МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
Факультет комп'ютерно-інтегрованих технологій, мехатроніки і робототехніки
Кафедра робототехніки, електроенергетики та автоматизації ім. проф. Б.Б. Самотокіна

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи магістра

на тему: «**Автоматизована система управління намотуванням філаменту як основного етапу процесу виготовлення 3D-принтерного матеріалу»**

Виконав студент 2 курсу, групи АТ-28м спеціальності 174 "Автоматизація, комп'ютерно-інтегровані технології та робототехніка"
Горлаківський Антон Сергійович
(прізвище та ініціали студента)

Керівник: к.т.н., доц. Ткачук А.Г.
(науковий ступінь, вчене звання, посада, прізвище та ініціали)

Рецензент: д.т.н., проф. Подчашинський Ю.О.
(науковий ступінь, вчене звання, посада, прізвище та ініціали)

Житомир – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ "ЖИТОМИРСЬКА ПОЛІТЕХНІКА"
Факультет комп'ютерно-інтегрованих технологій, мехатроніки і робототехніки
Кафедра робототехніки, електроенергетики та автоматизації ім. проф. Б.Б. Самотокіна

Спеціальність 174 "Автоматизація, комп'ютерно-інтегровані технології та робототехніка"

ЗАТВЕРДЖУЮ
Завідувач кафедри
робототехніки, електроенергетики
та автоматизації
ім. проф. Б.Б. Самотокіна
к.т.н., доц. Громовий О.А.

к.т.н., доц. Громовий О.А.
«_____» _____ 2024р.

ЗАВДАННЯ
на кваліфікаційну роботу магістра

Студент *Горлаківський Антон Сергійовичу*

1. Тема роботи: *Автоматизована система управління намотуванням філаменту як основного етапу процесу виготовлення 3D-принтерного матеріалу*

Затверджена Наказом по університету від *«12» листопада 2024 р. № 646/с*

2. Термін здачі закінченої роботи на кафедру *«20» грудня 2024 р.*

3. Вихідні дані до роботи *Вступ та аналіз загальних відомостей. Розробка робочого макета пристрою для намотування. Створення графічного інтерфейсу оператора на базі Raspberry. Розробка алгоритмів керування та регулювання крокових двигунів мовою «С». Розробка системи відслідковування та фільтрації вхідних сигналів з датчиків на основі мікроконтролера ATmega2560. Налагодження та тестування зібраної установки*

4. Зміст пояснювальної записки (перелік питань, які підлягають розробці) *технічне завдання, аналіз використаного обладнаня та стеку технологій, аналіз об'єкта, розробка алгоритму взаємодії оператора, розробка алгоритму керування моторів та взаємодії з датчиками, розробка структурної, функціональної та принципової схем*

5. Перелік презентативного матеріалу (з точним визначенням обов'язкових слайдів/креслень)

Структурна схема, функціональна схема, принципова схема, блок-схема алгоритму системи

6. Консультанти з кваліфікаційної роботи магістра із зазначенням розділів роботи, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1, 2, 3	Ткачук А.Г., ктн, доцент	11.09.2024	11.09.2024
4,5,6	Ткачук А.Г., ктн, доцент	11.09.2024	11.09.2024

7. Дата видачі завдання 11.09.2024

Календарний план

№ з/п	Назва етапів виконання кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	Підготувати звіт про переддипломну практику	*1.11.2024 – 1.12.2024*	виконано
2.	Збір літературних даних за темою дипломної роботи та відомих систем вимірювання рівня рідини	*21.09.2024 – 24.09.2024*	виконано
3.	Ознайомитись з основними принципами побудови систем управління кроковими двигунами	*24.09.2024 – 11.10.2024*	виконано
4.	Розробка структурної та функціональної схеми системи керування	*11.10.2024 – 02.11.2024*	виконано
5.	Розробка принципової схеми	*02.11.2024 – 10.11.2024*	виконано
6.	Налагодження взаємодії між пристроями для коректної роботи	*10.11.2024 – 19.11.2024*	виконано
7.	Розробка блок-схеми алгоритму роботи пристрою намотування та дій оператора	*20.11.2024 – 11.12.2024*	виконано
8.	Оформлення пояснювальної записки та презентаційного матеріалу	*30.11.2024 – 11.12.2024*	виконано

Студент _____ А.С Горлаківський
(підпис, дата) (прізвище та ініціали)

Керівник роботи _____ А.Г. Ткачук
(підпис, дата) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота магістра на тему "Автоматизована система управління намотуванням філаменту як основного етапу процесу виготовлення 3D-принтерного матеріалу" містить 92 сторінок, 60 рисунків, 1 таблицю та 13 джерел.

Магістерська робота присвячена розробці автоматизованої системи управління процесом намотування філаменту — ключового етапу у виробництві матеріалу для 3D-друку. У роботі проаналізовано основні технологічні вимоги до намотування полімерного філаменту, зокрема дотримання стабільної швидкості, натягу нитки та точності укладання на котушку. Розроблено структурну і функціональну схему системи автоматичного регулювання параметрів намотування, сформовано алгоритм її роботи.

ANNOTATION

The master's thesis titled "Automated Control System for Filament Winding as a Key Stage in the Production Process of 3D Printer Material" consists of 92 pages, 60 figures, 1 table, and 13 references.

The thesis is devoted to the development of an automated control system for the filament winding process — a crucial stage in the production of material for 3D printing. The study analyzes the main technological requirements for winding polymer filament, including maintaining stable speed, tension control, and accurate placement of the filament on the spool. A structural and functional diagram of the automatic control system has been developed, along with an operational algorithm of the system.

РЕФЕРАТ

Дипломна робота присвячена створенню автоматизованої системи управління намотуванням філаменту, що є одним із ключових етапів виробництва матеріалу для 3D-друку. Основна мета проєкту — забезпечити високу точність і ефективність намотування при мінімізації втрат матеріалу та забезпеченні стабільної роботи системи.

Система складається з кількох підсистем, інтегрованих у єдину структуру. Для керування механізмами використано комбінацію Raspberry Pi, яке відповідає за високорівневе управління та забезпечення графічного інтерфейсу оператора, та мікроконтролера ATmega2560, який реалізує низькорівневу логіку управління двигунами та обробку сигналів датчиків. У роботі були розроблені алгоритми регулювання швидкості намотування залежно від стану компенсуючого механізму.

Графічний інтерфейс користувача, створений на Raspberry Pi забезпечує зручний доступ до налаштувань, таких як швидкість намотування, параметри філаменту (довжина, діаметр, ширина), а також дозволяє запускати чи зупиняти процес намотування. На етапі розробки було проведено експериментальні дослідження, які виявили залежність точності роботи системи від швидкості комунікації між Raspberry Pi і ATmega2560. Крім того, проведено тестування стабільності роботи компенсуючого механізму при різних базових швидкостях, що дозволило оптимізувати параметри роботи системи.

Для вирішення проблем, пов’язаних із електромагнітними перешкодами від драйвера серводвигуна, були впроваджені фільтри, що значно зменшили кількість хибних спрацювань системи. Після інтеграції всіх підсистем система була протестована в реальних умовах, де довела свою надійність, точність та продуктивність. Впровадження таких рішень дозволяє підвищити ефективність виробничих процесів, пов’язаних із 3D-друком, і може бути масштабоване для інших промислових застосувань.

Зміст

ВСТУП 7
1. Огляд сучасних технологій у виробництві 3D-принтерного філаменту .. 8
1.1. Аналіз ринку філаменту для 3D-друку 8
1.2. Основні вимоги до намотування філаменту 18
1.3. Сучасні методи та системи автоматизації намотувальних процесів 20
1.4. Вибір технологій і компонентів для автоматизації 22
2. Технічне завдання та концептуальна розробка системи 28
2.1. Постановка задачі автоматизації намотування філаменту 28
2.2. Розробка структури автоматизованої системи управління 29
2.3. Вибір обладнання та технологій: сенсори, виконавчі механізми, контролери, інтерфейси 37
3. Розробка апаратної частини пристрою 47
3.1. Розробка електричної принципової схеми 47
3.2. Розробка структурної схеми пристрою 51
3.3. Розробка функціональної схеми системи 53
3.4. Розробка схеми алгоритму роботи панелі оператора 54
3.5. Опис алгоритму роботи оператора 59
4. Розробка програмного забезпечення 62
4.1. Розробка графічного інтерфейсу та алгоритмів обміну даними на базі Raspberry Pi 62
4.2. Розробка алгоритмів керування та регулювання крокових 76
5. Експериментальні дослідження та тестування 86
5.1. Оцінка та інтеграція всіх підсистем 86
5.2. Тестування працездатності пристрою 87
Висновок 90
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 91

ВСТУП

Зростання популярності 3D-друку стимулює розвиток технологій виготовлення філаменту, який є основним витратним матеріалом для 3D-принтерів. Для забезпечення якісно друку, потрібен якісний матеріал. Окрім хімічних властивостей філаменту, одним із ключових факторів є процес намотування екструдованого філаменту на бобіну, що вимагає високої точності, стабільності та автоматизації. Ручне виконання цього процесу є малоефективним, призводить до значних відхилень у якості та збільшує витрати часу. У зв’язку з цим автоматизація процесу намотування стає важливим завданням, яке забезпечує підвищення продуктивності та стабільності виробництва.

Метою роботи є розробка автоматизованої системи управління процесом намотування філаменту, яка забезпечує точне регулювання швидкості, стабільність натягу та зручний інтерфес керування для оператора.

Об’єктом дослідження є процес намотування екструдованого філаменту на бобіну, влив параметрів намотування на якість. Предметом дослідження виступає автоматизована система управління, яка забезпечує ефективну організацію цього процесу.

Практичне значення роботи полягає у створенні робочого макету пристрою, який може бути використаний у виробничих лініях для виготовлення філаменту. Застосування запропонованої системи дозволить підвищити якість намотування, зменшити втрати матеріалу та покращити ефективність виробництва.

1. ОГЛЯД СУЧАСНИХ ТЕХНОЛОГІЙ У ВИРОБНИЦТВІ 3D-ПРИНТЕРНОГО ФІЛАМЕНТУ

1.1. Аналіз ринку філаменту для 3D-друку

Адитивні технології, більш відомі як 3D-друк, є однією з форм адитивного виробництва. Цей процес дозволяє створювати тривимірні об'єкти шляхом пошарового нанесення матеріалу згідно з цифровою моделлю. Реалізація такого виробництва здійснюється за допомогою спеціалізованого обладнання — 3D-принтерів, які перетворюють віртуальні 3D-моделі в фізичні об'єкти через послідовне нанесення пластичного матеріалу. У порівнянні з іншими технологіями адитивного виробництва, 3D-принтери відзначаються швидкістю роботи, доступністю та простотою використання. Вони дозволяють друкувати деталі й механізми з різноманітних матеріалів, забезпечуючи їхні унікальні механічні та фізичні властивості в одному циклі.

Часто 3D-друк називають «магічною» технологією через його здатність створювати готові вироби з моделей, створених у CAD-системах. Проте, реальний процес вимагає значної підготовчої роботи, а також подальшої обробки надрукованих деталей для досягнення необхідної якості.

Fused Deposition Modeling — FDM. Основною частиною принтера, що з'явився на ринку в 1991 р., є екструдована голівка. У ній матеріал (ливарний віск або пластик, що надходять з котушок) нагрівається до температури плавлення і подається в зону друку. Головка переміщається по двох координатах, синтезуючи певний шар моделі. Потім платформа опускається, створюється новий шар і т. д.

FDM переклад - (моделювання методом наплавлення) з'явилася у 1991 році. Основним компонентом таких 3D-принтерів є екструзійна голівка, яка плавить матеріал (зазвичай пластик або ливарний віск, що подається з котушок) та наносить його в зону друку. Голівка рухається по осях X та Y, створюючи шар за шаром модель, після чого платформа опускається для формування наступного шару.



Рис. 1.1 Типовий 3D принтер технології FDM

Переваги FDM-технології:

- Простий перехід між різними нетоксичними матеріалами.
- Економічність та висока продуктивність.
- Низькі робочі температури та мінімальне втручання оператора в процес.
- Можливість створення кольорових моделей.
- Відносно точний результат друку.

Недоліки FDM-технології:

- Утворення швів між шарами.
- Ризик засмічення екструзійної голівки в разі її зупинки.
- Розшарування через температурні коливання під час друку.
- Груба текстура поверхні надрукованих виробів.

Вартість FDM-принтерів починається приблизно від 150 доларів США.

Після підготовки моделі до роботи користувач вставляє котушку з філаментом у принтер. Як тільки сопло досягає заданої температури, нитка подається через екструзійну головку, яка наносить її шарами згідно з траєкторією, визначеною проєктом. Екструзійна головка рухається по осях X, Y і Z за допомогою тривісної системи.

Загалом усі FDM-принтери складаються з двох основних частин.

Екструдер(рис. 1.1.2) відповідає за подачу та розплавлення філаменту. Він складаються з кількох частин — механізму подачі філаменту (фідера), хотенду для плавлення матеріалу та сопла (філь'єри) для подачі розплаву. Хотенди, своєю чергою, поділяються на три зони — холодну, роздільну (термобар'єр) і гарячу.

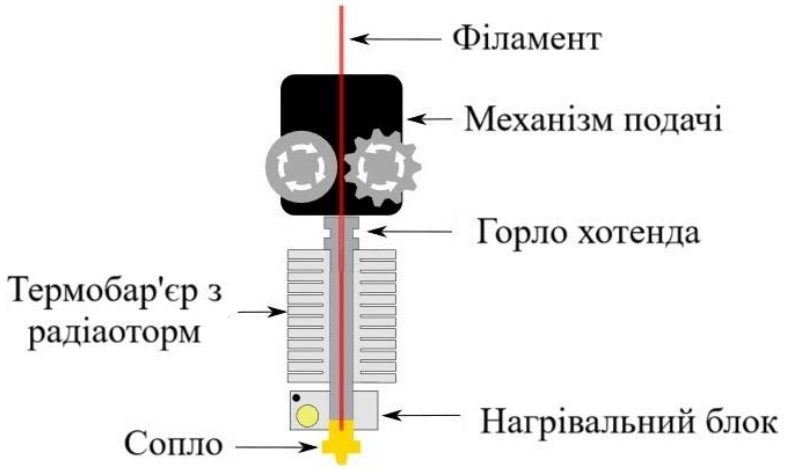


Рис. 1.2 Будова екструдера принтера FDM

Завдання термобар'єра — забезпечити максимально різкий перехід матеріалу з твердої фази в рідку. Надто плавний перехід може спричинити застрягання філаменту через високе тертя, викликане тепловим розширенням у твердій фазі. Тому термобар'єри обладнують радіаторами з повітряним охолодженням. Філамент проходить через горловину в холодній частині, потім через охолоджуваний термобар'єр, а далі в гарячу частину, де нагрівається термоблоком і видавлюється через сопло.

Сопла можуть виготовлятися з різних матеріалів. Найчастіше використовується латунь, популярна через низьку вартість і добру теплопровідність. Для роботи з високоабразивними матеріалами необхідно використовувати зносостійкі сопла, наприклад, із загартованої сталі.

Залежно від методу подачі філаменту екструдери поділяються на боуденівські та директ-екструдери (див. «Боуденівська та директ-подача»).

Робоча платформа - це плоска поверхня, на яку наноситься філамент. Зазвичай платформа підігрівається, щоб забезпечити зчеплення між шарами та уникнути деформації. Вона може переміщуватися по осях X і Y, адаптуючись до розмірів друкованого об’єкта.

Типи робочих платформ:

Нагріта платформа - використовується для запобігання деформації матеріалу під час друку, особливо для матеріалів, таких як ABS або PETG.

Температура підтримує оптимальне зчеплення між шарами та основою.

Холодна платформа - підходить для друку матеріалами з низькою схильністю до деформації, наприклад PLA.

Часто має спеціальне покриття, що покращує адгезію, як-от скло або магнітні накладки.

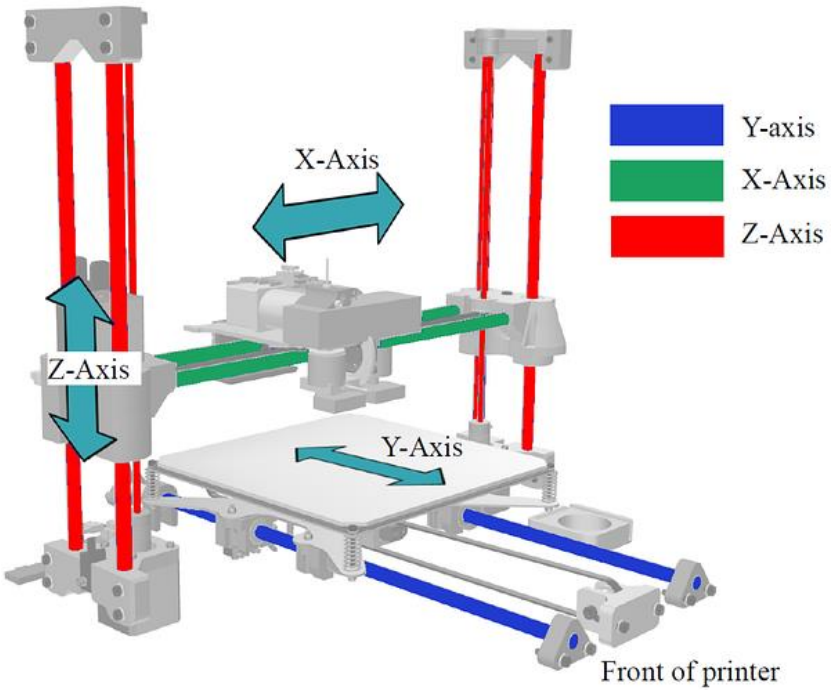


Рис. 1.3 Картезіанська кінематика

Кінематика 3D-принтера визначає, як його частини рухаються відносно одна одної, щоб створювати моделі. Вона впливає на точність, швидкість і надійність друку. Основні типи кінематик — це картезіанська, дельта, CoreXY, H-bot, SCARA та полярна.

Класична конструкція (картезіанська)

Картезіанська конструкція 3D-принтерів базується на системі координат XYZ, де кожна вісь відповідає за рух у своєму напрямку. Така конструкція є однією з найпоширеніших через свою простоту та доступність у виробництві.

Система CoreXY.Ремінна механіка забезпечує синхронний рух по осях X та Y. Знижує масу рухомих компонентів, що підвищує швидкість і точність. Використовується у високопродуктивних принтерах.

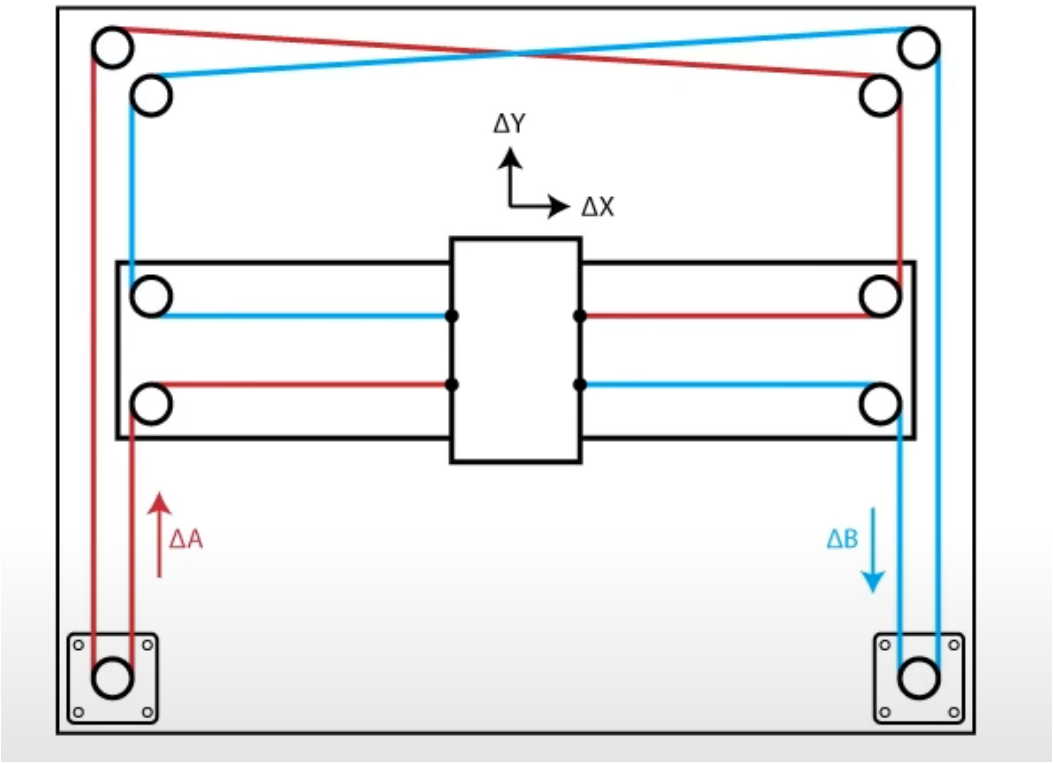


Рис. 1.4 Кінематика CoreXY

Математично рух можна описати так. Якщо A - рух першого двигуна, B рух другого двигуна, то рух у напрямку x та y визначається як:

$$\Delta x = \frac{1}{2}(\Delta A + \Delta B)$$

$$\Delta y = \frac{1}{2}(\Delta A + \Delta B)$$

Дельта-конструкція

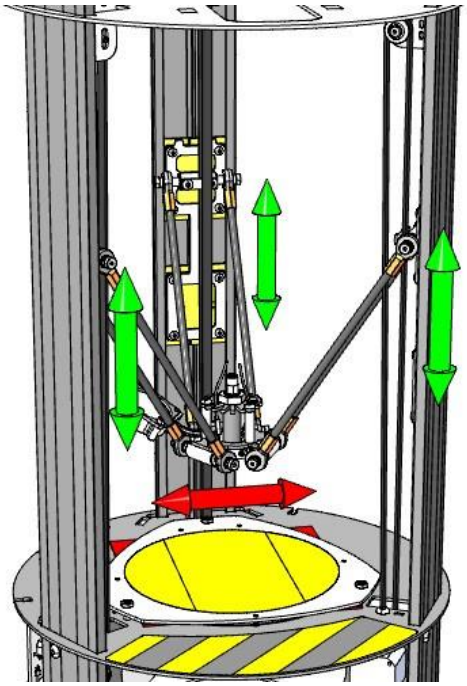


Рис. 1.5 Кінематика дельта

Екструдер підвішений на трьох напрямних, що забезпечують рух у будь-якій точці в межах робочої зони.

