  // Мінімальна затримка між кроками (мкс)
#define MAX_FEED_RATE (1000000.0/MIN_STEP_DELAY) // Максимальна подача (швидкість руху)
#define MIN_FEED_RATE 0.01   // Мінімальна подача

// Напрямок обертання для дуг
#define ARC_CW 1     // за годинниковою стрілкою
#define ARC_CCW -1   // проти годинникової стрілки

// Скільки сегментів на мм при розбитті дуг
#define MM_PER_SEGMENT 10

/* Піни двигунів і енкодерів */
#define MOTOR_X_STEP 2
#define MOTOR_X_DIR 5
#define MOTOR_X_ENA 8

#define MOTOR_Y_STEP 3
#define MOTOR_Y_DIR 6
#define MOTOR_Y_ENA 8

#define MOTOR_Z_STEP 4
#define MOTOR_Z_DIR 7
#define MOTOR_Z_ENA 8

#define SWITCH_1 9
#define SWITCH_2 10
```

```
/* Глобальні змінні для координат та подачі */
char Serial_Buffer [MAX_BUFER]; // Буфер для отримання команд
int symqnty;                    // Кількість символів в буфері

float px = 0, py = 0, pz = 0;  // Поточна позиція у mm
float fr = 0;                  // Поточна швидкість подачі в кроках/сек

long step_delay;               // Затримка між кроками (мкс)
char mode_abs=1;               // Режим координат: 1 - абсолютний, 0 - відносний

/* --- Функції керування двигунами --- */

/* Зробити один крок по осі X з напрямком dir (1 чи -1) */
void Motor_X_Step(int dir) {
   digitalWrite(MOTOR_X_ENA, LOW); // Вмикаємо двигун
   digitalWrite(MOTOR_X_DIR, dir == 1 ? HIGH : LOW); // Встановлюємо напрямок
   digitalWrite(MOTOR_X_STEP, HIGH); // Сигнал кроку
   digitalWrite(MOTOR_X_STEP, LOW);
}

/* Зробити один крок по осі Y */
void Motor_Y_Step(int dir) {
   digitalWrite(MOTOR_Y_ENA, LOW);
   digitalWrite(MOTOR_Y_DIR, dir == 1 ? HIGH : LOW);
   digitalWrite(MOTOR_Y_STEP, HIGH);
   digitalWrite(MOTOR_Y_STEP, LOW);
}

/* Зробити один крок по осі Z */
void Motor_Z_Step(int dir) {
   digitalWrite(MOTOR_Z_ENA, LOW);
   digitalWrite(MOTOR_Z_DIR, dir == 1 ? HIGH : LOW);
   digitalWrite(MOTOR_Z_STEP, HIGH);
   digitalWrite(MOTOR_Z_STEP, LOW);
}
```

```
/* Вимкнути усі двигуни (для безпеки чи збереження енергії) */
void Disable() {
   digitalWrite(MOTOR_X_ENA, HIGH);
   digitalWrite(MOTOR_Y_ENA, HIGH);
   digitalWrite(MOTOR_Z_ENA, HIGH);
}

/* Налаштування пінів Arduino */
void Setup_Controller() {
   pinMode(MOTOR_X_ENA, OUTPUT);
   pinMode(MOTOR_Y_ENA, OUTPUT);
   pinMode(MOTOR_Z_ENA, OUTPUT);
   pinMode(MOTOR_X_STEP, OUTPUT);
   pinMode(MOTOR_Y_STEP, OUTPUT);
   pinMode(MOTOR_Z_STEP, OUTPUT);
   pinMode(MOTOR_X_DIR, OUTPUT);
   pinMode(MOTOR_Y_DIR, OUTPUT);
   pinMode(MOTOR_Z_DIR, OUTPUT);
}

/* Затримка з поділом на мс та мкс */
void Pause(long us) {
   delay(us / 1000);          // мілісекунди
   delayMicroseconds(us % 1000); // мікросекунди
}

/* Встановлення швидкості подачі (feed rate)
  nfr - швидкість у кроках/сек */
void Feed_Rate(float nfr) {
   if (fr == nfr) return; // Якщо не змінилось - не змінюємо
   if (nfr > MAX_FEED_RATE || nfr < MIN_FEED_RATE) {
      Serial.print(F("Помилка: швидкість поза межами: "));
      Serial.println(nfr);
      return;
```

```
    }
    step_delay = 1000000.0 / nfr; // Переводимо швидкість у затримку між кроками (мкс)
    fr = nfr;
}

/* Встановлення нової позиції */
void Position(float npx, float npy, float npz=0) {
    px = npx;
    py = npy;
    pz = npz;
}