Відсутність окремих осей X та Y. Забезпечує високу швидкість, але має обмеження в точності. Математична модель руху даної кінематики є досить складною, тому у даній роботі не доцільно приводити її приклад.

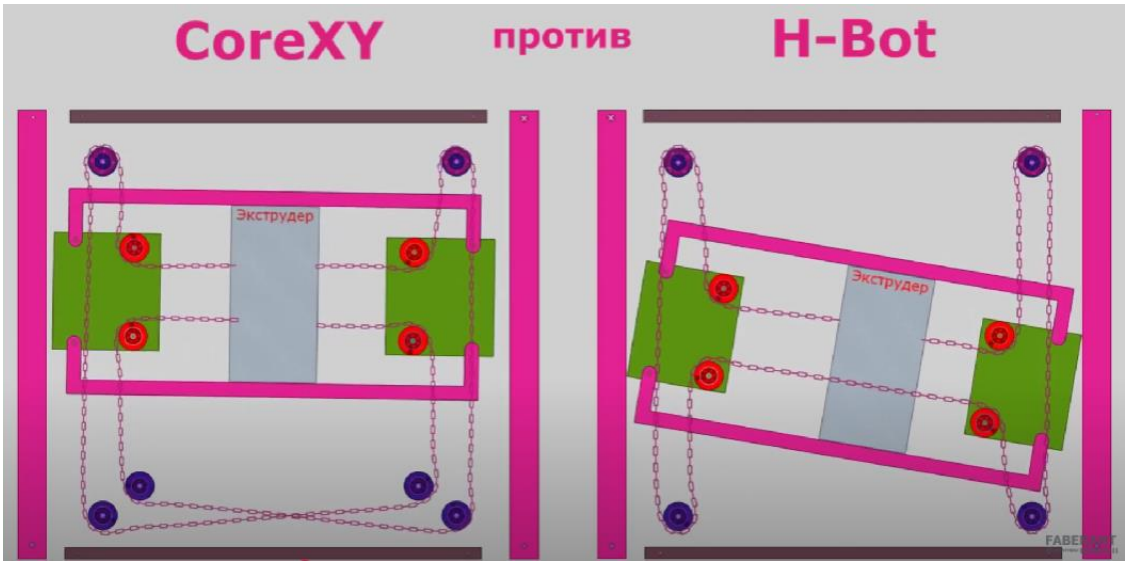


Рис. 1.6 Кінематика H-bot

Гармошкова система H-bot. Схожа на CoreXY, але ремені рухаються більш лінійно, що знижує ризик деформації. Простий дизайн, однак менш надійний під час друку великих моделей через перекоси.

SCARA-маніпулятор - арм-подібна механіка з гнучким досягненням будь-яких точок у робочій зоні.

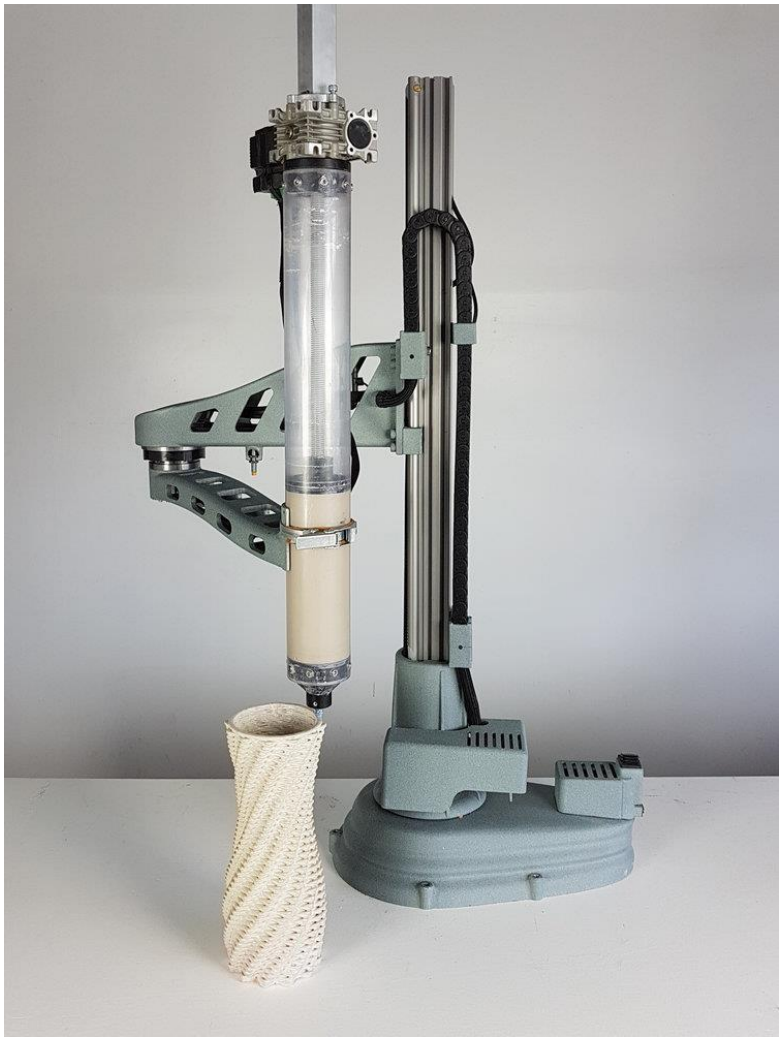


Рис. 1.7. Приклад принтера кінематики scara

Використовується в експериментальних 3D-принтерах.

Система керування. До її складу входить мікроконтролер, який інтерпретує команди слайсера, регулює температуру екструдера та платформи, а також координує рух двигунів.

Матеріали для FDM-друку

PLA - легко друкується, має низький рівень деформації, глянсову поверхню та екологічне походження (на основі кукурудзяного крохмалю або цукрової тростини). Підходить для створення прототипів і декоративних моделей.

ABS - відрізняється високою механічною міцністю, стійкістю до високих температур і ударам. Використовується для функціональних компонентів, таких як корпуси пристроїв і автомобільні деталі.

PETG - поєднує властивості PLA та ABS. Стійкий до хімічного впливу й ударів, широко застосовується в медичній, харчовій та рекламній галузях.

ASA - альтернатива ABS з кращою стійкістю до ультрафіолету, атмосферних впливів і хімічного впливу. Підходить для зовнішніх конструкцій і функціональних деталей.

Технологія FDM: Вплив на галузі та перспективи

Для багатьох підприємств FDM пропонує ефективний і продуктивний спосіб прототипування та дрібносерійного виробництва. Ця технологія дозволяє швидко вносити зміни до проєктів, забезпечуючи командам можливість оперативно проходити через процес розробки та адаптувати дизайн за потреби. Крім того, для дрібносерійного виробництва FDM часто є більш економічно вигідним у порівнянні з традиційними методами, особливо коли йдеться про складні або індивідуальні вироби.

Технологія FDM залишила значний слід у багатьох галузях, таких як автомобілебудування, авіакосмічна промисловість, охорона здоров'я та виробництво споживчих товарів. Завдяки своїй здатності створювати складні та довговічні деталі швидко й економічно, FDM є привабливим вибором для швидкого прототипування, функціонального тестування та навіть серійного виробництва.

Автомобілебудування та авіакосмічна промисловість: FDM використовується для створення прототипів нових конструкцій, допоміжних інструментів для збирання і навіть готових деталей. Легкі та міцні компоненти FDM ідеально підходять для галузей, де важлива економія ваги.

FDM змінила підхід до медицини, забезпечуючи можливість створення пацієнт-специфічних моделей для хірургічного планування та індивідуальних протезів з використанням біосумісних матеріалів.

Завдяки FDM компанії можуть швидко створювати прототипи нових продуктів, проводити ринкові тестування та навіть виробляти на замовлення індивідуальні готові вироби.

Можливості для хобі та навчання. FDM-технологія зробила революцію у світі хобі та DIY-ентузіастів, відкривши можливості для створення та інновацій

у домашніх умовах. Від особистих проєктів до навчальних платформ, FDM стала незамінним інструментом для творчих людей.

3D-принтери доступні в різних моделях — від початкового рівня до професійного, щоб задовольнити різні потреби.

Моделі початкового рівня, доступні та прості у використанні, наприклад Creality Ender 3 або Anycubic i3 Mega. Забезпечують хорошу якість друку й активно підтримуються спільнотами ентузіастів.

Моделі середнього рівня - Prusa i3 MK3S, Bambu lab, вони пропонують баланс між ціною та функціональністю.

Моделі для професіоналів та інженерів - Bambu Lab X1E Combo, Ultimaker S5 або Formlabs забезпечують високу якість друку, розширені функції та більший об’єм побудови, що робить їх ідеальними для серйозних хобістів і малих підприємств.

Освітній потенціал FDM

FDM-технологія також слугує чудовим освітнім інструментом. Вона забезпечує практичний досвід у різних дисциплінах STEM (наука, технології, інженерія, математика), дозволяючи студентам глибше зрозуміти ці галузі.

Технологія FDM продовжує розширювати межі створення та виробництва, пропонуючи як захоплююче хобі, так і можливості для підприємництва. Незалежно від того, чи ви професіонал, який шукає оптимальні рішення, чи ентузіаст, який відкриває нові горизонти, FDM надає безліч можливостей для інновацій та розвитку.

Використання та важливість технології FDM

FDM (Fused Deposition Modeling) – це одна з найбільш поширених технологій 3D-друку. Завдяки її доступності, універсальності та здатності працювати з різними матеріалами, FDM знайшла застосування у багатьох галузях і стала ключовою для сучасного виробництва.

Прототипування та дизайн

Швидке виготовлення прототипів: FDM дозволяє створювати фізичні моделі з цифрових проєктів, що дає змогу швидко тестувати конструкції та проводити ітерації.

Дизайн-концепти: Архітектори, дизайнери та інженери використовують FDM для створення макетів і концептуальних моделей.

Приклад: Автомобільна компанія може виготовити деталь для тестування форми та функціоналу перед запуском серійного виробництва.

Освіта

STEM-освіта: FDM-принтери використовують для навчання учнів і студентів інженерії, програмуванню, 3D-моделюванню.

Демонстрація реальних процесів виробництва: У школах та університетах 3D-принтери використовуються для показу роботи сучасних технологій.

Приклад: Учні можуть створювати власні механізми або досліджувати закони фізики, друкуючи експериментальні установки.

Медицина

Персоналізовані вироби: FDM дозволяє створювати індивідуальні протези, ортези, хірургічні моделі.

Навчання лікарів: Моделі органів, виготовлені за допомогою FDM, допомагають у плануванні складних операцій.

Приклад: Друк моделі кістки пацієнта для тренування перед операцією.

Виробництво деталей

Малі серії: Виготовлення невеликих партій деталей за низькою ціною, зокрема для заміни пошкоджених компонентів.

Засоби виробництва: Створення нестандартних інструментів, пристосувань або шаблонів для виробничих потреб.

Приклад: Підприємство друкує власні кріплення або захисні корпуси для електроніки.

Індустрія розваг

Моделювання: FDM активно використовується у створенні моделей, макетів, костюмів для фільмів, ігор, виставок.

Тематичні аксесуари: Виробництво сувенірів, кастомних фігурок чи декору.

Приклад: 3D-друк елементів обладунків для зйомок фантастичних фільмів.

Військова та аерокосмічна галузі

Виготовлення прототипів: Легкі та міцні матеріали роблять FDM корисною для створення компонентів, що піддаються екстремальним умовам.

Заміна деталей: Можливість друкувати деталі безпосередньо на місці, наприклад, у польових умовах.

Приклад: Друк тимчасових деталей для ремонту літаків.

Хобі та DIY-проєкти

Персоналізовані вироби: Люди друкують унікальні об'єкти, такі як прикраси, іграшки, корисні речі для дому.

Відновлення: Друк деталей для ремонту побутових пристроїв.

Приклад: Друк кріплень для меблів, які більше не виробляються.

FDM-технологія стала важливим інструментом у багатьох галузях завдяки таким ключовим перевагам:

Низький поріг входу для новачків і можливість створення деталей за мінімальні витрати. Використання з широким спектром матеріалів – від PLA для простих виробів до спеціалізованих матеріалів для інженерії чи медицини.

Можливість швидко виготовляти прототипи та дрібносерійні деталі. FDM ідеально підходить для персоналізованого виробництва, де кожна деталь може бути унікальною. Ну і найбільш важлива тема це підтримка комюніті, адже загалом спільнота, що займається FDM-друком, налічує приблизно 10–15 мільйонів людей по всьому світу. Хоча FDM має певні обмеження її переваги роблять цю технологію незамінною у багатьох сферах життя.

FDM – це не лише спосіб створювати фізичні об’єкти, а й рушійна сила інновацій, що відкриває нові горизонти для творчості, навчання та виробництва.

1.2.Основні вимоги до намотування філаменту

В процесі друку пруток рухається від котушки до сопла. Електродвигун постійного струму забезпечує рівномірну подачу прутка. В процесі роботи пруток може затягуватись до екструдера, а може відкатуватись в зворотньому напряку. Це спричинено певними особливостями FDM друку.

Для стабільного та якісного друку, одним із важливих факторів є - стабільна та вільна подача філаменту до сопла. З іншого боку завадою для цього можуть бути певні фактори.

Детальні розглянемо проблеми, пов’язані з неточною намоткою філаменту на котушках.

Якщо філамент намотаний з заплутуваннями чи перехрещенням можуть утворюватись вузли чи прогалини. Під час роботи 3D-принтера такі ділянки блокують плавний рух філаменту, що спричиняє паузи в друку або повну зупинку. Також часто трапляються випадки коли філамент після перегину розривається, це призводидь до втрати деталі, а також часу на те, щоб виявити та усунути проблему. Це негативно впливає на якість виробу та змушує користувача починати друк заново.

Хаотична намотка збільшує відстань між витками, через що котушка не використовується повністю. Як результат, зменшується кількість філаменту, який можна намотати на котушку стандартного розміру.

Нерівномірна натяжка під час намотування викликає надмірний тиск на окремі ділянки філаменту, що може призвести до утворення тріщин, розтягування або зміни діаметра. Такі пошкодження роблять філамент менш придатним для використання, особливо у випадках, де потрібна висока точність друку. Також це може викликати проблеми з обертанням під час друку, що в свою чергу може спричинити механічні пошкодження принтера або збої в подачі.

Неточна намотка може призводити до застрягання філаменту, яке користувачі часто вирішують обрізанням пошкоджених або заплутаних ділянок. Це не тільки збільшує відходи матеріалу, а й спричиняє фінансові втрати також це може впливати на збільшення екологічних забруднень.

Котушки з нерівною намоткою виглядають неакуратно та створюють у користувачів враження низької якості виробництва. Це може вплинути на довіру клієнтів, особливо у випадках, коли продукт продається під преміум-брендом або на конкурентному ринку.



Рис. 1.8. Приклад погано та гарно вкладеного філаменту

Заплутування чи нерівна подача філаменту викликає нерівномірний тиск у зоні екструзії, що призводить до змін у товщині шарів друкованого об’єкта. Це може вплинути на механічні та естетичні характеристики готового виробу.

Котушки з нерівномірною намоткою часто менш стійкі до ударів або поштовхів під час транспортування. Крім того, неправильна намотка може спричинити зміщення шарів філаменту, що призводить до подальших ускладнень під час використання.

Якісна намотка забезпечує рівномірний і безперервний потік матеріалу в процесі друку. Це особливо важливо для довготривалих або складних проектів, які вимагають великої кількості пластику. Тому дуже важливо для виробників використовувати промислові методи намотування для забезпечення усіх необхідних параметрів намотування, що значно знижує ризик проблем під час друку та збільшує довіру клієнтів до виробника.

Проблеми, пов'язані з неточним намотуванням, стають значним бар'єром для якісного 3D-друку. Ця тема активно обговорюється у спеціалізованих статтях і на форумах ентузіастів, де користувачі діляться досвідом і рішеннями для уникнення цих проблем.

1.3. Сучасні методи та системи автоматизації намотувальних процесів

Наразі велика кількість підприємств використовують готові ришіння. Саме такі рішення мають великий вплив на ринку.

Передові методи намотування

Еволюція технології намотування призвела до появи передових технологій, адаптованих до конкретних застосувань. Деякі відомі методи включають:

Шарова обмотка : забезпечує рівномірний розподіл матеріалу в окремих шарах, що зазвичай використовується у виробництві трансформаторів і двигунів. Цей метод мінімізує електричні втрати та підвищує ефективність пристрою.

Довільна обмотка : ідеально підходить для застосувань, де обмеження простору менш критичні, наприклад, для певних типів індукторів. Цей метод є швидшим і економічно ефективнішим, але вимагає ретельного контролю натягу, щоб уникнути дефектів.

Тороїдальна обмотка : використовується для таких компонентів, як трансформатори та датчики, де магнітні властивості є вирішальними. Ця техніка передбачає намотування дроту навколо кільцеподібного сердечника, що оптимізує передачу енергії та зменшує шум.

Намотування млинців : ця техніка, яка часто використовується в аерокосмічній та медичній промисловості, передбачає створення плоских котушок, що робить її придатною для компактних і легких пристроїв.

Кожен із цих методів розроблено для задоволення конкретних потреб його застосування, будь то максимізація ефективності, зменшення шуму чи забезпечення компактності.

Тому виробникам намотувальних пристроїв, для забезпечення конкурентих переваг, важливо також слідувати таким критеріям.

Автоматизований намотувальний пристрій мінімізує людську помилку, забезпечуючи рівномірне намотування матеріалів і уникнення таких дефектів, як заплутування або неправильне накладання шарів.

Автоматизовані системи забезпечують високу швидкість виробництва, зберігаючи при цьому точність, яка є життєво важливою для промислового виробництва.

Точні системи контролю забезпечують ефективне використання матеріалів відповідно до цілей сталого розвитку.

Також, важливим фактором є гнучкість. Системи мають адаптуватися до різних типів матеріалів і конструкцій без значної реконфігурації, що забезпечує універсальність для виробників

1.4.Вибір технологій і компонентів для автоматизації

Для створення намотувального пристрою необхідно поєднання ключових компонентів, в механізм який забезпеить потрібні вимоги. А саме, забезпечують точний контроль швидкості, позиціювання, автоматизацію та зворотний зв'язок для ефективної роботи пристрою. Такі пристрою поділяються на виконавчі, сенсори та основні(main). В даному випадку задачі були розділені на саме такі частини.

Важливим для намотувального пристрою є механізм каретки. Він виконує зворотньо-поступальні рухи, забезпечуючи рівномірне дискретне крокування для вкладання матеріалу в межах одного шару.

Кроковий двигун, також відомий як кроковий або крокуючий двигун, — це безщітковий електродвигун постійного струму, який обертається за рахунок серії невеликих і дискретних кутових кроків. Крокові двигуни можуть бути встановлені в будь-яке задане крокове положення(рис. 1.4.1) без необхідності використання датчика положення для зворотного зв'язку. Крокове положення може швидко збільшуватися або зменшуватися для створення безперервного обертання, або двигуну можна задати утримання позиції на одному конкретному кроці. Двигуни варіюються за розмірами, швидкістю, роздільною здатністю кроків і моментом.

Для забезпечення головного обертового руху котушки, яка виконує намотування філаменту доцільно використовувати серво двигун з енкодером та підтримкою обертового моменту. Такий мотор обовязково повинен обертатись зі сталою швидкістю та підлаштовувати збільшувати чи зменшувати крутний момент під ситуацію. В процесі виробництва виникають різні фактори які дестабілізують натяг філаменту, що тягне за собою погіршення якості вкладання.

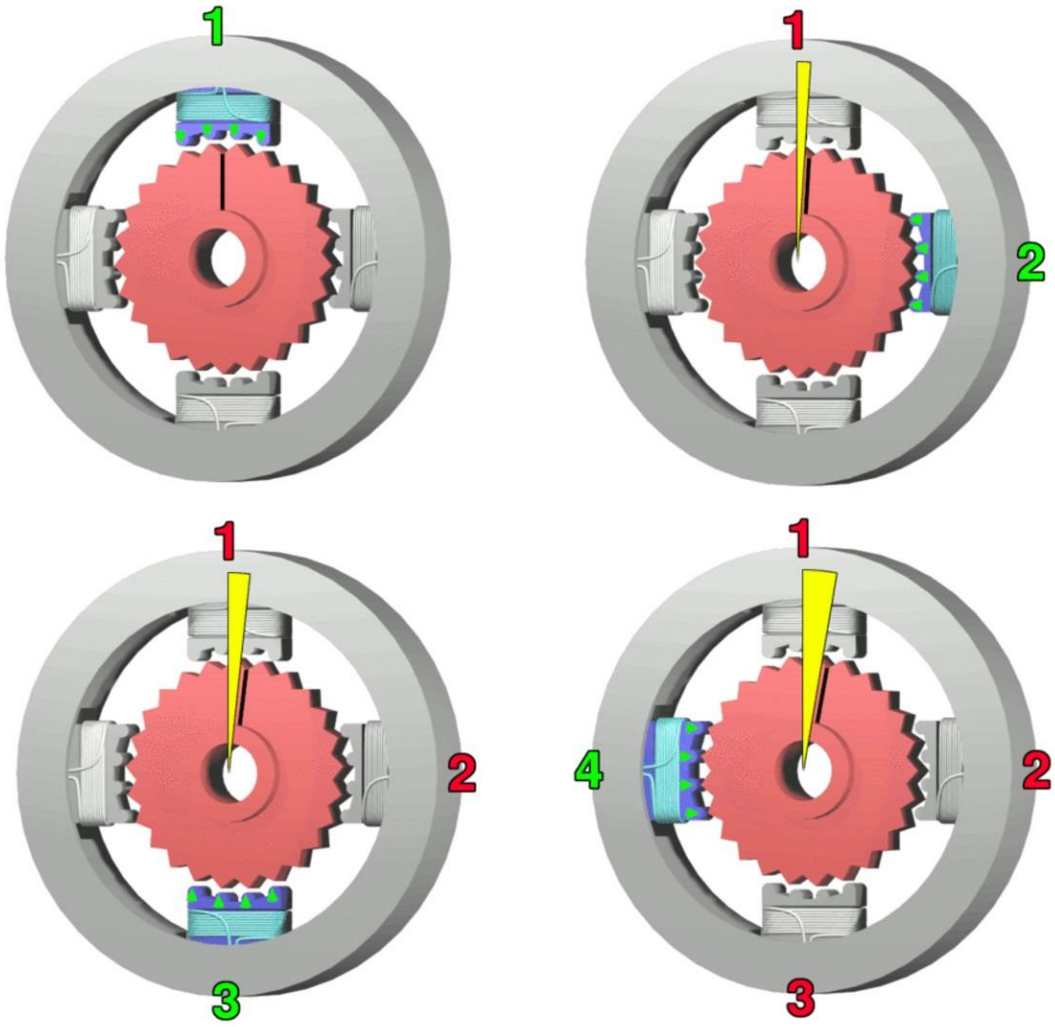


Рис. 1.9. Зміна кута двигуна покроково

Сервомотор (або серводвигун) — це обертальний або лінійний привід, який забезпечує точний контроль положення, швидкості та прискорення в механічних системах. Він складається з двигуна, датчика для зворотного зв'язку положення і контролера, що разом формують сервомеханізм(схематично рис.). Сервомотори не є окремим класом двигунів, але термін часто використовується для позначення моторів, що підходять для систем замкнутого керування.

Принцип роботи простий, двигун генерує обертальний рух, датчик фіксує поточне положення та/або швидкість, далі контролер порівнює задане положення з фактичним і забезпечує корекцію для досягнення точності.

Для керування такими моторами використовується сигнал, для керування обмотками в певний проміжок часу. Таким чином, для будь якого кровкового мотору потрібен окремий драйвер, що керуватиметься контролером та перемикатиме напругу на обмотках.

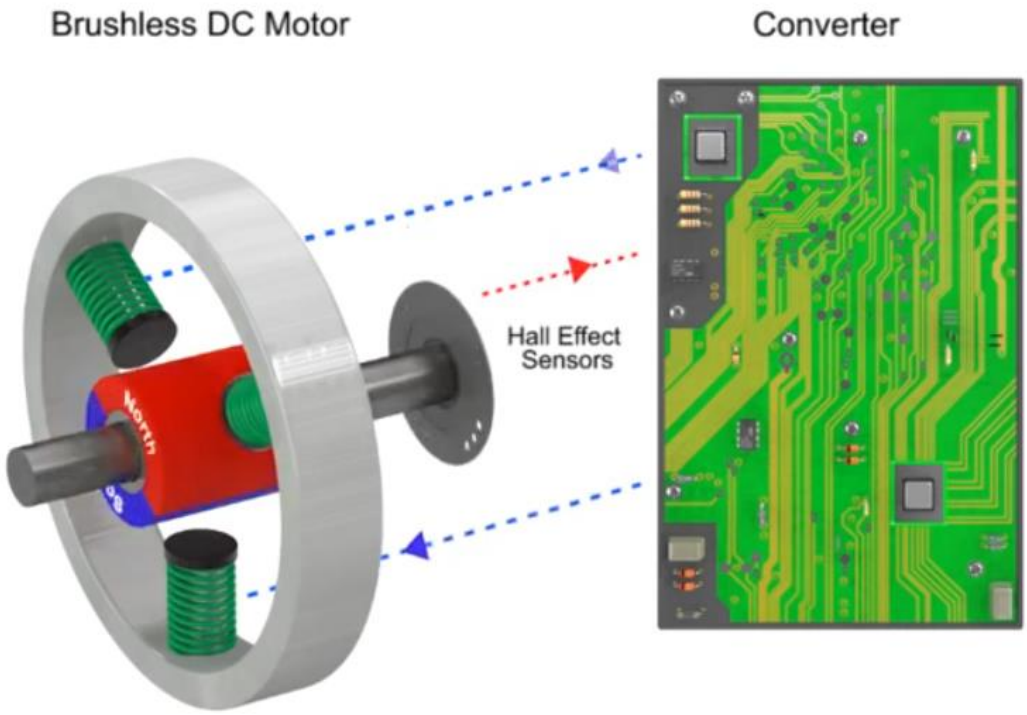


Рис. 1.10. Схематична робота серводвигуна

Кроковий двигун — це багатофазний синхронний двигун змінного струму. Ідеально його слід живити синусоїдальним струмом. Проте повно-крокова хвильова форма є лише грубим наближенням синусоїди, через що двигун відчуває сильні вібрації. Для кращого наближення синусоїди розроблено різні техніки керування: півкрокове управління та мікрокрокування.

Їстнує декілька видів крокових дивгунів, та способів керувати ними.

Хвильове керування (одна фаза активна):

- В цьому методі на кожному кроці активна лише одна обмотка.
- Цей режим забезпечує менший момент, ніж інші методи.
- Рідко використовується через низьку ефективність.

Повно-крокове керування (дві фази активні):

- Дві обмотки завжди активні, що забезпечує максимальний момент двигуна.
- Такий метод дозволяє досягати стабільної роботи при максимальному навантаженні.

Півкрокове управління:

- Чергуються стани "дві фази активні" та "одна фаза активна".
- Підвищує роздільну здатність, але знижує момент на повному кроці (приблизно на 30%).
- Не потребує значної модифікації електроніки для реалізації.

Мікрокрокування:

Використовуються спеціальні драйвери для створення синусоїдальних струмів у обмотках.

Забезпечує найгладшу роботу двигуна та мінімізує резонанс.

Може досягати високої точності, але обмежується механічним люфтом і похибками. Не менш важливими компонентом є датчики. Було використано енкодери та ємністні датчики.

Енкодери були використані для обрахунку довжини намотаного філаменту та для визначення положення копменсуючого механізму.

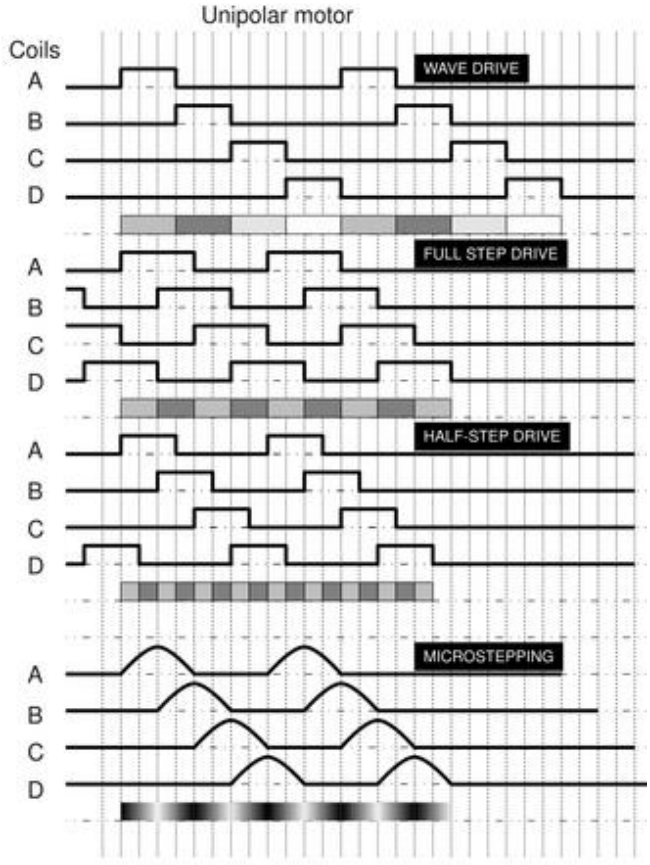


Рис. 1.11. Різні режими приводу, що показують струм котушки 4-фазного уніполярного крокового двигуна.

Інкрементальний енкодер — це лінійний або обертовий електромеханічний пристрій, який генерує два вихідних сигнали, позначених

як A і B, що видають імпульси під час руху пристрою. Ці сигнали разом вказують як факт руху, так і його напрямок.

Багато інкрементальних енкодерів мають додатковий вихідний сигнал, зазвичай позначений як індекс (або Z), який вказує, що енкодер знаходиться у певній референтній позиції. Деякі моделі також забезпечують сигнал стану (позначений, наприклад, як alarm), який вказує на внутрішні несправності, такі як вихід з ладу підшипників або датчика.

Сигнали з екодеру потрапляють до контролера, обробляються і після чого виконуються відповіді дії.

Інкрементальний енкодер — це лінійний або обертовий електромеханічний пристрій, який генерує два вихідних сигнали, позначених як A і B, що видають імпульси під час руху пристрою. Ці сигнали разом вказують як факт руху, так і його напрямок.

Багато інкрементальних енкодерів мають додатковий вихідний сигнал, зазвичай позначений як індекс (або Z), який вказує, що енкодер знаходиться у певній референтній позиції. Деякі моделі також забезпечують сигнал стану (позначений, наприклад, як alarm), який вказує на внутрішні несправності, такі як вихід з ладу підшипників або датчика.

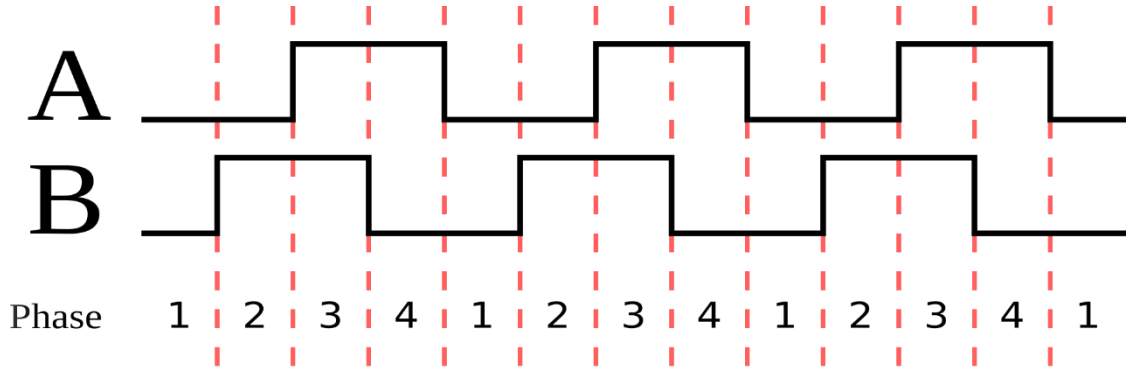


Рис. 1.12. Вигляд сигналу енкодера

Індуктивні датчики в даній розробці слугують кінцевою опорною точкою, для знаходження краю котушки.

Принцип дії заснований на змінюванні амплітуди коливань генератора при внесенні в активну зону датчика металевого, магнітного, феромагнітного або аморфного матеріалу певних розмірів. При подачі живлення на кінцевий вимикач, в ділянці його чутливої поверхні утворюється змінюване магнітне

поле, яке наводить у внесеному в зону матеріалі вихрові струми, які спричинюють змінення амплітуди коливань генератора. Результатом є аналоговий вихідний сигнал, величина якого залежить від відстані між датчиком і контрольованим предметом.

2. ТЕХНІЧНЕ ЗАВДАННЯ ТА КОНЦЕПТУАЛЬНА РОЗРОБКА СИСТЕМИ

2.1.Постановка задачі автоматизації намотування філаменту

Процес намотування філаменту для 3D-друку має критично важливе значення для забезпечення якості та зручності використання матеріалу. Основні завдання, що стоять перед автоматизацією цього процесу, висвітлені далі.

1. Контроль точності намотування

Філамент повинен намотуватись на котушку рівномірними шарами без перехрещень або зміщень. Забезпечити уникнення занадто щільного або розрідженого вкладання. Це значно впливає на масу котшки на виході.

Вирішити проблему надмірно слабкого натягу чи занадто сильного натягу, який може спричинити сплутування або розрив під час друку. Забезпечення вимірювання точної довжини філаменту.

2. Управління натягом філаменту

Натяг має залишатися стабільним протягом усього процесу, щоб уникнути пошкодження матеріалу або нерівномірного намотування. Також потрібна конструкція, що буде стабілізувати пруток, адже в процесі намотування він може гойдатись та вібрувати. Також потрібно сконструювати систему попереднього натягу, що дасть змогу філаменут, що потрапляє до пристрою намотування мати певну силу натягу. Розробити систему відслідковування положення стабілізуючого механізму.

3. Моніторинг та керування параметрами

Розробити зручний та простий інтерфейс оператора. Забезпечити контроль та зміну певних параметрів в процесі роботи.

Впровадити лічильник поточного намотаного метражу а також можливість зміни довжини намотування в метрах(500м, 800м, 1000м). Адже котушки різних виробників мають різний розмір та геометричні параметри. Також для забезпечення різної ваги котушок.

Швидкість обертання котушки. Забезпечити можливість змінювати швидкість обертання відповідно до заданої швидкості лінії, також підлаштовувати швидкість відповідно до стабілізуючого механізму. Тобто є

базова задана швидкість, і вона регулюється цим механізмом в більший бік до 150% RPM і в меншу сторону до 0 RPM. Забезпечити синхронізацію швидкості намотування з швидкістю вкладання філаменту.

Забезпечити можливість реглювання параметру ширини бабіни. Адже кінцевиком буде встановлено нульове положення і для регулювання ширини, на яку пересвається вкладальна каретка, на дисплеї повинно бути відповідні параметри. Також забезпечити зміну кроку вкладання. Базово для філаменту це 1.75мм. Існує ряд певних виробничих особливостей, що вимагають зменшити чи збільшити крок вкладання, для запобіганню погіршенню якості. За допомогою використання крокових двигунів можливо задавати лінійне пересування з точністю в 0,001мм.

Забезпечити роботу усіх складових даної установки в асинхронному режимі. Зв'язок із системами виробництва філаменту, такими як екструдери, електродвигуни та сенсори для забезпечення узгодженої роботи.

2.2.Розробка структури автоматизованої системи управління

Розробка структури АСУ для намотування філаменту базується на принципах автоматичного контролю та управління всіма етапами процесу. Кожна дрібна функція затрачає час мікроконтролера, тому доводиться розділяти задачі відстеження датчиків, керування двигунами та відображення інформації я UI та корегування параметрів. Нижче наведено ключові компоненти їх функції в системі.

Найголовніше це обробка сигналів з датчиків та кнопок. В системі є два енкодери та індуктивний датчик. На основі цих датчиків лежить робота всієї установки з намотування. Для обробки сигналів з енкодерів було використано апартні можливості контролера Atmega2560. А саме апартні таймери. Переривання генеруються окремим апаратним таймером, який знаходиться в мікроконтролері поруч із обчислювальним ядром.

Апаратний таймер, або лічильник, виконує дуже просту задачу: рахує "тікі" тактового генератора (що визначає частоту роботи всієї системи) і, залежно від режиму роботи, може змінювати стан вихідних пінів або надсилати сигнал мікроконтролеру при певних значеннях лічильника. Таким чином,

"роздільна здатність" роботи таймера становить один тик тактового генератора; при частоті 16 МГц це 0.0625 мікросекунди.

Другий важливий аспект: таймер-лічильник працює і рахує імпульси паралельно з обчислювальним ядром. Це дозволяє генерувати ШІМ-сигнал навіть на високій частоті без впливу на виконання основного коду, оскільки все відбувається паралельно.

В Arduino Mega (на базі ATmega2560) є iscnm таких таймери, кожен з яких може активувати незалежне переривання зі своїм періодом. Для обчислення часу функції millis() і micros() використовують таймер 0. Якщо змінити налаштування цього таймера, коректний підрахунок часу припиниться, а також може змінитися робота ШІМ на пінах 5 і 6.

Деякі бібліотеки теж використовують переривання таймерів, наприклад, Servo використовує перший таймер, а функція tone() — другий. Таймери також генерують ШІМ-сигнал на своїх пінах, і їх переналаштування може змінити режим роботи, вимкнути ШІМ або змінити його частоту.

На відміну від сторонніх бібліотек, можливості апаратних таймерів Atmega можна використовувати напряму через внутрішні регістри мікроконтролера. Це забезпечує високий рівень гнучкості та контроль над функціональністю таймерів без необхідності додаткових бібліотек.

Використання таймерів ATmega2560 здійснюється за допомогою налаштування спеціальних регістрів. Наприклад:

«TCCRnA» та «TCCRnB»: Визначають режим роботи таймера (Normal, PWM, CTC) і встановлюють дільник частоти.

«OCRnA/B»: Порівняльні регістри для генерації переривань або зміни ШІМ-сигналу.

«TIMSKn»: Увімкнення переривань за збігом або переповненням.

«TCNTn»: Рахунковий регістр, який містить поточне значення лічильника таймера.

Мікроконтролер ATmega2560 має два типи таймерів.

«Timer0» і «Timer2» — 8-бітні таймери. Використовуються для задач із меншою точністю. Підходять для генерації ШІМ-сигналів з помірною

роздільною здатністю. Можуть працювати у режимах Fast PWM, Phase Correct PWM, CTC, і Normal.

«Timer1», «Timer3», «Timer4», «Timer5» — 16-бітні таймери.

Висока точність та великий діапазон значень. Використовуються для точного відліку часу, генерації сигналів або роботи із сервоприводами.

16-бітні таймери дозволяють працювати з великим числом тактів до переповнення, що корисно для задач, які потребують точного вимірювання довгих інтервалів або низькочастотних сигналів.

Усі таймери мають окремі виходи для роботи з ШІМ, кожен з яких може генерувати незалежні сигнали. Підтримка режиму CTC (Clear Timer on Compare Match) дозволяє створювати точні таймери або лічильники подій.

Апаратні зовнішні переривання в мікроконтролерах: принцип роботи та застосування.

Що таке апаратні зовнішні переривання?

Зовнішнє апаратне переривання (External Hardware Interrupt) виникає, коли відбувається зміна сигналу на певному піні мікроконтролера. Цей процес реалізується апаратно, без постійного опитування пінів центральним процесором. Мікроконтролер миттєво реагує на зміну сигналу, виконуючи відповідний обробник переривання, а потім повертається до виконання основного коду.

Основні типи сигналів, які можуть викликати переривання:

Зростання (Rising) — перехід від низького рівня (0) до високого (1).

Спад сигналу (Falling) — перехід від високого рівня (1) до низького (0).

Зміна рівня (Change) — кожна зміна сигналу, незалежно від напрямку.

Постійний рівень (Low) — переривання триває, доки сигнал залишається низьким.

Таким чином використання таких переривань знижує навантаження на мікроконтролер. Він не витрачає час на постійне опитування входів, що звільняє обчислювальні ресурси для інших завдань.. Також забезпечують вискоу точність та зменшують ризик втрати сигналу. Апаратні переривання дозволяють обробляти швидкі імпульси, які можуть залишитися непоміченими

під час звичайного циклічного опитування пінів. Саме ця особливість і є основною для роботи з енкодерами.

Ще одна перевага - реакція на події під час затримок у коді. Навіть якщо основна програма зайнята довгими операціями або перебуває у стані очікування, переривання може бути оброблено.

Режими енергозбереження - сікроконтролер може "спати", доки не відбудеться подія на вході, що викличе переривання та розбудить пристрій.

```
volatile int count = 0;

void setup() {
  pinMode(2, INPUT); // Підключаємо пін 2
  // Налаштовуємо переривання
  attachInterrupt(digitalPinToInterrupt(2), countPulse, RISING);
  Serial.begin(9600);
}

void loop() {
  Serial.println(count); // Виводимо кількість імпульсів
  delay(1000);
}

void countPulse() {
  count++; // Інкрементуємо лічильник
}
```

Рис. 2.1. Приклад коду використання зовнішнього переривання

Для роботи з перериваннями в середовищі використовуються функції:

attachInterrupt(digitalPinToInterrupt(pin), counterPulse, mode) — Прив’язує переривання до піну.

pin — номер піну, який генеруватиме переривання.

counterPulse — функція, яка буде виконана при спрацьовуванні переривання.

mode — режим спрацьовування (RISING, FALLING, CHANGE, LOW).

detachInterrupt(pin) — вимикає переривання для вказаного піну.

За відображення інформації та зміну її впроцесі намотування відповідає RaspberryPi з сенсорним дисплеєм.

Raspberry Pi (/paɪ/) — це серія компактних одноплатних комп'ютерів (SBC), розроблених у Великій Британії. Оригінальний комп'ютер Raspberry Pi був створений фондом Raspberry Pi Foundation у співпраці з компанією

Broadcom. З 2012 року всі продукти Raspberry Pi розробляються компанією Raspberry Pi Ltd, яка спочатку була дочірньою організацією Фонду.

Проєкт Raspberry Pi спочатку орієнтувався на популяризацію викладання основ комп'ютерних наук у школах. Однак перша модель стала значно популярнішою, ніж очікувалося, і її почали використовувати за межами цільового ринку для найрізноманітніших задач, таких як робототехніка, автоматизація будинків і промислових процесів, а також ентузіастами комп'ютерних і електронних технологій. Популярність пристрою пояснюється його низькою вартістю, модульною конструкцією, відкритим дизайном, а також підтримкою стандартів HDMI і USB.

Можливості Raspberry Pi

Повноцінний комп'ютер який може використовуватися як робоча станція з встановленою операційною системою Raspberry Pi OS (на основі Debian Linux).

Підтримує роботу з програмами для програмування, офісних завдань та веб-серфінгу.

Підходить для розробки програмного забезпечення різними мовами - Python, C/C++, JavaScript, Java та ін.

Має величезну підтримку комюніті та професіоналів. Велика кількість підтриуваних та постійно оновлюваних бібліотек. Завдяки цьому можна створювати легкі застосунки та проекти. Також можливе створення важких проектів, адже платформа є дуже потужною та стрімко розвивається.

Нерідко використовується для створення розумних пристроїв: домашніх серверів, медіацентрів, систем відеоспостереження, роботів, IoT-пристроїв.

Апартно платформа дає можливість працювати з GPIO-пінами, що дозволяють підключати датчики, двигуни, реле та інші компоненти. Є підтримка I2C, SPI, UART, що розширює можливості інтеграції.

Raspberry Pi може використовуватися для запуску нейронних мереж, наприклад, за допомогою TensorFlow Lite або OpenCV.

PySide6 — це офіційна Python-реалізація Qt6 для створення графічних інтерфейсів. У поєднанні з Qt Creator, зручним інструментом для дизайну UI, можна швидко створювати застосунки.