/* --- Алгоритм Брезенхема для лінійного руху в 3D --- */
/* Рух з поточної позиції (px, py, pz) до (newx, newy, newz) */
/* По кроках робимо рух по трьох осях одночасно */
void TD_Line(float newx, float newy, float newz) {
    long i;
    long dx = abs(newx - px);
    long dy = abs(newy - py);
    long dz = abs(newz - pz);

    int dirx = (newx > px) ? 1 : -1;
    int diry = (newy > py) ? 1 : -1;
    int dirz = (newz > pz) ? 1 : -1;

    long max_delta = max(dx, max(dy, dz)); // Визначаємо головну вісь руху
    long over_x = max_delta / 2;
    long over_y = max_delta / 2;
    long over_z = max_delta / 2;

    for (i = 0; i < max_delta; i++) {
        over_x += dx;
        over_y += dy;
        over_z += dz;
```

```
        if (over_x >= max_delta) {
            over_x -= max_delta;
            Motor_X_Step(dirx);
        }
        if (over_y >= max_delta) {
            over_y -= max_delta;
            Motor_Y_Step(diry);
        }
        if (over_z >= max_delta) {
            over_z -= max_delta;
            Motor_Z_Step(dirz);
        }
        Pause(step_delay);
    }
    // Оновлюємо координати позиції після руху
    px = newx;
    py = newy;
    pz = newz;
}

/* Функція, що повертає кут у діапазоні 0...2*PI (0 - 360 градусів) */
float atan3(float dy, float dx) {
    float a = atan2(dy, dx);
    if (a < 0) a += 2.0 * PI;
    return a;
}

/* --- Рух по дузі у площині XY з інтерполяцією Z --- */
/* cx, cy - центр дуги */
/* startz, endz - початковий і кінцевий Z */
/* x, y, z - кінцева точка */
/* dir - напрямок руху: ARC_CW або ARC_CCW */
void TD_Arc(float cx, float cy, float startz, float x, float y, float endz, float dir) {
    float dx = px - cx;
    float dy = py - cy;
```

```
    float radius = sqrt(dx*dx + dy*dy);

    float angle1 = atan3(dy, dx);
    float angle2 = atan3(y - cy, x - cx);
    float theta = angle2 - angle1;

    // Корекція кута для правильного напряму руху
    if (dir > 0 && theta < 0) angle2 += 2 * PI;
    else if (dir < 0 && theta > 0) angle1 += 2 * PI;
    theta = angle2 - angle1;

    float arc_length = abs(theta) * radius;
    int segments = ceil(arc_length * MM_PER_SEGMENT);

    for (int i = 0; i < segments; ++i) {
        float scale = (float)i / segments;
        float angle = angle1 + theta * scale;
        float nx = cx + cos(angle) * radius;
        float ny = cy + sin(angle) * radius;
        float nz = startz + (endz - startz) * scale; // Інтерполяція Z

        TD_Line(nx, ny, nz); // Рух по сегменту дуги
    }
    // Заключний рух до кінцевої точки дуги
    TD_Line(x, y, endz);
}

/* --- Функція парсингу числового значення по коду (X, Y, Z, F, I, J тощо) --- */
/* Якщо не знайдено відповідного коду, повертає val (поточне значення) */
float Parse_Number(char code, float val) {
    char *ptr = Serial_Buffer;
    while ((long)ptr > 1 && (*ptr) && (long)ptr < (long)Serial_Buffer + symqnty) {
        if (*ptr == code) {
            return atof(ptr + 1);
        }
```

```
        ptr = strchr(ptr, ' ');
        if (!ptr) break;
        ptr++;
    }
    return val;
}

/* --- Вивід позиції або повідомлень --- */
void OutPut(const char *code, float val) {
    Serial.print(code);
    Serial.println(val);
}

/* Вивід поточної позиції та подачі */
void Current_Position() {
    OutPut("X", px);
    OutPut("Y", py);
    OutPut("Z", pz);
    OutPut("F", fr);
    Serial.println(mode_abs ? "Режим: Абсолютний" : "Режим: Відносний");
}

/* --- Довідка --- */
void Help() {
    Serial.println(F("Довідка з команд G-коду:"));
    Serial.println(F("G00 [X] [Y] [Z] [F] - лінійний рух"));
    Serial.println(F("G01 [X] [Y] [Z] [F] - лінійний рух"));
    Serial.println(F("G02 [X] [Y] [Z] [I] [J] [F] - дуга за годинниковою"));
    Serial.println(F("G03 [X] [Y] [Z] [I] [J] [F] - дуга проти годинникової"));
    Serial.println(F("G04 P[сек] - пауза"));
    Serial.println(F("G90 - абсолютний режим"));
    Serial.println(F("G91 - відносний режим"));
    Serial.println(F("G92 [X] [Y] [Z] - задати позицію"));
    Serial.println(F("M18 - вимкнути приводи"));
    Serial.println(F("M100 - допомога"));
```