PySide6 дозволяє створення сучасних, адаптивних інтерфейсів. Інтеграція з апаратним забезпеченням Raspberry Pi. Широкий набір віджетів для створення кнопок, форм, таблиць та інших компонентів.

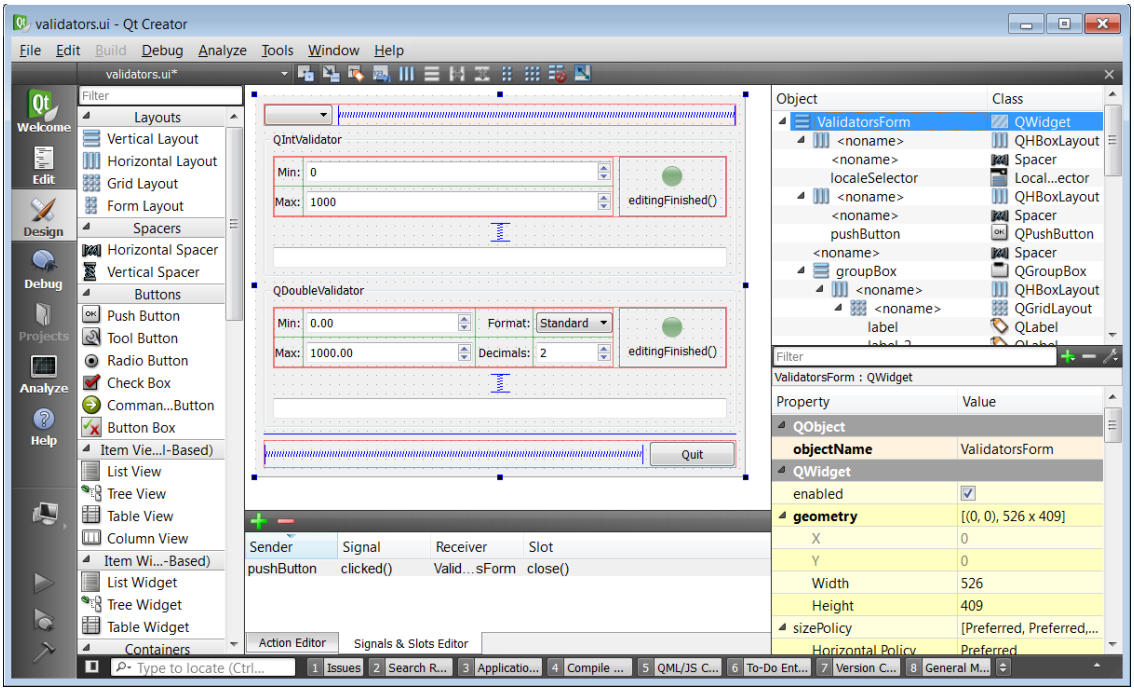


Рис. 2.2. Інтерфейс прграми QTCreator

Qt Creator — це інтегроване середовище розробки (IDE), яке входить до екосистеми Qt і призначене для створення сучасних кросплатформених застосунків. Воно підтримує розробку для різних операційних систем, таких як Windows, macOS, Linux, Android та iOS, що дозволяє використовувати єдиний код для створення додатків на різних платформах. Qt Creator надає широкий спектр можливостей для розробників, зокрема підтримку графічних інтерфейсів за допомогою Qt Designer, який дозволяє створювати багатовіконні застосунки з використанням готових або кастомізованих віджетів. Середовище забезпечує інтеграцію з бібліотеками та фреймворками, такими як Boost, OpenCV, OpenGL, а також можливість використання 2D та 3D графіки завдяки бібліотекам Qt Quick і Qt 3D.

Qt Creator має вбудовані інструменти для налагодження, профілювання та оптимізації коду, а також підтримує різні системи збірки, такі як CMake, QMake та Meson. Інтеграція із системами контролю версій (Git, Mercurial, Subversion) дозволяє легко керувати змінами в коді безпосередньо в IDE. Середовище забезпечує зручний текстовий редактор із підсвічуванням синтаксису та

автоматичним завершенням коду, що підвищує продуктивність роботи. Крім того, Qt Creator дозволяє створювати анімовані графічні інтерфейси, працювати з мультимедіа та реалізовувати інтерактивні елементи.



Рис. 2.3. Приклади інтефейсів

Інструмент підтримує розробку для вбудованих систем (embedded) і забезпечує інтеграцію з віддаленими пристроями та контейнерами. Завдяки доступності шаблонів проєктів, великій кількості документації та навчальних прикладів, Qt Creator є зручним як для початківців, так і для досвідчених розробників. Це середовище дозволяє створювати графічні, мобільні, настільні та вбудовані застосунки, що робить його універсальним і потужним інструментом для сучасної розробки.

Для керування моторами було розроблено алгоритм розрахунку швидкості, далі швидкість переаховувалась в оберти на/хв відправлялась на контролер керування кроковими двигунами і далі, цей контролер виконував ШИМ сигнал потрібних характеристик. Вихідний шим залежав від таких параметрів як: к-сть кроків на оберт двигуна, роздільна здатність драйвера, крок механізму передач(гвинта чи зубчаті колеса) потрібна шивдкість в даний момент та напрямок поворту. В процесі формуваннння швидкості також бере учать компенсуюча ланка.

На рисунку 2.2.4 продемонстрований фрагмент основго коду, панелі оператора, що відповідає за граничні обмеження швидкості, наприклад коли величина компенсації занадто низька, чи перевищею встановлений ліміт. В кожному з випадків відбувається певна дія, наприклад знижується швидкість або зменшується та інші.

```
if count < 1:
    logger.debug('encoder value < 0')
    count = 1
elif count > HIGH_COMPENSATION_LIMIT:
    logger.error(f'encoder velue too much big')
    count = HIGH_COMPENSATION_LIMIT

if 1 <= count < LOW_COMPENSATION_LIMIT:
    is_speed_changed = True
    count = LOW_COMPENSATION_LIMIT - count
    speed_k = 1 + count  / LOW_COMPENSATION_LIMIT * SPEED_UP_K
    speed_s2 = user_speed_s2 * speed_k * (1 - 0.025 * (finished_meters // 50))
elif LOW_COMPENSATION_LIMIT < count:
    is_speed_changed = True
    count -= LOW_COMPENSATION_LIMIT
    speed_k = 1 - count / LOW_COMPENSATION_LIMIT * SPEED_UP_K
    speed_s2 = user_speed_s2 * speed_k * (1 - 0.025 * (finished_meters // 50)) / 1.55
```

Рис. 2.4. Фрагмент коду Python

Проце керування швидкостю обертання двигуна відбувається таким чином. В інтефейсі встановлена певна швидкість V та залежності від положеня компенсуючої ланки величина від 0 до 500 (абстрактна величина).

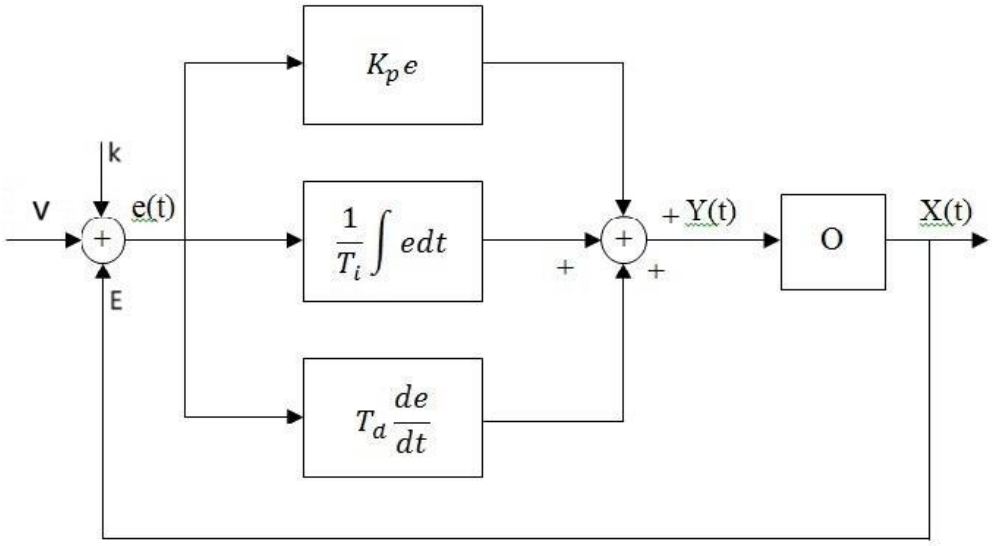


Рис. 2.5. Закон регулювання швидкості головного мотору

Ця величина перераховується в величину RPM, і далі складається з величиною заданої швидкості двигуна і на виході отримуємо фактичну

потрібну швидкість. Для мотору каретки швидкість розраховується таким чином – половина від швидкості головного приводу. Також каретка рухається на половину від заданого діаметру кожні пів оберту бабіни. Як видно на рис. 2.2.4 мотор керується законом ПІД регулювання, та містить зворотній зв'язок у вигляді енкодеру «E» (блок розрахунку вбудований в драйвер). Далі задана швидкість «V», яку встановлює оператор та коефіцієнт «k» - коефіцієнт компенсаційного пристрою. Y(t) вихідна керуюча дія. X(t) параметр, що регулюється(в даному випадку RPM).

2.3. Вибір обладнання та технологій: сенсори, виконавчі механізми, контролери, інтерфейси

Для зчитування найважливіших параметрів використані оптичні енкодери. Оптичні інкрементні енкодери мають низку переваг, які роблять їх популярним вибором у промислових, робототехнічних та інших високоточних застосуваннях. Ось ключові переваги таких енкодерів:

Оптичні енкодери використовують світлові датчики для зчитування міток на диску, що забезпечує високу роздільну здатність. Вони можуть досягати тисяч імпульсів на оберт (PPR), що забезпечує детальне відстеження положення.

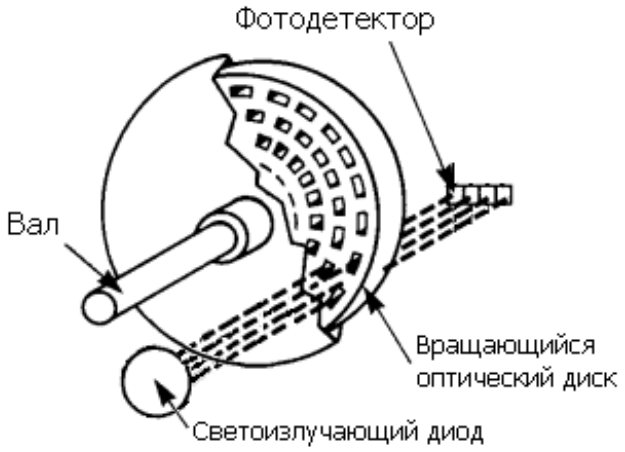


Рис. 2.6 Принцип роботи енкодера

Завдяки швидкому зчитуванню оптичного сигналу, ці енкодери можуть працювати на високих швидкостях обертання без втрати точності. Це особливо важливо для систем, які потребують швидкої реакції, наприклад, у серводвигунах або конвеєрних лініях.

Оптичні енкодери не мають безпосереднього контакту між сенсором і рухомою частиною (диском). Це значно зменшує механічний знос і збільшує довговічність пристрою. Завдяки оптичним методам зчитування, вони менш чутливі до механічних похибок, таких як люфти чи вібрації, порівняно з магнітними або механічними енкодерами.



Рис. 2.7. Вигляд використаного енкодера

Даний енкодер містить такі характеристики:

- енкодер оптичний інкрементний 1000 імпульсів
- робоча напруга: DC5-24V
- максимальна механічна швидкість 6000 об/мин
- частота: 100 kHz
- вихід: NPN відкритий колектор

Для забезпечення керування режимами стар/стоп/пауза використані промислові тактові кнопки.

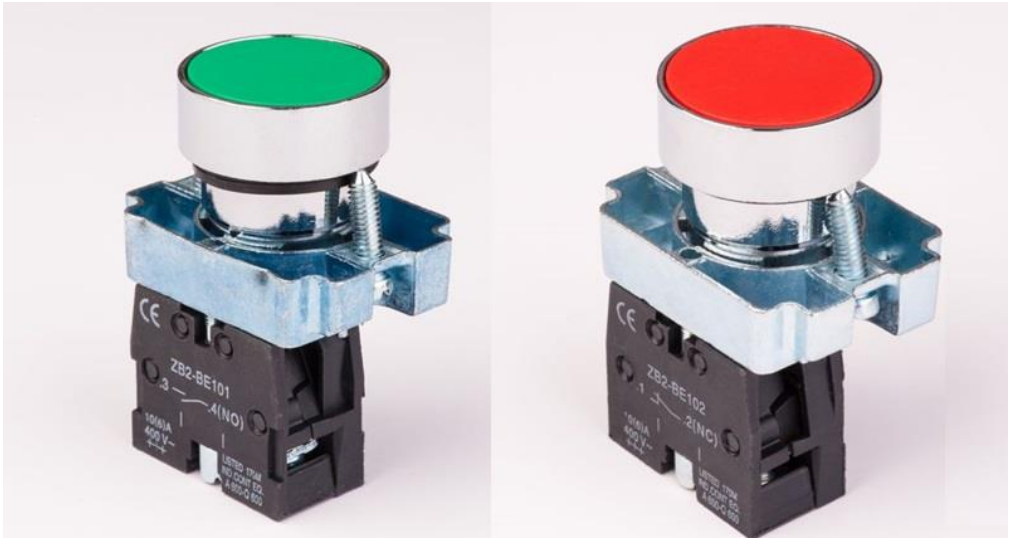


Рис. 2.8. Тактові кнопки

Для визначення нульової позиції каретки було використано індуктивний



Рис. 2.9. Індуктивний дактчик

Датчик наближення LJ12a3-4-Z/BX.

Його технічні характеристики:

Вихідний сигнал: NPN, нормально відкритий;

Вихідний струм: 300 мА;

Відстань спрацювання: 5 мм;

Частота: 300 Гц;

Живлення: 6-36 В;

Різьба: М12.

Для запезпечення роботи інтефейсу та зручності його використання. Також для швидкої розробки застосунку було обрано Raspberry pi4. Вона має величезний апаратний функціонал. Raspberry Pi 4B значно збільшує швидкість процесора, продуктивність мультимедіа, пам’ять та підключення.

Дана модифікація Raspberry Pi 4 Model B має 8 ГБ пам'яті LPDDR4-3200 SDRAM, яка ділиться між CPU і GPU. Контролер підтримує апаратне декодування H.265 / HEVC (до 4Kp60) і H.264 (до 1080p60). Це розвантажує

CPU при роботі з потоковим відео в інтернеті та перегляді «важких» відеофайлів в режимі медіацентру.



Рис. 2.10. Raspberry Pi4B

Raspberry Pi 4B включає високопродуктивний 64-розрядний чотириядерний процесор і підтримує подвійний дисплей з роздільною здатністю до 4K через пару портів micro-HDMI, двосмугову бездротову мережу 2.4/5.0 ГГц, Bluetooth 5.0, гігабітний Ethernet, 2x micro HDMI ports; 2x USB 3.0 + 2x USB 2.0 та окремий роз’єм для підключення дисплею.

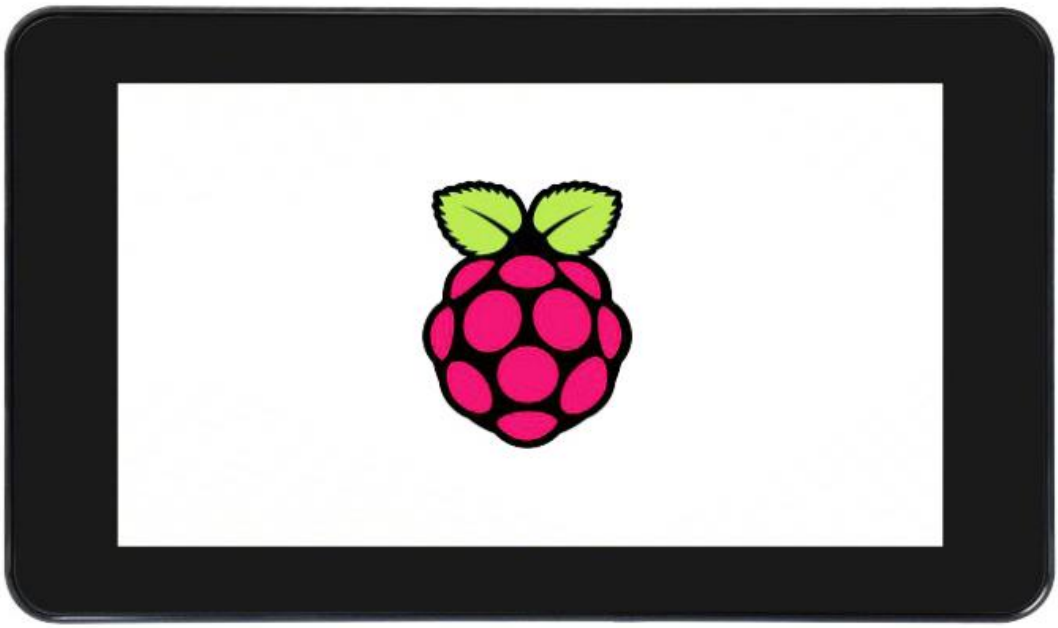


Рис. 2.11. 7.0" TFT DSI дисплей 800x480 Wavshare

Для відображення інформації та взаємодії з нею було обрано сенсорний 7 дюймовий дисплей. Містить ємнісний сенсор та достатньо яскравий екран, що забезпечую хорошу видимість в межах виробництва.

Щоб вибрати відповідний серводвигун для намотування котушки, потрібно провести аналіз та математичні розрахунки. Основні параметри, які слід врахувати, включають масу філаменту, радіус котушки, швидкість намотування та опір системи.

1. Сила супротиву прутка(пікова) F = 10 кг * 9,81 м/с^2 = 98 Н.
2. Радіус бабіни може досягати 18см: R = 0.18 м.
3. Швидкість обертання: ω = 200РМ = 20.94 град/с.
4. Маса бабіни з філаментом може досягати: m = 1.5 кг.
5. Час для розгону: t =2с.
6. Момент, створений натягуванням прутка:

$$\tau_{\text{натяг}} = F * R = 98{,}1 * 0{,}18 = 17.658\text{Н/м}$$



Рис. 2.12. Серводвигун з енкодером в корпусі

Сумарний момент включає момент, потрібний для розгону бабіни (τрозгін) і момент натягування:

$$\tau_{заг} = \tau_{розгін} + \tau_{натяг}$$

$$\tau_{заг} = 0.2546 + 17.658 = 17.9 \text{ Н/м}$$

Потужність, необхідна для підтримання руху:

$$P = \tau_{заг} * \omega = 17.9126 \cdot 20.94 = 375.05 \text{Вт}$$

З урахування запасу(K=1,5)

$$P = P * K = 75.05 \cdot 1.5 = 562.58 \text{Вт}$$

Можна зробиит висновок що потрібно двигун з потужністю понад 600W та швидкість більшою ніж 200 RPM. Також слід врахувати що це має бути серво двигун з енкодером та підтримуванням крутного моменту.

Серводвигун 90ST-M02430 являє собою привод змінного струму, який призначений для комплектування верстатного обладнання. Сервопривід має низку переваг: високий обертальний момент щодо до інерції; точність позиціювання; відстежується положення завдяки використанню зворотного зв'язку, якщо немає резонансу та вібрації; точність підтримання швидкості та стабільність обертового моменту; за малих розмірів двигуна можна отримати високу потужність; можливість швидкого розгону та гальмування. Максимальна ефективна робота серводвигуна забезпечується спільний використанням із серводрайвером 90ST-M02430.



Рис. 2.13. Серводвигун та драйвер

Драйвер 90ST-M02430 (сервопідсилювач) призначений для керування серводвигуном 90ST-M02430. Серводрайвер разом з відповідним йому серводвигуном являє собою високоточний пристрій. Для керування напрямком і кроком обертання двигуна можна використовувати не тільки контролер, а й генератор імпульсів, який необхідно під'єднати до порту CN2 драйвера.

Керування кареткою за допомогою крокового двигуна полягає в точному контролюванні його рухів, щоб переміщувати каретку по заданій траєкторії. Крокoвий двигун забезпечує точність завдяки визначеному числу кроків на оберт. Для цього було використано крокoвий двигун Nema23 23HS2430 3.0 A 28 кг/см.

Основні характеристики:

- Струм фази: 3.0 A
- Зусилля 28 кг.
- Кут кроку: 1,8 °
- Довжина двигуна: 112 мм
- Діаметр валу: 8 мм

A+ Червоний B+ Жовтий

A – Зелений B - Синій



Рис. 2.14. Крокoвий двигун каретки 23HS2430 3.0 A

Для керування цим двигуном було обрано драйвер DM556D. ифровий драйвер крокового двигуна DM556D розроблений на основі сучасних технологій управління цифрових сигнальних процесорів. Високі експлуатаційні

якості драйвера засновані потужної технології управління 32-розрядним процесором. Цифровий драйвер володіє стабільними робочими характеристиками, працює з низьким рівнем шуму, вібрації, температури; мікрокрок має 16 режимів роботи. Для КД з кроком 1,8° можна досягти максимум 51200 імпульсів на оберт (1/256); володіє технологією придушення резонансу; низьке підвищення температури значно покращує показники крокових двигунів. Драйвер має таку функцію як самоналаштування онлайн, тому він може без ручних налаштувань автоматично налаштувати необхідні й оптимальні параметри для різних двигунів і забезпечить високу продуктивність.

Напруга живлення 24-50В, підходить для управління двигунами з максимальним робочим струмом до 5,6 А (можна налаштувати від 2,1 до 5,6 А)

Особливості і характеристики:

- Захист від перевантаження по струму і напрузі;
- низьке підвищення температури, плавна робота двигуна;
- Розміри: 118x76x34 мм;
- Вага: 300 г

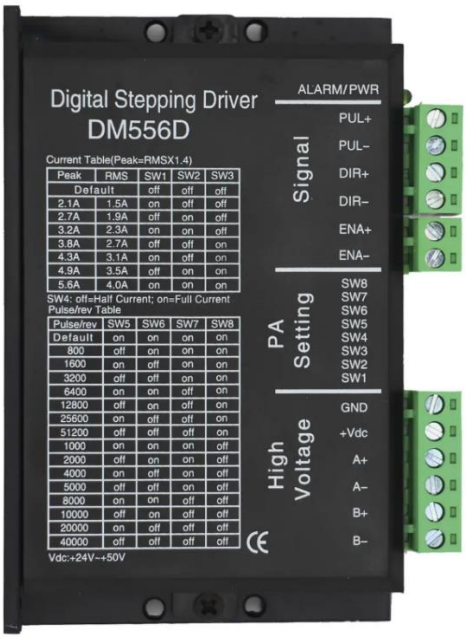


Рис. 2.15. Драйвер DM556D для двигуна Nema23

Для живлення було обрано Імпульсний блок живлення S-400-48, 48V, 8.3 A, 400W.

Високоекономічний блок живлення в захисному кожусі 48V 8,3 А 400W застосовується з метою енергозабезпечення електроприладів з постійною напругою 48 вольта. У корпус блоку вбудований вентилятор, шляхом якого здійснюється охолодження Особливості: захист від замикання, перевантаження, перенапруги. Працює в широкому діапазоні температур від 0° до +50°С. Блоки живлення широко застосовується для електроживлення сучасних електроприладів, світлодіодної техніки, ЧПК (ЧПУ), промислової автоматики та багато іншого.



Рис. 2.16. Блок живлення

Характеристики:

- Вихідна напруга: 48
- Вихідна потужність: 400W
- Струм: 8,3 А*
- Вхідна напруга обирається перемикачем
- Захист: коротке замикання / перевантаження / перенапруження
- Примусове повітряне охолодження вбудованим вентилятором DC
- ШІМ-управління
- Вбудований фільтр електромагнітних перешкод

- Авто відновлення після спрацьовування захисту по перевантаженню
- Стабільність навантаження: ±0,5%
- Регулювання Uвих ± 10%
- Ефективність (ККД) : 83%
- Час затримки при установці: 200 ms, 50 ms, 36 ms
- Опір ізоляції: :500VDC/100MΩ
- Клас захисту: IP20
- Робоча температура: -10 +50 C
- Габаритні розміри: 215×115×50мм
- Вага: 1,1 кг

3. РОЗРОБКА АПАРАТНОЇ ЧАСТИНИ ПРИСТРОЮ

3.1.Розробка електричної принципової схеми

Повна електрична принципова схема представлена в **додатку А**

Електрична принципова схема для керування кроковим двигуном в даній системі використовує кілька ключових компонентів для забезпечення безпеки та ефективної роботи. Живлення для системи подається через мережу 380 V і контролюється силовим магнітним пускачем (KM1), що дозволяє безпечно включати та вимикати живлення всього обладнання. Пускач активується котушкою, що вмикається через реле KM2, яке отримує сигнал від Raspberry Pi, що служить пультом керування оператора.

Драйвер з двигуном підключений комплектними від виробника кабелями, а вбудований енкодер, що слідкує за положенням валу, також підключений через відповідний кабель з роз'ємами для зворотного зв'язку. Реле та пін-контакти забезпечують управління двигуном, дозволяючи оператору керувати обертанням валу через Raspberry Pi, що подає відповідні сигнали.

Кроковий двигун і драйвер керуються через пін-контакти мікроконтролера ATmega2560. Сигнали step (пін 3) і direction (пін 2) визначають кроки і напрямок обертання двигуна, забезпечуючи точний контроль кожного оберту. Для безпеки використовуються також сигнальні піни: reset (пін 24) скидає двигун на початку роботи. Таке керування потрібне в разі виникнення помилки на драйвері, вона запишеться в енергонезалежну пам’ять EEPROM, щоб стерти помилкау потрібно подати сигнал reset або зайти в налаштування драйвера. Сигнал enable (пін 22) дозволяє включати або вимикати двигун.

Додатково, система має фідбек-сигнали від драйвера, які дозволяють стежити за станом роботи мотора. Наприклад, servo ready (пін 34) повідомляє про готовність сервоприводу до роботи, alarm (пін 36) сигналізує про наявність помилки, а position complete (пін 38) вказує, що двигун завершив рух до заданої позиції. Це забезпечує ефективне та безпечне управління процесом намотування філаменту, даючи можливість своєчасно реагувати на помилки або зміни в умовах роботи.

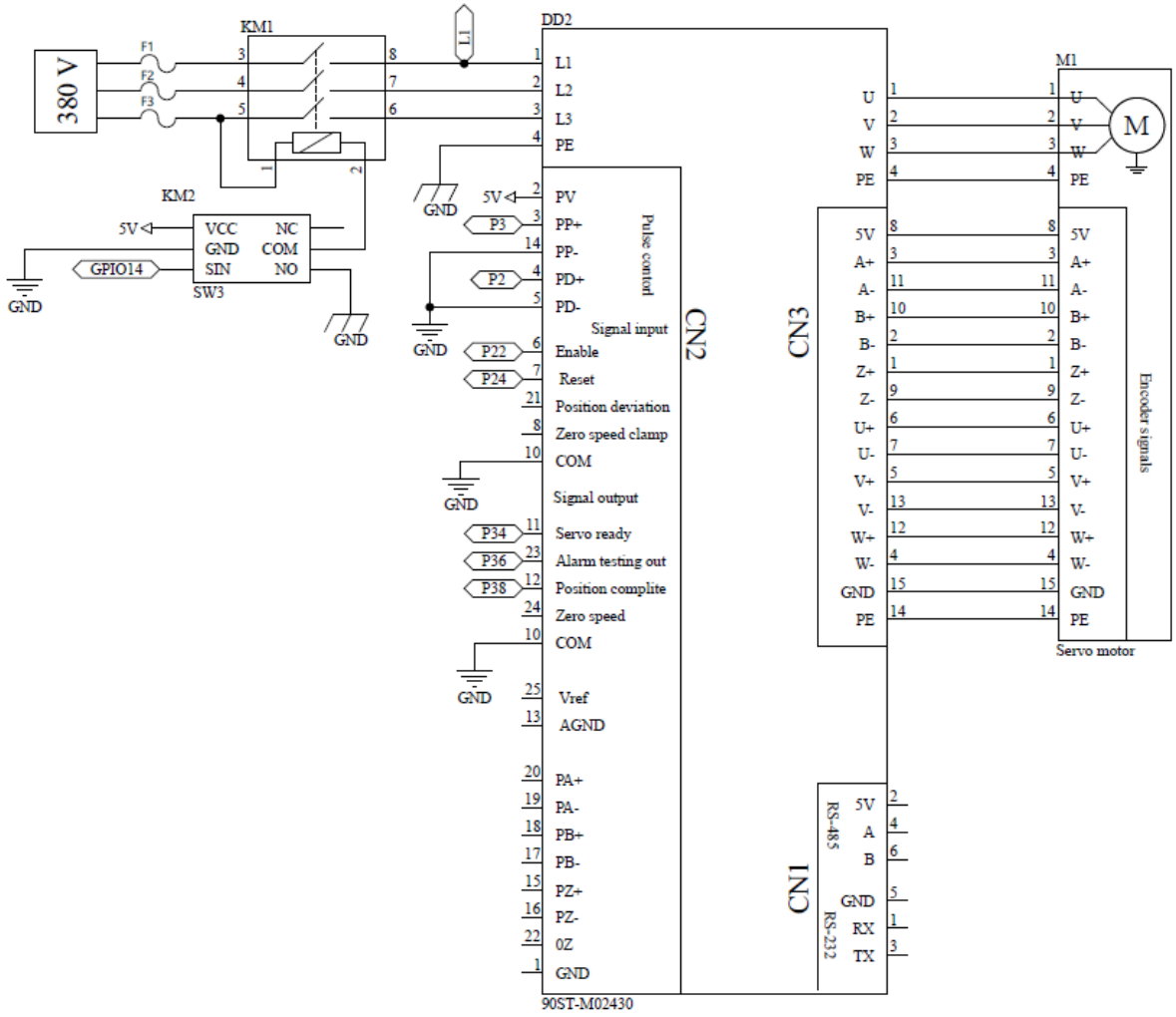


Рис. 3.1. Електрична схема підключення серводвигуна

Для підключення крокового двигуна, який не має вбудованого енкодера, використовуваного для пересування каретки, схема живлення та управління має деякі особливості. Живлення для драйвера крокового двигуна складається з джерела 24-48 V AC-DC, що забезпечує необхідну потужність для роботи двигуна.

Фаза мережі (L1) подається через силовий вимикач (KM1), що дає змогу увімкнути живлення двигунів та їх драйверів одночасно при активації пускачем. Це забезпечує синхронізацію подачі живлення на всі системи, що підключені до цієї мережі.

Щодо керування двигуном, мікроконтролер (наприклад, ATmega2560) підключений до сигналів драйвера. Сигнал PUL (пін 5) відповідає за імпульсні кроки двигуна, DIR (пін 4) визначає напрямок обертання, а ENA (пін 6) відповідає за включення або вимикання двигуна. Ці сигнали дозволяють чітко

керувати роботою крокового двигуна, визначати його рухи та контролювати його стан.

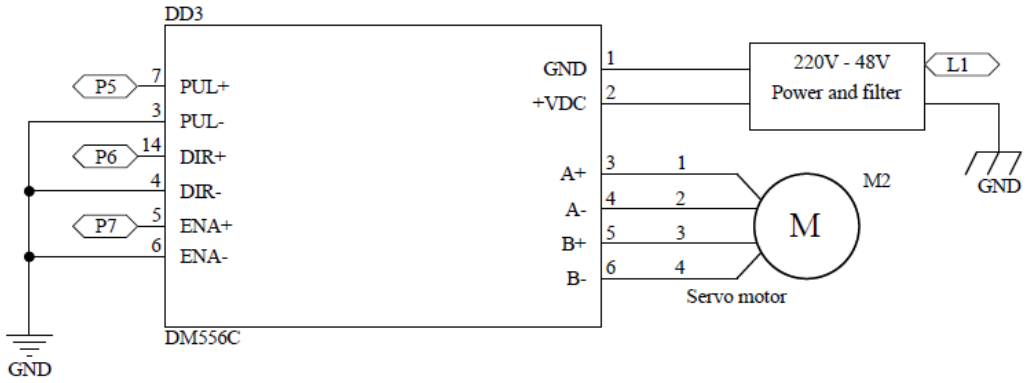


Рис. 3.2. Електрична схема крокового двигуна каретки

Панель оператора що складається з Raspberry Pi4B має підключення лише екрану та блоку живлення. Також, для безпечної роботи панель керування виконую функцію вмикання та вимикання живлення для крокових та серводвигунів. Зв’язок з Arduino відбувається за допомогою протоколу usb.

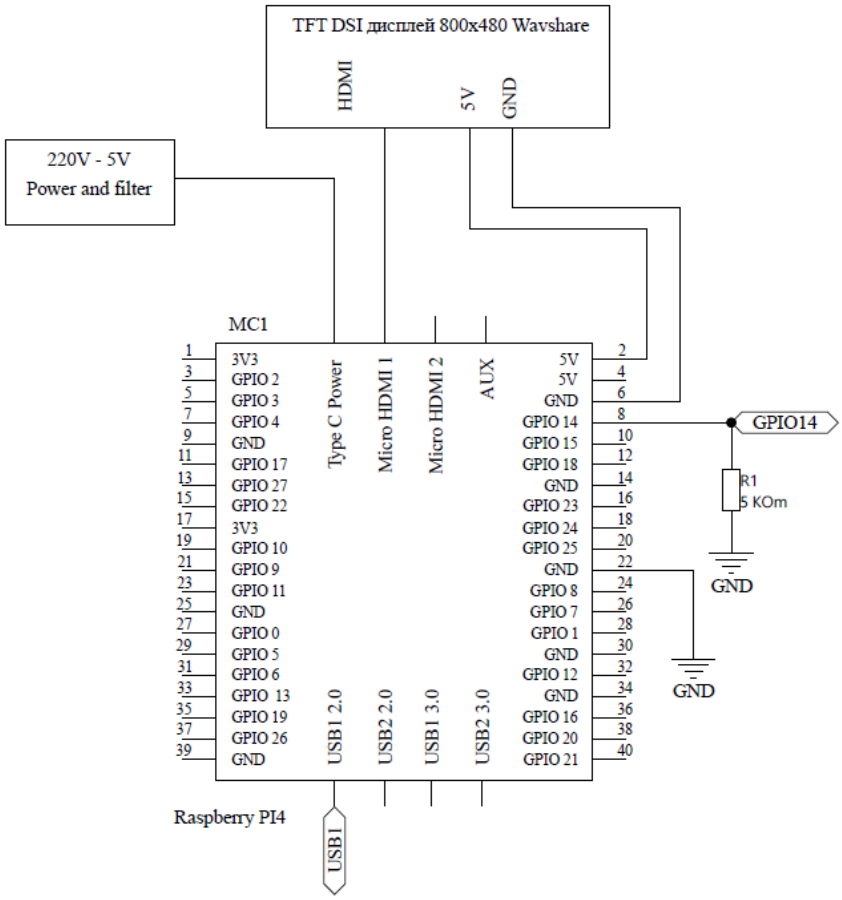


Рис. 3.3. Схема підключення Raspberry Pi4

На рис. 3.1.4 показано схему підключення цифрових входів/виходів до мікроконтролера Arduino Mega 2560. У цій схемі реалізовано використання

кнопок для подачі сигналів на вхідні порти мікроконтролера. Основою роботи схеми є підтягувальні резистори (pull-up), які забезпечують стабільність сигналів на входах.

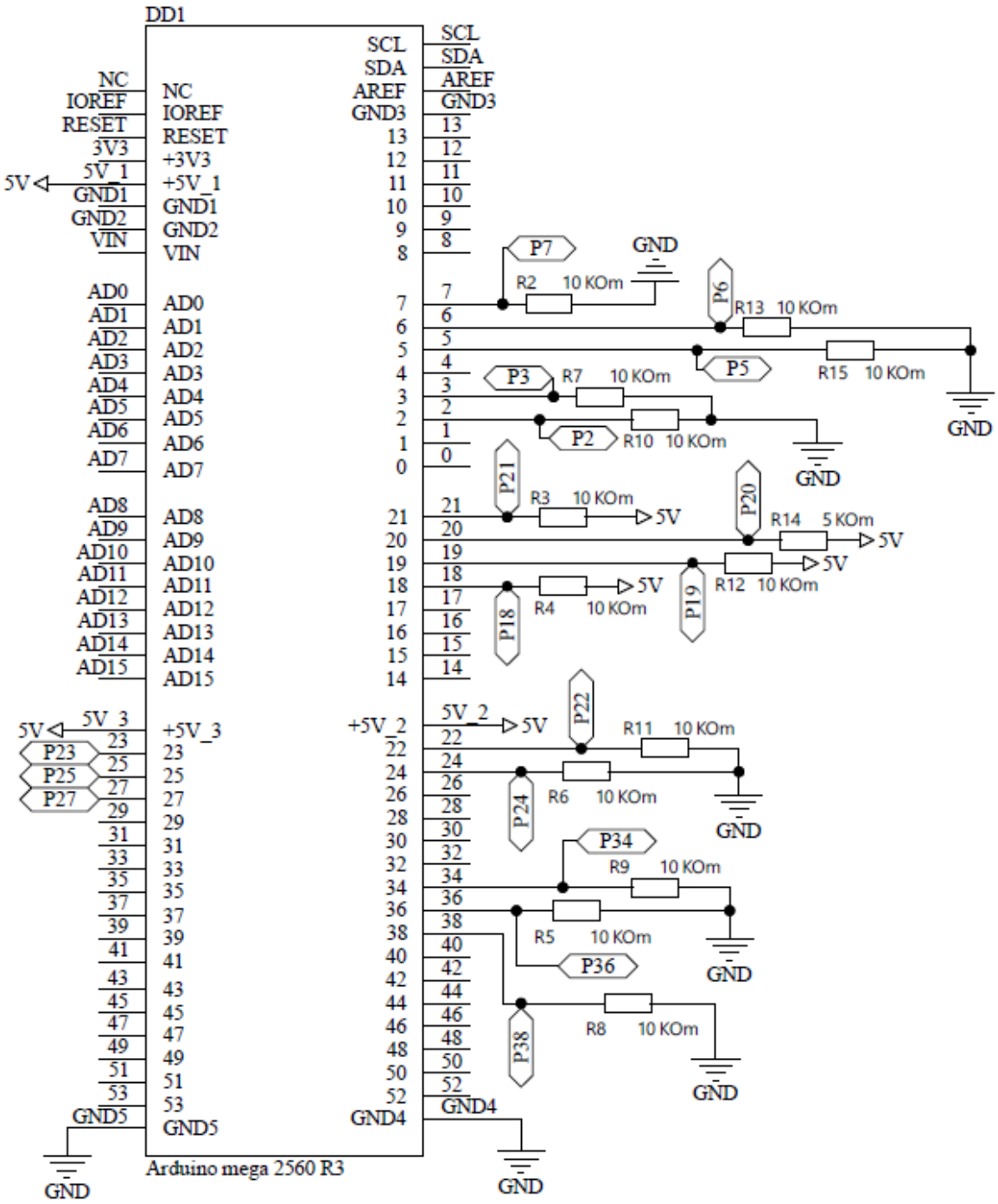


Рис. 3.4. Схема підключення пінів Arduino mega 2560

Кожен сигнал підключений до окремого цифрового порту мікроконтролера (наприклад, P7, P8, P9, P13 тощо). До цих портів через підтягувальні резистори номіналом 10 кОм подається напруга 5V або 0V, яка стабілізує рівень сигналу, коли кнопка не натиснута. Таким чином, у початковому стані на вході мікроконтролера встановлюється логічна одиниця

або нуль. Коли користувач натискає кнопку, вхідний сигнал замикається і на порті сигнал змінюється, якщо був 0 то стане 1 і навпаки. Таким чином працює апаратне зовнішнє переривання.

Усе це дозволяє забезпечити коректну роботу всієї системи. Схема передбачає підключення кількох кнопок до портів P25 та P23 мікроконтролера без використання переривань, це дозволяє незалежно зчитувати стан кожної кнопки.

Ця схема використана для реалізації керування кроковими двигунами та зчитування інформації з інших пристроїв, як кнопки енкодери та індуктивний датчик для задання нульовової позиції каретки. Завдяки використанню підтягувальних резисторів забезпечується стабільність і надійність сигналів, а мікроконтролер може обробляти зміни стану кнопок у реальному часі. Така система є простою у реалізації та забезпечує надійне управління електронними пристроями.

3.2.Розробка структурної схеми пристрою

Структурна схема — схема, яка визначає основні функціональні частини виробу, їх взаємозв'язки та призначення. Під функціональною частиною розуміють складову частину схеми: елемент, пристрій, функціональну групу, функціональну ланку, дана описує автоматизовану систему керування намотування філаменту.

Вхідні сигнали надходять від двох енкодерів, що вимірюють довжину намотаного філаменту(енкодер 1) та кут положення компенсатора(енкодер 2) та індуктивного датчика для визначення нульового положення або «нуля» котушки. Тактові кнопки "Старт/Стоп", які дозволяють вручну запускати чи зупиняти процес намотування.

Усі ці сигнали передаються до центрального контролера Atmega2560, який обробляє їх і забезпечує керування драйверами двигунів а також відправлення інформації на головний пристрій – міні ПК. Raspberry Pi 4B, підключений до Atmega2560 через USB, служить керуючим модулем обчислення швидкості та інших подій, адже Atmega2560 є вузлом, що виконує команди головного контролера та передає інформацію з датчиків назад. Далі одноплатний ПК

передає цю інформацію на дисплей та отримує інформацію про натискання на сенсор(кнопки на панелі оператора) через інтерфейс DSI для візуалізації даних системи та виконання дій. Atmega2560 також керує драйвером серводвигуна, підключеним до серводвигуна з енкодером, і драйвером крокового двигуна для забезпечення точного руху.

Живлення системи забезпечується блоками живлення: один з них перетворює 220В у 5В для живлення Raspberry Pi, Atmega2560, кнопок, датчиків та одноканального реле, а інший перетворює 220В у 48В для драйверу крокового двигуна. Одноканальне реле, кероване Raspberry, вмикає магнітний силовий пускач, який забезпечує подачу трифазної напруги 380В на приводи. У схемі чітко розрізняються зв’язки типу "Живлення" та "Сигнал" за стилем ліній, що допомагає зрозуміти, як передається електроживлення та інформація між компонентами.