```
    Serial.println(F("M114 - показати поточну позицію"));
    Serial.println(F("Команди мають вводитися великими літерами"));
}

/* --- Обробка M-команд */
void Handle_M_Command(int mcode) {
    switch (mcode) {
        case 18:  // Вимкнути двигуни
            Disable();
            Serial.println(F("Приводи вимкнені"));
            break;
        case 100: // Довідка
            Help();
            break;
        case 114: // Показати позицію
            Current_Position();
            break;
        default:
            Serial.print(F("Невідома M-команда: "));
            Serial.println(mcode);
    }
}

/* --- Головна функція виконання команди G-коду --- */
void Execute_Command() {
    int cmd = (int)Parse_Number('G', -1);  // Читаємо номер команди G

    switch (cmd) {
        case 0:
        case 1: {
            // Лінійний рух (G00 та G01)
            Feed_Rate(Parse_Number('F', fr));  // Оновлюємо швидкість (якщо є)
            float newX = Parse_Number('X', mode_abs ? px : 0) + (mode_abs ? 0 : px);
            float newY = Parse_Number('Y', mode_abs ? py : 0) + (mode_abs ? 0 : py);
            float newZ = Parse_Number('Z', mode_abs ? pz : 0) + (mode_abs ? 0 : pz);
```

```
        TD_Line(newX, newY, newZ);
        break;
    }
    case 2:
    case 3: {
        // Дуга (G02 - за годинниковою, G03 - проти)
        Feed_Rate(Parse_Number('F', fr));
        float cx = Parse_Number('I', mode_abs ? px : 0) + (mode_abs ? 0 : px);
        float cy = Parse_Number('J', mode_abs ? py : 0) + (mode_abs ? 0 : py);
        float newX = Parse_Number('X', mode_abs ? px : 0) + (mode_abs ? 0 : px);
        float newY = Parse_Number('Y', mode_abs ? py : 0) + (mode_abs ? 0 : py);
        float newZ = Parse_Number('Z', mode_abs ? pz : 0) + (mode_abs ? 0 : pz);
        TD_Arc(cx, cy, pz, newX, newY, newZ, (cmd == 2) ? -1 : 1);
        break;
    }
    case 4:
        // Пауза
        Pause((long)(Parse_Number('P', 0) * 1000000)); // P задається в секундах, переводимо в мікросекунди
        break;
    case 90:
        // Абсолютний режим координат
        mode_abs = 1;
        Serial.println(F("Режим координат: Абсолютний"));
        break;
    case 91:
        // Відносний режим координат
        mode_abs = 0;
        Serial.println(F("Режим координат: Відносний"));
        break;
    case 92:
        // Встановлення поточної позиції без руху
        px = Parse_Number('X', px);
        py = Parse_Number('Y', py);
        pz = Parse_Number('Z', pz);
```

```
            break;
        case -1:
            // Немає G-коду, можливо M-код або інше
            if (Serial_Buffer[0] == 'M') {
                int mcode = (int)Parse_Number('M', -1);
                if (mcode != -1) Handle_M_Command(mcode);
                else Serial.println(F("Невідома команда"));
            } else {
                Serial.println(F("Невідома команда"));
            }
            break;
        default:
            Serial.print(F("Невідома G-команда: G"));
            Serial.println(cmd);
            break;
    }
}

/**
* Підготуємо вхідний буфер для отримання нового повідомлення
* та повідомимо пристрій, який підключений через
56* послідовний порт, що він готовий
* до прийому наступного повідомлення.
**/
void Ready() {
symqnty=0; // очищуємо буфер
Serial.print(F(">")); // подаємо сигнал готовності до прийому
}
/**
* Початкові налаштування контролера
*/
void setup() {
Serial.begin(BAUD_RATE); // ініціалізація послідовного порту
Setup_Controller();
Position(0,0); // встановлюємо початкову позицію
```

```
Feed_Rate((MAX_FEED_RATE + MIN_FEED_RATE)/2); // Встановлюємо швидкість переміщення (подачі) за замовчуванням
Help(); // Привітаємось!:)
Ready();
}
/**
* Головний цикл
*/
void loop() {
// Очікуємо команди на вході послідовного порту
while(Serial.available() > 0) { // якщо щось надійшло
char c=Serial.read(); // отримуємо
Serial.print(c); // Надсилаємо зворотнє ехо, слугує для перевірки обладнання
if(symqnty<MAX_BUFER-1) Serial_Buffer[symqnty++]=c; // зберігаємо отримане
if((c=='\n') || (c == '\r')) {
// вхідна команда отримана
Serial_Buffer[symqnty]=0;
Serial.print(F("\r\n"));
Execute_Command();
Ready();
} }}
```