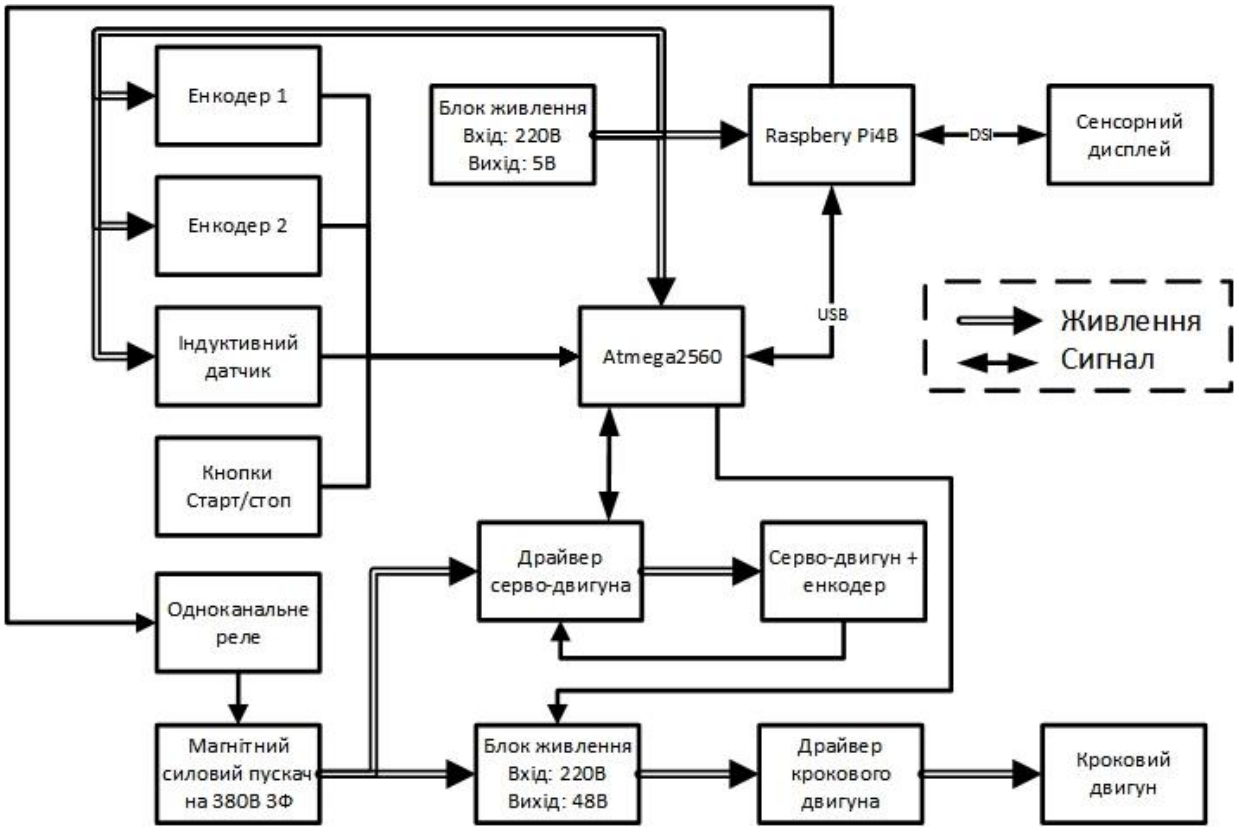


Рис. 3.5. Спрощена структурна схема АСУ намотування філаменту

3.3.Розробка функціональної схеми системи

Схема відображає автоматизовану систему намотування філаменту, що включає кілька ключових механізмів і датчиків для забезпечення безперервного процесу. Основні елементи системи включають вхід філаменту з екструдера, компенсуючий механізм для акумулювання зайвого філаменту, механізм намотування, систему стабілізації, контролери руху каретки та насадки для котушки. Система використовує зворотний зв'язок через енкодери, індуктивний датчик, кнопки керування та контролери для забезпечення точності й надійності роботи.

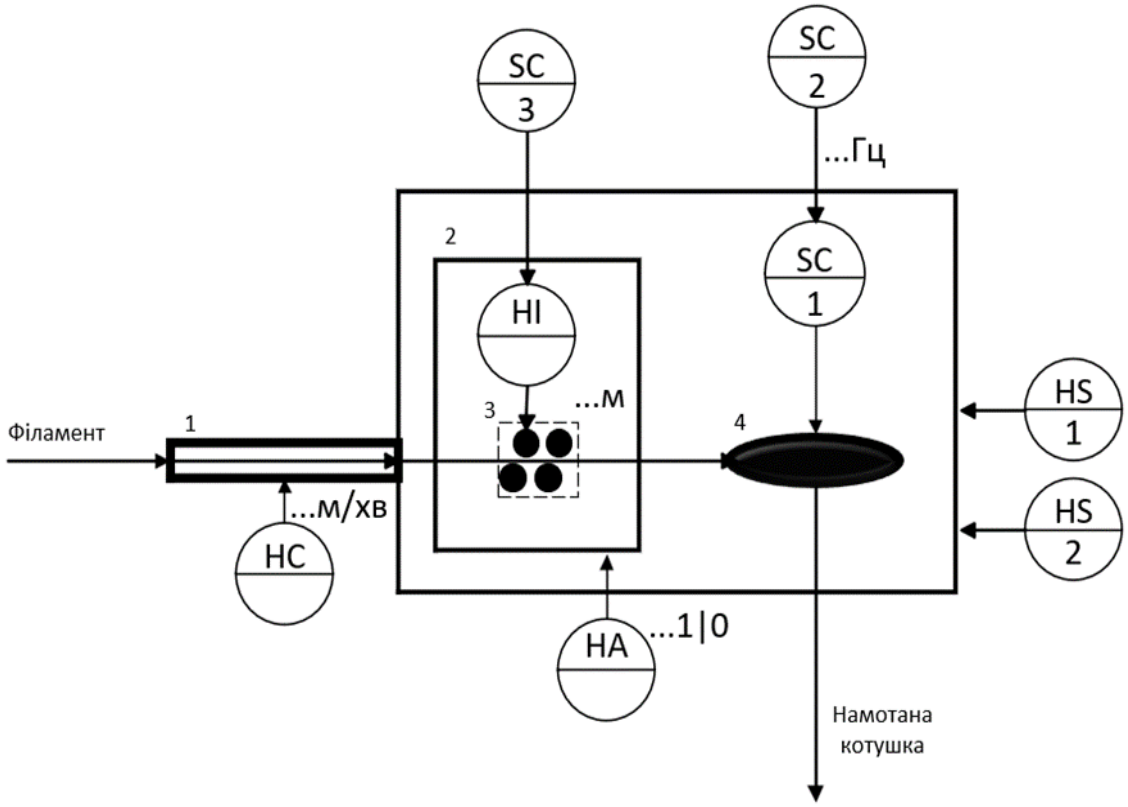


Рис. 3.6. Спрощена функціональна схема автоматизації

Процес починається з надходження охолодженого філаменту від екструдера до компенсуючого механізму (1). Цей механізм акумулює філамент за допомогою системи блоків і компенсує його під час заміни котушки, запобігаючи утворенню провисання. Кут компенсуючої ланки вимірюється енкодером HC, який впливає на швидкість намотувального пристрою для підтримання натягу.

Каретка забезпечує рівномірне укладання філаменту виток до витка на котушку. Кроковий двигун (SC3) приводить каретку в зворотно-поступальний

рух, враховуючи діаметр філаменту (наприклад, 1,75 мм) і задану довжину кроку. Індуктивний датчик (HA) визначає нульову позицію каретки під час повернення в початкову точку. Це дозволяє уникати зсуву витків та забезпечує рівномірний розподіл філаменту.

Котушка, встановлена на спеціальну насадку (4), обертається завдяки мотору SC(1). Енкодер SC(2) контролює оберти, забезпечуючи узгодженість із швидкістю намотування та кареткою що вкладає витки. Стабільність процесу підтримується системою блоків (3), один із роликів якої оснащений енкодером HI для вимірювання довжини намотаного філаменту. Це дозволяє контролеру зупинити намотування після досягнення заданої довжини (наприклад, 800 м, 1000 м або 1500 м).

Система керується кнопками "Старт" (HS(1)) та "Стоп" (HS(2)), які дозволяють почати або завершити процес намотування, а також здійснити екстрену зупинку у разі потреби.

3.4.Розробка схеми алгоритму роботи панелі оператора

Схема і алгоритм роботи пристрою забезпечують безперервний процес намотування філаменту на котушку, контроль його натягу та дотримання необхідної ваги котушки. Алгоритм роботи складається етапів, що будуть описані далі. Слід зазначити, що представлена схема являється дещо спрощеною, адже повна реалізація схеми є комерційною таємницею і може зашкодити користувачу пристрою.

Весь алгоритм роботи програмної частини можна умовно позділити на чотири частини:

1. Ініціалізація
2. Підключення до Atmega2560
3. Режим очікування
4. Режим роботи

Автоматизована система намотування філаменту розпочинає свою роботу з етапу ініціалізації. Після запуску системи на базі Raspberry Pi відбувається завантаження операційної системи та основного програмного забезпечення, яке здійснює управління всім процесом. На цьому етапі відбувається зчитування

збережених параметрів із пам’яті (EEPROM), зокрема, встановлених швидкостей обертів, довжини філаменту, кроку для намотування та важливі для початкового запуску файли програми а також бібліотека логування та її параметри. Якщо виявляються помилки у завантажених даних або пошкоджені файли програми, синтаксичні помилки, система замінює некоректні параметри на значення за замовчуванням, записує інформацію в файл логування. В решті випадків намагається перезавантажитись.

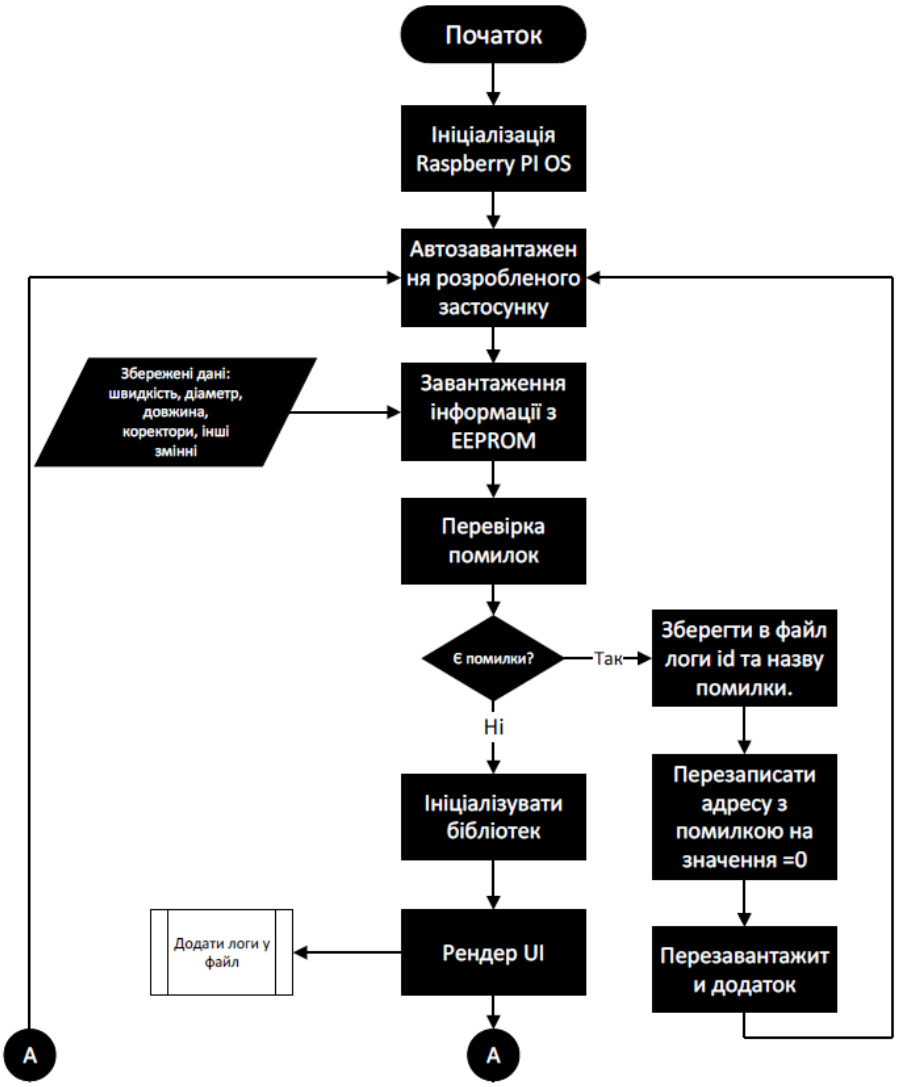


Рис. 3.7. Частина(ініціалізація) Алгоритму роботи пристрою

Далі перевіряється з'єднання з контролером Atmega2560 через USB, використовуючи обмін командами для перевірки зв’язку. Як тільки контролер Atmega2560 ініціалізувався він знаходиться в режимі пошуку головного пристрою. Він посилає спеціальний хеш-ключ щосекунди, доти доки не отримає у відповідь запрограмовано команду. Якщо така команда була

отримана мікроконтролер відправляє у відповідь «OK» і переходить до виконання основного циклу Loop де і реалізований основний функціонал. Схема-алгоритм цієї частини представлена на рис. 3.4.2.

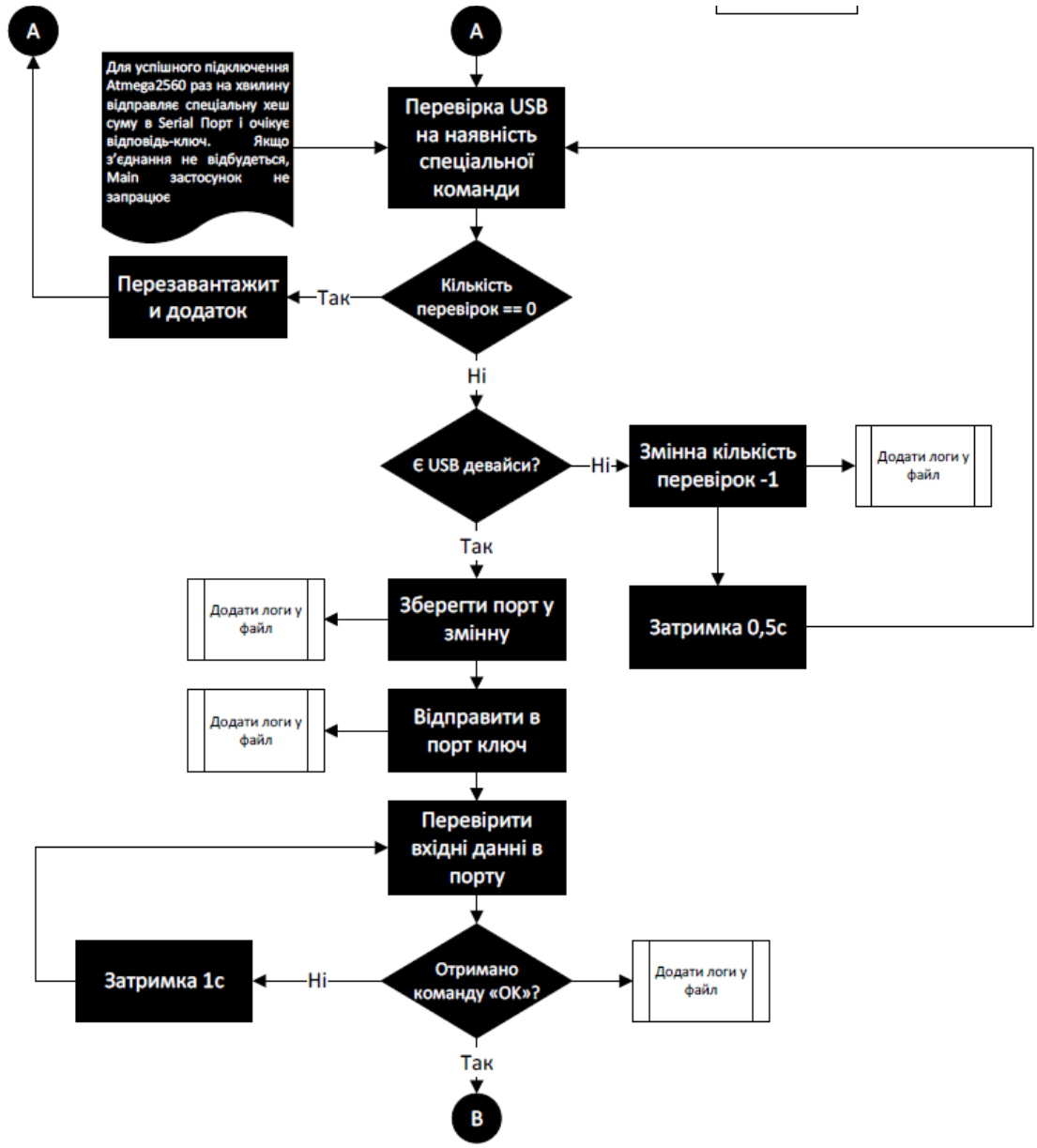


Рис. 3.8. Частина схеми-алгоритму підключення до Atmega2560

Після успішної ініціалізації система переходить у режим готовності до роботи. Автоматично відбувається відправка головним контролером команди "Homing". Каретки починає шукати нульове положення відповідної котушки. Каретка їде в бік індуктивного датчика, і як тільки на піні мікроконтролера буде 1, каретка зупиниться та запамятає це положення як початкове. Далі розблокується керування пристроєм і оператор може виконати налаштування за допомогою сенсорного дисплея. Після цього система переходить у режим очікування команди "Старт", яку можна активувати через фізичну тактову

кнопку. Від тепер система знаходиться в режим і очікування. Основно програма розділена на три потоки: моніторинг «serial» порту USB на наявність нових команд отриманих від мікроконтролера Atmega2560. Якщо такі повідомлення було отриманго, відбувається декодування повідомлення, розбиття на строки та визначення змісту отриманого повідомлення. Далі якщо повідомлення містило певні ключові команди, викликаються відповідні функції описані в коді програми. Поток який реєструє натискання на сенсорний дисплей. При натисканні на сенсорному екрані певних віртуальних кнопок відбувались виклики функцій які відповідали за налаштування пристрорю намотування.

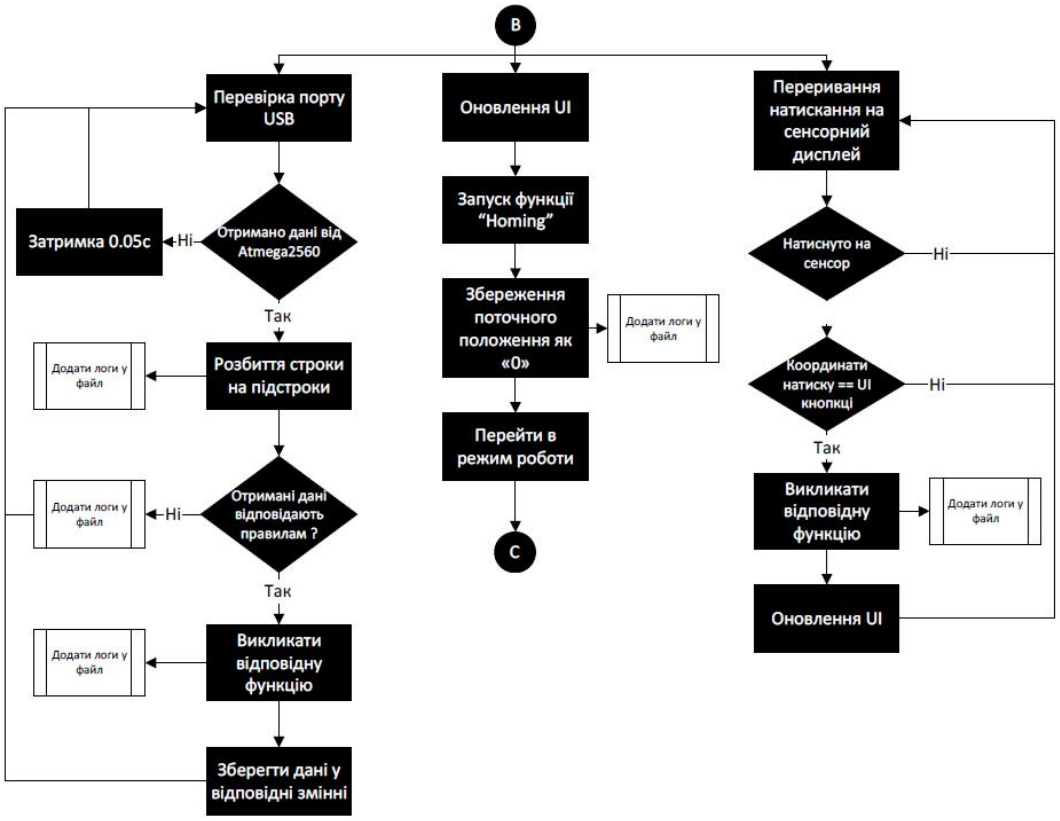


Рис. 3.9. Частина алгоритму роботи – «потоки моніторингу»

І основний цикл який працює неперервно доки не буде намотано задана кількість філаменту, натиснута кнопка зупинити процес намотування або виникнення помилки. Процес починається з оновлення інтерфейсу користувача (UI), що забезпечує відображення актуального стану системи.

Панель оператора автоматично відправляє команду зупинки двигунів, для уникненя перебоїв у комунікації та забезпечення безпеки, а також додає відповідну інформацію до лог-файлу. Далі у циклі перевіряється, чи отримана

команда "Старт". Якщо команда запуску відсутня, система продовжує оновлювати інтерфейс, але у разі надходження команди переходить до перевірки, чи досягнуто заданої довжини намотування філаменту. Якщо задану довжину вже намотано, система завершує роботу та записує цю подію до логів.

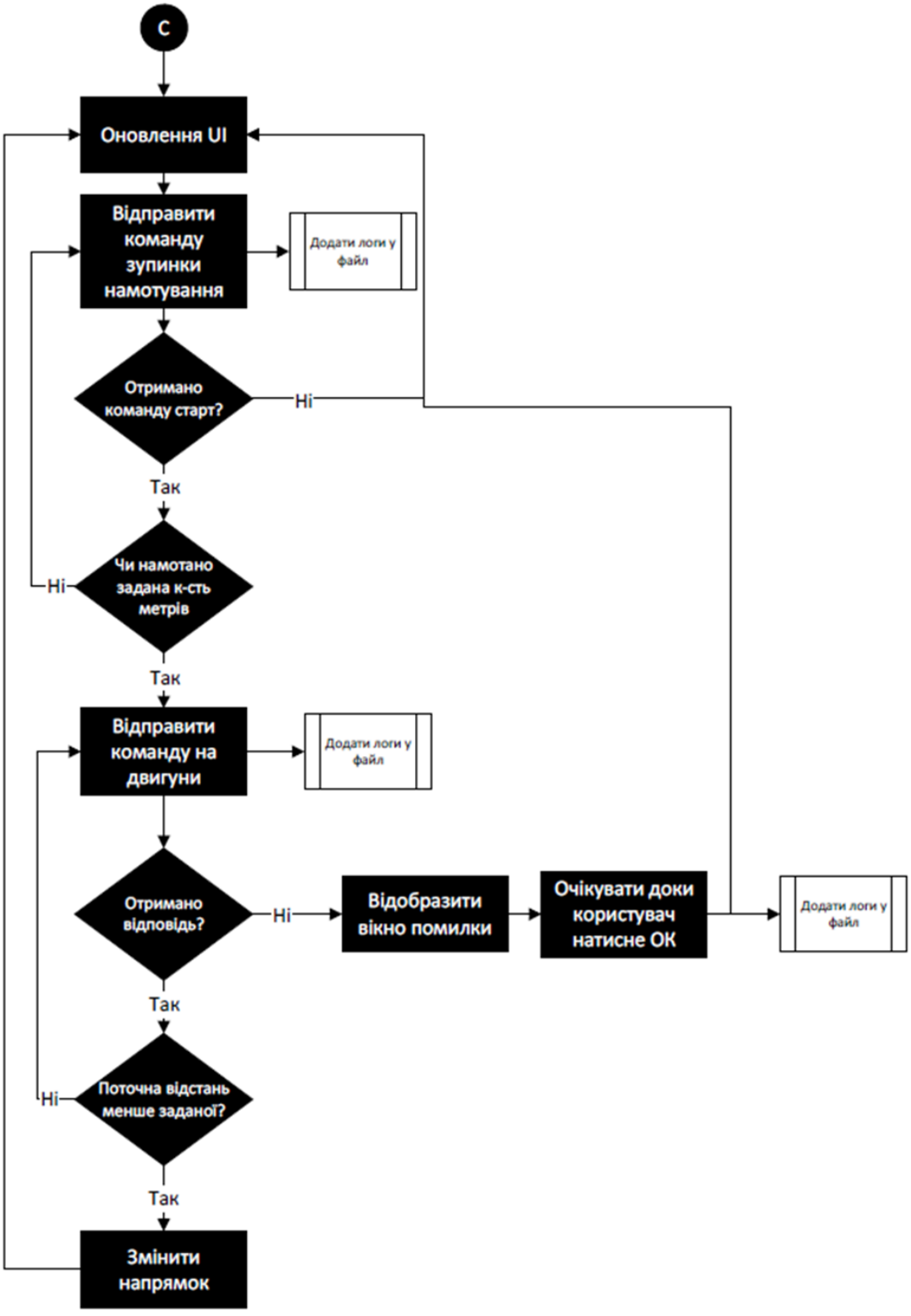


Рис. 3.10. Частина(робоча) алгоритму

У випадку, коли задану довжину ще не досягнуто, система надсилає команду на двигуни для продовження роботи. Водночас перевіряється, чи

отримано відповідь від двигунів. Якщо відповіді немає тричі система відображає вікно помилки на інтерфейсі користувача і переходить у режим очікування, поки оператор не натисне "ОК". Ця подія також записується до логів. Якщо відповідь від двигунів отримана, перевіряється поточна відстань, яку пройшла каретка та довжина намотаного філаменту. Якщо відстань менша за задану, система змінює напрямок руху каретки для забезпечення рівномірного намотування.

Після зміни напрямку або підтвердження роботи двигунів система повертається до циклу перевірки інтерфейсу користувача та параметрів роботи, продовжуючи процес намотування до отримання команди зупинки або досягнення заданої довжини. Весь процес супроводжується записом даних про події, команди та помилки у лог-файли, що забезпечує можливість діагностики та аналізу роботи системи.

Слід зазначити, що схема-алгоритм представлена у спрощеному вигляді. Алгоритм роботи системи забезпечує безперервний контроль параметрів у реальному часі: довжини філаменту, швидкості обертання котушки, натягу філаменту та положення каретки.

Завдяки цьому система підтримує стабільний процес намотування, забезпечує високу точність і якість укладання витків, а також запобігає пошкодженню філаменту. Усі налаштування, результати роботи та помилки зберігаються у логах для подальшого аналізу, що дозволяє оптимізувати процес і усувати можливі недоліки.

3.5.Опис алгоритму роботи оператора

Даний алгоритм створений для операторів лінії намотування філаменту. Це покрокова інструкція, що потрібно виконувати та як реагувати на ті чі інші події.

1. Підготовка до роботи включає в себе перевірку фізичного стан системи:

Переконатися, що всі компоненти встановлені та підключені, присутнє заземлення та відсутні оголоне провода і т.д.

Переконатися унаявності живлення та відсутності механічних пошкоджень корпусу чи інших деталей пристрою.

Обовязково переконатись чи не блокує нічого фізечне переміщення вкладаючої каретки.

Увімкнути живлення пристрою. Повинна відбутись звукова ініціалізації а також ввімкнутись дисплей. На дисплеї повинна бути картинка різнокольорових ліній.

Якщо на данному етапі не виникло проблем тоді, дочекатися завантаження Raspberry Pi та оновлення інтерфейсу користувача.

2. Ініціалізація та налаштування

На сенсорному з’явиться меню оператора. Пристій автоматично перейде до виконання “Homing”. Після завершення контролер встановить каретку у нульову позицію ("дом").

Далі потрібно переконатись що всі параметри задані(окрім поточної довджини), якщо ні задати їх вручну, при зміні параметрів їх значення запишеться у пам'ять. Ввести довжину філаменту для намотування (наприклад, 800 м, 1000 м, 1500 м), крок каретки(стандартно 1,75) швидкість намотування.

Якщо всі параметри в нормі, а на дисплеї немає повідомлень про помилку, потрбно демонтувати котушку, якщо вона є та встановити пусту.

Встановіть катушку на вал пристроою та зафіксуйте гайкою. Далі переконатися, що котушка правильно встановлена, а філамент заправлений у компенсуючий механізм і стабілізуючі ролики.

3. Запуск процесу

Натиснути кнопку "Старт" фізичну кнопку "HS1".

Система перевірить отримання команди "Старт" і розпочне намотування філаменту:

Каретка почне рух для укладки витків. Котушка почне обертатися з регульованою швидкістю.

4. Моніторинг процесу

Слідкувати за станом системи через інтерфейс користувача, перевіряти поточну довжину намотаного філаменту та помилки за наявності.

Слідкувати за натягом філаменту (через стан компенсуючого механізму). Обов’язково контролювати стабілізуючі ролики, адже в цьому місці можуть

бути значні коливання що можуть спричинити заломування або петлі та в подальшому розриви.

У разі виникнення помилок:

Відреагувати на повідомлення про помилку, відображене на дисплеї.

Натиснути "ОК" для підтвердження усунення проблеми.

За необхідності провести корекцію, наприклад, перевірити встановленні параметри та візульно оглянути двигуни чи датчики.

5. Завершення процесу

Система автоматично зупиниться після досягнення заданої довжини філаменту, що також буде супроводжене звуковим сигналом.

В такому випадку потрібно відрізати філамент та зафіксувати на котушці. Відкрутити фіксуючу гайку та замінити котушки. Далі повторити всі вищеописані дії ще раз.

Якщо необхідно завершити процес раніше, натиснути кнопку "Стоп" (HS2) для зупинки.

7. Вимкнення системи

Вимкнути живлення на панелі керування. Очистити систему від залишків філаменту чи пилу.

За потреби підготувати систему до наступного циклу намотування.

Ключові дії оператора

Підготовка системи: Увімкнення, перевірка та налаштування.

Запуск: Активація процесу через "Старт".

Контроль: Спостереження за параметрами системи, реагування на помилки.

Завершення: Зупинка, зняття котушки, аналіз результатів.

Підготовка до наступного циклу: Очищення, налаштування.

Цей алгоритм роботи оператора забезпечує ефективність і безпеку роботи системи.

4. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Розробка графічного інтерфейсу та алгоритмів обміну даними на базі Raspberry Pi

Інтерфейс системи намотування філаменту створений таким чином, щоб забезпечити зручність управління процесом для оператора. У верхній частині розташовані інформаційні поля, які відображають основні параметри роботи. Поле "Задано М" дозволяє оператору встановлювати необхідну довжину філаменту для намотування в метрах, тоді як поле "Фактичні М" відображає поточну довжину вже намотаного філаменту, яка оновлюється в режимі реального часу, та скидається до 0 після завершення намотування. Поруч знаходиться поле "Швидкість", яке показує поточну швидкість намотування.

Нижче розташовані поля для налаштування діаметра та ширини філаменту, зазначених у міліметрах. Поле "Діаметр ММ" використовується для введення діаметра філаменту, що дозволяє точно враховувати його характеристики під час намотування та впливає на крок, з якою рухаєтьтся вкладаюча каретка. Поле "Ширина ММ" відповідає за налаштування ширини котушки, що впливає на загальну відстань від краю котушки до краю.

Під цими полями знаходиться набір кнопок для регулювання параметрів. Оператор може вибрати крок зміни значення за допомогою кнопок із числами 10, 1, 0.1 або 0.01, залежно від необхідної точності. За допомогою кнопок "+" і "-" можна збільшувати або зменшувати значення в обраному кроці.

Внизу інтерфейсу розташована кнопка "Паркування", яка, відповідає за приведення системи у початковий стан та викликає сповіщення про те, що зупинити неможливо, натисніть фізичну кнопку.

Інтерфейс побудований таким чином, щоб оператор міг легко моніторити всі ключові параметри процесу, швидко вносити зміни у налаштування та контролювати процес намотування філаменту. Простий і логічний дизайн забезпечує швидкий доступ до всіх необхідних функцій, що робить роботу з системою інтуїтивно зрозумілою.



Рис. 4.1. Вигляд впровадженого дизайну

Розроблений застосунок містить не складну структуру. Всередині проекту є основна програма main.py яка запускає потрібні застосунки та імпортує потрібні файли. Папка “res” містить шрифти деякі зображення та інше. Далі папка “ui” містить інтерпритацію вікон та їх вмісту бібліотекою pyside6. Та папка “windows” яка містить файли, в яких ініціалізований клас вікна та основні об’єкти, змінні та методи вікон, які розробник може використовувати.

```python
import sys
from PySide6.QtWidgets import QApplication
from windows.main_window import MainWindow
from logs import logger

logger.info('init app')
app = QApplication(sys.argv)
app.setQuitOnLastWindowClosed(True)
widget = MainWindow()

1 usage
def main():
    sys.exit(app.exec())


if __name__ == "__main__":
    main()

logger.info('finish init main.py')
```

Рис. 4.2. Лістинг основного файлу main.py

Головний файл проекту є main.py(рис. 4.1.2). Він реалізує базову ініціалізацію графічного застосунку на основі бібліотеки PySide6. У коді здійснюється імпорт необхідних модулів, включаючи `sys`, клас `QApplication` з PySide6 для створення GUI-застосунку, клас основного вікна `MainWindow` з модуля `windows.main_window` і логер для запису подій із модуля `logs`. Спочатку логер фіксує початок ініціалізації програми. Далі створюється об'єкт `QApplication`, який відповідає за управління життєвим циклом застосунку. Організовано умову завершення роботи при закритті останнього вікна. Створюється екземпляр основного вікна програми `MainWindow`, яке відображатиметься користувачу.

Функція `main()` виконує запуск застосунку через метод `app.exec()`, і вихід з програми забезпечується через `sys.exit()`. Основна точка входу програми, позначена через `if __name__ == "__main__"`, викликає функцію `main()`, запускаючи застосунок. Після виконання app.exec() відбувається рендер класу MainWindow з файлу main_window.py Після завершення роботи логер фіксує завершення ініціалізації. Код відповідає за базовий запуск графічного інтерфейсу користувача.

Всередині папок windows є два файли вікон. Файл головного вікна та вікна попереджень. Всередині них було реалізовано весь функціонал застосунку. Додатково логіка та деякі функції були винесені у файл business.py. Далі приклад коду вікна сповіщень.

```
7   class AlertWindow(QMainWindow):
8       def __init__(self, message: str, title: str, parent=None,
9                    is_ok_button_hide=True, is_cancel_button_hide=True,
10                   ok_button_function=lambda: None, cancel_button_function=lambda: None):
11          super().__init__(parent)
12          if IS_FULL_SCREEN:
13              self.showFullScreen()
14
15          self.ui = AlertUI()
16          self.ui.setupUi(self)
17          self.ui.info_label.setText(message)
18          self.setWindowTitle(title)
19          self.ok_button_function = ok_button_function
20          self.cancel_button_function = cancel_button_function
```

Рис. 4.3. Фрагмент коду

Дане вікно було створене в редакторі QTCreator в поєднанні з бібліотекою PySide6. Воно потрібне для відображення помилок та сповіщень, адже системні вікна та функції linux були відключен та заблоковані. Це було зроблено для безпеки пристрою та з метою унеможливити сторонні втручання в код застосунку. В результаті компіляції заготовки в редакторі було отримано файли в папках “ui” та “windows”. Наприклад початковим скомпільованим кодом ініціалізації вікна сповіщення на рис. 4.1.3.

Додатково було реалізовано функціонал виклику вікна з можливістю вимикати або вмикати відображення певних елементів. А також додано call-back для виклику функцій при натисканні на певні кнопки всередині UI.

```
3 usages
class AlertWindow(QMainWindow):
    def __init__(self, message: str, title: str, parent=None,
                 is_ok_button_hide=True, is_cancel_button_hide=True,
                 ok_button_function=lambda: None, cancel_button_function=lambda: None):
        super().__init__(parent)
        if IS_FULL_SCREEN:
            self.showFullScreen()

        self.ui = AlertUI()
        self.ui.setupUi(self)
        self.ui.info_label.setText(message)
        self.setWindowTitle(title)
        self.ok_button_function = ok_button_function
        self.cancel_button_function = cancel_button_function
        self.ui.close_button.setHidden(is_ok_button_hide)
        self.ui.cancelButton.setHidden(is_cancel_button_hide)
        self.ui.close_button.clicked.connect(self.ok_button_action)
        self.ui.cancelButton.clicked.connect(self.cancel_button_action)

    1 usage
    def ok_button_action(self, _=None):
        self.ok_button_function()
        self.close()

    1 usage
    def cancel_button_action(self, _=None):
        self.cancel_button_function()
        self.close()
```

Рис. 4.4. Повна реалізація вікна сповіщення

Найскладнішою частиною проекту є файл реалізації main вікна. Для коректної взаємодії та зручностей написання та відлагодження функції проекту були реалізовані в різних файлах. Попередньо всі файли імпортовано безпосередньо в код main вікна.

На початку ініціалізовано головне вікно програми, налаштовує необхідні модулі та бібліотеки для роботи.

```
3   import RPi.GPIO as GPIO
4   from PySide6.QtCore import QTimer
5   from PySide6.QtWidgets import QMainWindow, QApplication
6
7   import settings
8   from business import *
9   from ui.main_ui import Ui_MainWindow as MainUI
10  from windows.alert_window import AlertWindow
11
12  logger.info('init main_windows.py')
13  load_dotenv()
```

Рис. 4.5. Імпортовані файли в main_window.py

У коді імпортується модуль RPi.GPIO, що забезпечує доступ до GPIO-портів Raspberry Pi, а також компоненти бібліотеки PySide6, такі як QTimer, QMainWindow та QApplication, які потрібні для створення графічного інтерфейсу та роботи з таймерами.

Додатково імпортуються модулі settings, в ньому представлені налаштування програми для полегшення розробки, деякі змінні для відлагодження. Також логіка основного функціоналу, обробка інформації та внутрішнього протоколу передачі даних реалізовано всередині модуля business.

З файлу ui.main_ui завантажується інтерфейс головного вікна програми Ui_MainWindow, перейменований як MainUI, а з модуля windows.alert_window — клас AlertWindow, що забезпечує функціонал вікон попереджень.

На початку роботи логер записує повідомлення init main_windows.py, яке фіксує факт ініціалізації головного вікна. Функція load_dotenv() завантажує змінні середовища з .env файлу, що дозволяє використовувати конфіденційні дані, такі як ключі або налаштування, безпосередньо у програмі. Таким чином, код забезпечує налаштування базових компонентів для запуску програми, включаючи інтерфейс, доступ до апаратної частини Raspberry Pi та обробку подій.

Клас MainWindow, що є нащадком класу QMainWindow, реалізує основне вікно графічного застосунку, забезпечуючи його функціонал, інтерактивність

та взаємодію з користувачем і апаратними компонентами. У конструкторі класу виконується початкова ініціалізація компонентів і налаштувань.

Спочатку записується повідомлення в лог про початок ініціалізації. Викликається метод _update_app, обгорнутий у блок try-except, для обробки можливих помилок, які записуються в лог. Потім викликається ініціалізатор базового класу через super().__init__(parent), після чого створюються та налаштовуються основні компоненти інтерфейсу. Основне вікно пов’язане з об’єктом MainUI, який відповідає за структуру графічного інтерфейсу, і виконується метод setupUi для його налаштування.

```
37  class MainWindow(QMainWindow):
38      def __init__(self, parent=None):
39          logger.info('init MainWindow class in main_window.py')
40          try:
41              self._update_app()
42          except Exception as e:
43              logger.error(f'{e}')
44          super().__init__(parent)
45          self.alert_window = None
46          self.ui = MainUI()
47          self.ui.setupUi(self)
48          if settings.IS_CHANGE_DETECTORS:
49              self.ui.change_1.setHidden(True)
50              self.ui.change_2.setHidden(True)
51          self.show()
52          self.showFullScreen()
53          self.ui.diametr_button.setDisabled(True)
54          self.pressed_change_value_button = None
55          self.value_to_change = None
56          self.ui.len_button.setText(f'{get_max_meters()}')
57          self.ui.diametr_button.setText(f'{get_max_diametr()}')
58          self.ui.width_button.setText(f'{get_max_width()}')
59          self.ui.user_speed_s2_button.setText(f'{get_user_speed_s2()}')
60          self.ui.park_button.clicked.connect(lambda: threading.Thread(target=homing).start())
61          self.ui.add_button.clicked.connect(self.add_value)
62          self.ui.minus_button.clicked.connect(self.minus_value)
63          self.ui.len_button.clicked.connect(self.len_button_clicked)
64          self.ui.diametr_button.clicked.connect(self.diametr_button_clicked)
65          self.ui.user_speed_s2_button.clicked.connect(self.user_speed_s2_button_clicked)
66          self.ui.width_button.clicked.connect(self.width_button_clicked)
67          self.ui.button_10.clicked.connect(self.button_10_clicked)
68          self.ui.button_1.clicked.connect(self.button_1_clicked)
69          self.ui.button_01.clicked.connect(self.button_01_clicked)
70          self.ui.button_001.clicked.connect(self.button_001_clicked)
71          self.update_ui_timer = QTimer()
72          self.update_ui_timer.timeout.connect(self.update_ui)
73          self.update_ui_timer.start(500)
74          threading.Thread(target=homing).start()
75          threading.Thread(target=self.check_button_pressing).start()
76          logger.info('finish init MainWindow class in main_window.py')
```

Рис. 4.6. Ініціалізація main_window.py

Залежно від параметрів у файлі налаштувань (settings.IS_CHANGE_DETECTORS), деякі елементи інтерфейсу, такі як

кнопки change_1 і change_2, приховуються. Далі вікно відображається на екрані в повноекранному режимі. Деякі кнопки, наприклад, diametr_button, на початку роботи вимикаються (setDisabled(True)), щоб запобігти їхньому використанню до належної ініціалізації.

Початкові значення для полів "довжина", "діаметр", "ширина" та "швидкість" встановлюються відповідно до результатів викликів функцій get_max_meters, get_max_diametr, get_max_width та get_user_speed_s2. Далі кнопка "Паркування" (park_button) прив’язується до функції homing, яка запускається у новому потоці. Інші кнопки також підключаються до відповідних функцій, які змінюють параметри, запускають чи зупиняють двигуни, а також налаштовують додаткові параметри.

Таймер QTimer використовується для регулярного оновлення інтерфейсу користувача кожні 500 мс через виклик методу update_ui.

Після завершення ініціалізації у лог записується повідомлення про її успішне завершення. Таким чином, клас MainWindow забезпечує інтерактивність застосунку, роботу з апаратними компонентами та постійне оновлення інтерфейсу, що робить його основним компонентом програми.Далі представлено чотири методи, насправді їх більше, які обробляють натискання відповідних кнопок у графічному інтерфейсі програми. Кожен метод змінює вигляд кнопок, виділяючи натиснуту кнопку, і задає значення для зміни параметрів системи.

Методи button_10_clicked, button_1_clicked, button_01_clicked і button_001_clicked відповідають за обробку натискання кнопок з різними кроками зміни значень: 10, 1, 0.1 та 0.01 відповідно. Коли викликається кожен з методів, в лог записується повідомлення про натиснення відповідної кнопки, наприклад, "pressed button '10'". Після цього змінюється стиль натиснутої кнопки через метод setStyleSheet, де вказується, що її фон має бути яскраво-блакитним (#00ffc8), а інші кнопки отримують фон із темно-зеленим відтінком (#38a962). Такий підхід дозволяє візуально відобразити активну кнопку для зручності користувача.

```
1 usage
def button_10_clicked(self):
    logger.info('pressed button "10"')
    self.ui.button_10.setStyleSheet("background-color : #00ffc8;font: 26pt Furore; color: #fff; border: none; border-radius: 15px;")
    self.ui.button_1.setStyleSheet("background-color : #38A962;font: 26pt Furore; color: #fff; border: none; border-radius: 15px;")
    self.ui.button_01.setStyleSheet("background-color : #38A962;font: 26pt Furore; color: #fff; border: none; border-radius: 15px;")
    self.ui.button_001.setStyleSheet("background-color : #38A962;font: 26pt Furore; color: #fff; border: none; border-radius: 15px;")
    self.value_to_change = 10

1 usage
def button_1_clicked(self):
    logger.info('pressed button "1"')
    self.ui.button_1.setStyleSheet("background-color : #00ffc8;font: 26pt Furore; color: #fff; border: none; border-radius: 15px;")
    self.ui.button_10.setStyleSheet("background-color : #38A962;font: 26pt Furore; color: #fff; border: none; border-radius: 15px;")
    self.ui.button_01.setStyleSheet("background-color : #38A962;font: 26pt Furore; color: #fff; border: none; border-radius: 15px;")
    self.ui.button_001.setStyleSheet("background-color : #38A962;font: 26pt Furore; color: #fff; border: none; border-radius: 15px;")
    self.value_to_change = 1

1 usage
def button_01_clicked(self):
    logger.info('pressed button "0.1"')
    self.ui.button_01.setStyleSheet("background-color : #00ffc8;font: 26pt Furore; color: #fff; border: none; border-radius: 15px;")
    self.ui.button_10.setStyleSheet("background-color : #38A962;font: 26pt Furore; color: #fff; border: none; border-radius: 15px;")
    self.ui.button_1.setStyleSheet("background-color : #38A962;font: 26pt Furore; color: #fff; border: none; border-radius: 15px;")
    self.ui.button_001.setStyleSheet("background-color : #38A962;font: 26pt Furore; color: #fff; border: none; border-radius: 15px;")
    self.value_to_change = 0.1

1 usage
def button_001_clicked(self):
    logger.info('pressed button "0.01"')
    self.ui.button_001.setStyleSheet("background-color : #00ffc8;font: 26pt Furore; color: #fff; border: none; border-radius: 15px;")
    self.ui.button_01.setStyleSheet("background-color : #38A962;font: 26pt Furore; color: #fff; border: none; border-radius: 15px;")
    self.ui.button_1.setStyleSheet("background-color : #38A962;font: 26pt Furore; color: #fff; border: none; border-radius: 15px;")
    self.ui.button_10.setStyleSheet("background-color : #38A962;font: 26pt Furore; color: #fff; border: none; border-radius: 15px;")
    self.value_to_change = 0.01
```

Рис. 4.7. Реалізація деяких кнопок

У завершенні кожного методу змінна self.value_to_change отримує відповідне значення (10, 1, 0.1 або 0.01), яке використовується для подальшого налаштування параметрів системи, таких як діаметр, довжина або швидкість. Ця реалізація забезпечує простий і зрозумілий механізм інтерактивного керування параметрами системи через графічний інтерфейс.

Також пропонується до перегляду функціонал логування кожного натискання кнопок.

Логування – це запис будь-яких подій, які відбуваються під час роботи програми.

Коли викликаються відповідні функції кнопок, після натискання чи інших подій, відбувається логування. На рис. 4.1.8 показано логування таких параметрів як зміна діаметру філаменту, ширини котушки та швидкості введеної користувачем. Дана інформація відповідає групі “info”, адже містить лише відлагоджувальну інформацію. Також додатково викликаються функції оновлення відповідних полів на екрані.

```
def add_diametr(self, diametr):
    logger.info(f'operator add {diametr} diametr')
    add_diametr(diametr)
    self.ui.diametr_button.setText(f'{get_max_diametr()}')

2 usages
def add_width(self, width):
    logger.info(f'operstor add {width} width')
    add_width(width)
    self.ui.width_button.setText(f'{get_max_width()}')

2 usages
def add_user_speed_s2(self, s):
    logger.info(f'operator add {s} user_speed_s2')
    add_user_speed_s2(s)
    self.ui.user_speed_s2_button.setText(f'{get_user_speed_s2()}')
```

Рис. 4.8. Логування змін и параметрів

Усі основні функції та логіка виконання намотування знаходяться у файлі “bussines.py”. Саме реалізація деяких функцій всередині цих файлів являє собою комерційну таємницю, але є такі які не можуть зашкодити поточному користувачу. Саме про такі реалізації логіки далі буде іти мова.

```
def find_arduino(success_answer, success_response, baudrate=9600):
    ports = [f'/dev/ttyACM{i}' for i in range(10)] + [f'/dev/ttyUSB{i}' for i in range(10)]
    for port in ports:
        try:
            ser = serial.Serial(port, baudrate, timeout=1)
            start_time = time.time()
            for _ in range(10):
                # print(ser.readline().decode('utf-8'))
                if ser.in_waiting > 0:
                    response = ser.readline().decode('utf-8').strip()
                    if response == success_response:
                        logger.info(f"The device {success_response} found on port {port}")
                        ser.write(f'{success_answer}\r\n'.encode('utf-8'))
                        # while True:
                        #     print(ser.readline().decode('utf-8'))
                        return ser
                time.sleep(0.2)

        except (serial.SerialException, OSError):
            continue

    raise ArduinoConnectionError("Could not find arduino on USB ports")
```

Рис. 4.9. Функція пошуку Atmega2560 на портах USB

На зображенні представлено функцію find_arduino, яка виконує пошук підключеного до системи пристрою Arduino через USB-порти. Функція приймає три параметри: success_answer (відповідь, яку потрібно надіслати Arduino у разі успішного підключення), success_response (очікувана відповідь

від Arduino) ці два важливі аргументи знаходяться у зашифрованому форматі. Також baudrate (швидкість передачі даних, за замовчуванням встановлена на 9600).

Спочатку створюється список портів, які потрібно перевірити, включаючи /dev/ttyACM{i} та /dev/ttyUSB{i} для десяти можливих номерів портів кожного типу. Для кожного порту з цього списку функція намагається встановити з'єднання, використовуючи pyserial.Serial. Якщо з'єднання не вдається через виняток serial.SerialException або OSError, цикл переходить до наступного порту.

Якщо з'єднання встановлено, функція починає читати дані з порту протягом 10 ітерацій або поки у порту є вхідні дані (ser.in_waiting > 0). Отримана відповідь з порту декодується з формату UTF-8 та обрізається від зайвих символів (strip()). Якщо відповідь збігається з очікуваною (success_response), функція записує у лог повідомлення про успішне виявлення пристрою та надсилає повідомлення success_answer до Arduino. Також обов’язково зберігаються логи про успішне підключення. Після цього функція повертає об'єкт Serial, що представляє підключення.

У разі, якщо жоден порт не відповідає очікуваним параметрам, функція викидає виняток ArduinoConnectionError із повідомленням "Could not find arduino on USB ports". Таким чином, функція забезпечує автоматичний пошук Arduino на доступних USB-портах та встановлює з'єднання, якщо пристрій успішно ідентифіковано.

Далі алгоритм отримання інформації з енкодерів та розміщення її у відповідні глобальні змінні. Функція “get_serial_value()” вкладена в іншу значущу функцію, для легкості розуміння коду. Вона викликається одразу після відправлення спеціальної ключ-команди(запит на отримання даних) до мікроконтролера Atmega2560, і далі викликається ця функція для отримання довжини намотаного філаменту та кута компенсуючої ланки. У коді використовуються глобальні змінні: finished_meters, encoder_value, is_run, serial та prev_count, які забезпечують взаємодію між функцією і основними параметрами програми.

Спочатку функція зчитує дані з серійного порту за допомогою методу serial.readline(), декодує отриманий рядок у форматі UTF-8 і видаляє зайві символи (\r, \n, пробіли) для очищення даних. Очищений рядок розбивається на підрядки за роздільником “|”, і кожен підрядок обробляється окремо.

Для кожного підрядка виконується спроба розділення за роздільником “:”. Якщо перший елемент після розділення (values[0]) відповідає символу m і система перебуває в активному стані (is_run), то другий елемент (values[1]) перетворюється у число з плаваючою комою і записується у змінну finished_meters, яка, ймовірно, представляє довжину намотаного філаменту. Якщо ж перший елемент дорівнює “e”, функція викликає метод change_speed, передаючи йому значення другого елемента, перетворене у ціле число, що, ймовірно, регулює швидкість.

У разі виникнення помилки обробки будь-якого з підрядків, виняток обробляється, а інформація про помилку записується в лог.

```
def get_serial_value():
    global finished_meters, encoder_value, is_run, serial, prev_count
    decoded_string = serial.readline().decode('utf-8')
    cleaned_string = decoded_string.replace('\\r', '').replace('\\n', '').strip()
    # if IS_RUN:
        # logger.debug(f'arduino values: {cleaned_string}')
    for i in cleaned_string.split('|'):
        if i != '':
            try:
                values = i.split(':')
                if values[0] == 'm' and is_run:
                    finished_meters = float(values[1])
                if values[0] == 'e' and is_run:
                    change_speed(int(values[1]))
            except Exception as e:
                logger.error(f'get_serial_value {e}')
                continue
    threading.Timer( interval: 0.095, get_serial_value).start()
```

Рис. 4.10. Функція отримання довжини та кута з порту USB

Після завершення обробки функція ініціалізує таймер threading.Timer, який викликає get_serial_value через кожні 0.095 секунди, забезпечуючи регулярне оновлення даних. Таким чином, функція виконує безперервний цикл

зчитування, обробки та оновлення параметрів системи на основі даних, отриманих із серійного порту.

Далі продемонстровано реалізацію функції homing яка виконує процедуру повернення системи в початковий стан ("дом") для забезпечення коректної роботи механізму. Вона використовує глобальні змінні для взаємодії з іншими частинами програми, включаючи стан системи, параметри двигунів і змінні, що відображають прогрес роботи.

```
def homing(_=None, is_force=False):
    logger.info('run home function')
    global is_homing, is_first_run, speed_s1, finished_layers_count, \
        latest_bip_meters_count, finished_width, finished_meters, serial, last_s1_position
    if is_homing and not is_force:
        return
    logger.info('homing')
    stop_motors()
    clear_encoders()
    finished_layers_count = 0
    finished_meters = 0.0
    is_homing = True
    is_first_run = True
    latest_bip_meters_count = 50
    finished_width = 0
    clear_encoders()
    drivers_arduino.change_s1_speed(100)

    if is_change2():
        logger.info('homing change2')
        drivers_arduino.run_s1_engine(motor1_steps)

    if is_change1():
        logger.info('homing change1')
        drivers_arduino.run_s1_engine(-1 * motor1_steps)

    while not is_change2():
        drivers_arduino.run_s1_engine(-50)
    drivers_arduino.change_s1_speed(speed_s1)
    time.sleep(0.5)
    last_s1_position = 'right'
    is_homing = False
    logger.info('finish homing')
```

Рис. 4.11. Функція паркування в домашню позицію

Функція починається з запису у лог про запуск процедури. Якщо система вже перебуває у стані "homing" і не активовано примусове виконання (is_force=False), функція завершується, щоб уникнути повторного виконання.

Далі функція викликає зупинку двигунів (stop_motors) і очищення енкодерів (clear_encoders). Значення змінних, які відповідають за підрахунок намотаного матеріалу, ширини та завершених шарів, скидаються до початкових значень. Прапорець is_homing встановлюється у True, щоб позначити, що процедура "homing" активна. Для встановлення початкової швидкості викликається метод change_s1_speed(100).

Функція перевіряє стани кінцевих датчиків через умови is_change2() і is_change1(). Якщо спрацьовує is_change2, двигун запускається у прямому напрямку (run_s1_engine(motor1_steps)), а якщо is_change1 — у зворотному напрямку (run_s1_engine(-1 * motor1_steps)).

Після цього запускається цикл while, який працює доти, доки датчик is_change2 не сигналізує про досягнення кінцевого положення. У межах циклу двигун постійно рухається у прямому напрямку з невеликою швидкістю (50RPM), після чого виконується пауза у 0.5 секунди. Це забезпечує плавний рух до початкового положення.

Коли кінцеве положення досягнуто, змінна is_homing встановлюється у False, а остання позиція (last_s1_position) фіксується як "right". У лог записується повідомлення про завершення процедури.

Таким чином, функція homing відповідає за повернення механізму до нульової позиції, виконуючи необхідні перевірки та корекції положення, що забезпечує точність і стабільність роботи системи.

Далі функція “run_motors_loop()” відповідає за запуск двигунів і контроль їх роботи в системі. Вона використовує глобальні змінні для взаємодії з іншими частинами програми, зокрема змінні, що відобража ють стан системи, швидкість двигунів і попередні значення енкодерів.

На початку функція перевіряє значення, яке повертає функція read_br(). Якщо значення більше 1000, функція викликає quit(), що, ймовірно, завершує роботу програми для уникнення некоректного запуску. Далі перевіряється, чи знаходиться система в режимі "is_homing" або чи двигуни вже запущені (is_run). Якщо хоча б одна з цих умов виконується, функція завершується, щоб уникнути повторного запуску двигунів.

Якщо це перший запуск (is_first_run), у лог записується відповідне повідомлення, а прапорець is_first_run змінюється на False. Викликається функція clear_encoders() для очищення даних з енкодерів. Значення попередньої швидкості двигуна prev_s2_speed ініціалізується поточним значенням користувацької швидкості user_speed_s2, а змінні SPEED_S2 і SPEED_S1 приймають відповідні швидкості.

Далі перевіряється, чи значення SPEED_S2 більше нуля. Якщо це так, двигуну другого рівня задається швидкість через drivers_arduino.change_s2_speed(SPEED_S2). Якщо значення SPEED_S2 негативне, в лог записується повідомлення про помилку. Аналогічно, якщо SPEED_S1 більше нуля, двигуну першого рівня задається швидкість через drivers_arduino.change_s1_speed(SPEED_S1), інакше в лог також записується помилка.

```
def run_motors_loop(_=None):
    global is_run, is_first_run, is_homing, start_meter_encoder_value, prev_s2_speed, user_speed_s2, prev_count
    if read_br() ⩾ 1000:
        quit()
    if is_run or is_homing:
        return
    if is_first_run:
        logger.info('IS_FIRST_RUN = True')
        clear_encoders()
        is_first_run = False
        prev_s2_speed = user_speed_s2
        SPEED_S2 = prev_s2_speed
        SPEED_S1 = SPEED_S2
        prev_count = -100
        if SPEED_S2 > 0:
            drivers_arduino.change_s2_speed(SPEED_S2)
        else:
            logger.error(f'SPEED_S2 is negative: {SPEED_S2}')
        if SPEED_S1 > 0:
            drivers_arduino.change_s1_speed(SPEED_S1)
        else:
            logger.error(f'SPEED_S1 is {SPEED_S1}')
        time.sleep(0.5)

    is_run = True
    threading.Thread(target=start_namotka).start()
```

Рис. 4.12. Алгоритм перевірки моторів та їх запуск

Після завершення ініціалізації швидкостей функція виконує затримку в 0.5 секунди для стабілізації. Потім змінна is_run встановлюється у True, що позначає активний стан двигунів, і створюється новий потік, який запускає функцію “start_namotka()”. Ця функція відповідає за подальшу роботу механізму намотування.

Таким чином, run_motors_loop виконує ініціалізацію двигунів, встановлює їх швидкості й запускає процес намотування, забезпечуючи перевірку і обробку можливих помилок на етапі запуску.

4.2.Розробка алгоритмів керування та регулювання крокових

Мікроконтролер ATmega2560 Робоча напруга 5В Вхідна напруга (рекомендована) 7-12В Вхідна напруга (гранична) 6-20 В Цифрові Входи/Виходи 54 (14 з яких можуть працювати також як виходи ШІМ) Аналогові входи 16 Постійний струм через вхід/вихід 40 mA Постійний струм для виведення 3.3 50 mA Флеш-пам'ять 256 KB (з яких 8 KB використовуються для завантажувача) ОЗУ 8 KB Енергонезалежна пам'ять 4 KB Тактова частота 16 Мгц.



Pin Number	Pin Name	Description
10, 31, 61, 80	V_{CC}	IC Supply pins
11, 32, 62, 81, 99	GND	IC ground reference pins
98	A_{REF}	Reference supply for ADC
100	A_{VCC}	Supply pin for analog peripherals
33, 34	XTAL	Crystal oscillator pins
30	$\overline{RESET}$	Reset pin, active low
2 - 9	PE0 – PE7	GPIO Port E pins
12 – 18, 27	PH0 – PH6, PH7	GPIO Port H pins
19 - 26	PB0 – PB7	GPIO Port B pins
28 – 29, 51 – 52, 70,1	PG3 – PG4, PG0 – PG1, PG2,PG5	GPIO Port G pins
35 - 42	PL0 – PL7	GPIO Port L pins
43 - 50	PD0 – PD7	GPIO Port D pins
53 - 60	PC0 – PC7	GPIO Port C pins
63 – 69, 79	PJ0 – PJ6, PJ7	GPIO Port J pins
71 - 78	PA7 – PA0	GPIO Port A pins
82 - 89	PK7 – PK0	GPIO Port K pins
90 - 97	PF7 – PF0	GPIO Port F pins

Рис. 4.13 Pinout-специфікація Atmega2560

Реалізація керування кроковими двигунами на Atmega2560 потребує правильного розподілення ресурсу мікроконтролера. Для цього було використано деякі бібліотеки які задіюють апаратні таймери та переривання автоматично, що спрощує написання коду та зменшує час розробки.

Формально код був розділений на дві частини, частина для керування кроковими двигунами та частина обробки сигналу енкодерів. Фактично це все елементи одної програми для мікроконтролера.

Для обробки сигналів енкодерів використано бібліотеку RotaryEncoder. Ця бібліотека має ліцензію “Software License Agreement (BSD License)”. Це означає, що можна вільно використовувати програмне забезпечення, змінювати його та інтегрувати у свої проєкти. Однак, потрібно зберігати інформацію про авторські права та дотримуватися умов ліцензії. При використанні такого софту себе весь ризик використання програмного забезпечення лежить на розробнику що його застосовує, а автори не несуть відповідальності за його можливі недоліки чи наслідки його використання. Бібліотека взята з ресурсу GitHub за посиланням(дійсне станом на 13.12.2024): https://github.com/mathertel/RotaryEncoder .Також про це говорить інформація на початку заголовочного файлу бібліотеки.

```
1   // -----
2   // RotaryEncoder.h - Library for using rotary encoders.
3   // This class is implemented for use with the Arduino environment.
4   //
5   // Copyright (c) by Matthias Hertel, http://www.mathertel.de
6   //
7   // This work is licensed under a BSD 3-Clause style license,
8   // https://www.mathertel.de/License.aspx.
9   //
10  // More information on: http://www.mathertel.de/Arduino
11  // -----
12  // 18.01.2014 created by Matthias Hertel
13  // 16.06.2019 pin initialization using INPUT_PULLUP
14  // 10.11.2020 Added the ability to obtain the encoder RPM
15  // 29.01.2021 Options for using rotary encoders with 2 state changes per latch.
16  // -----
```

Рис. 4.14. Ліцензійна угода бібліотеки RotaryEncoder

Далі реалізація коду та підключення енкодерів. У представленому коді, написаному для Arduino, визначено налаштування для роботи з енкодерами і функція для перевірки їхньої позиції.

На початку коду підключаються бібліотеки <Arduino.h> та "RotaryEncoder.h", які забезпечують доступ до основних функцій Arduino та дозволяють працювати з енкодерами.

Далі оголошуються та ініціалізуються кілька змінних:

mm_distance — типу float, задає довжину у міліметрах, наприклад, 5000 мм.

count_clicks — типу int, задає кількість імпульсів енкодера, наприклад, 25400.

mm_for_one_click — типу double, обчислює відстань у міліметрах, що відповідає одному імпульсу, шляхом ділення загальної довжини (mm_distance) на кількість імпульсів (count_clicks) і масштабування до кілометрів.

globalMillis — типу unsigned long, використовується, ймовірно, для збереження часу або частоти оновлення (змінна поки що не ініціалізована).

```
#include <Arduino.h>
#include "RotaryEncoder.h"

float mm_distance = 5000;
int count_clicks = 25400;
double mm_for_one_click = mm_distance / count_clicks / 1000;
unsigned long heartbeatMillis;

#define PIN_IN1 19
#define PIN_IN2 18
#define PIN_IN3 22
#define PIN_IN4 21

RotaryEncoder *encoder = nullptr;
RotaryEncoder *encoder_2 = nullptr;

void checkPosition()
{
  encoder->tick();
  encoder_2->tick();
}
```

Рис. 4.15. Фрагмент ініціалізації енкодерів

Визначаються макроси для пінів, які будуть використовуватися для підключення енкодерів:

PIN_IN1, PIN_IN2, PIN_IN3, PIN_IN4 встановлюються на пін-коди 19, 18, 22 і 21 відповідно.

Оголошуються два вказівники RotaryEncoder:

encoder та encoder_2, які ініціалізуються як nullptr, тобто поки що не пов’язані з конкретними енкодерами.

Описано функцію checkPosition(), яка викликає метод tick() для кожного енкодера:

Метод tick() оновлює стан енкодера, наприклад, зчитує його поточну позицію чи обробляє зміни вхідного сигналу.

Наступний фрагмент коду

```
encoder = new RotaryEncoder(PIN_IN1, PIN_IN2, RotaryEncoder::LatchMode::TWO03);
encoder_2 = new RotaryEncoder(PIN_IN3, PIN_IN4, RotaryEncoder::LatchMode::TWO03);

// register interrupt routine
attachInterrupt(digitalPinToInterrupt(PIN_IN1), checkPosition, RISING);
attachInterrupt(digitalPinToInterrupt(PIN_IN2), checkPosition, RISING);
attachInterrupt(digitalPinToInterrupt(PIN_IN3), checkPosition, RISING);
attachInterrupt(digitalPinToInterrupt(PIN_IN4), checkPosition, RISING);
```

Рис. 4.16. Завершення ініціалізації та оголошення енкодерів

У цьому фрагменті коду налаштовуються два енкодери та реєструються обробники переривань для роботи з ними.

Спочатку створюються об'єкти енкодерів за допомогою оператора new. Для кожного енкодера вказуються відповідні пін-коди, підключені до сигналів енкодера, а також режим фіксації сигналу (LatchMode).

Перший енкодер (encoder) використовує пін-коди PIN_IN1 і PIN_IN2, а його режим фіксації задано як TW003.

Другий енкодер (encoder_2) підключений до пінів PIN_IN3 і PIN_IN4, і також працює в режимі TW003.

Після створення об'єктів енкодерів виконується реєстрація обробників переривань. Це здійснюється за допомогою функції attachInterrupt, яка дозволяє викликати певну функцію, коли на вході відповідного піну відбувається переривання.

Для кожного піну (PIN_IN1, PIN_IN2, PIN_IN3, PIN_IN4) встановлюється переривання на фронті сигналу (RISING), тобто обробник викликатиметься, коли сигнал зміниться з низького рівня на високий. Вказується функція-обробник checkPosition, яка буде викликана під час переривання. Ця функція оновлює стан енкодерів, використовуючи методи, що враховують зміну сигналів.

Таким чином, цей код забезпечує автоматичну обробку змін сигналів від енкодерів через переривання, що дозволяє точно і швидко відстежувати їхній стан без постійного опитування входів у головному циклі програми. Це робить

роботу з енкодерами ефективною та оптимізує використання ресурсів мікроконтролера.

Наступний код виконує цикл ініціалізації контролера, через серійний порт. Основна мета — очікування на отримання певного повідомлення та відправки (рядка "ok") для завершення ініціалізації.

```
bool _setup = true;
while(_setup){
  Serial.println(key);
  if (Serial.available()) {
      String inputString = Serial.readStringUntil('\n');
      if(inputString == inputKey);
      Serial.println("ok");
      _setup = false;
      return;
  }
  delay(500);

}
```

Рис. 4.17. Пошук головного пристрою на портах USB

Спочатку оголошується булева змінна _setup зі значенням true, яка слугує умовою для циклу while. Цикл працює, поки _setup дорівнює true. Всередині циклу виконується передача значення змінної key через серійний порт за допомогою функції Serial.println(key). Таким чином, програма надсилає певний запит або повідомлення для взаємодії із зовнішнім пристроєм.

Далі перевіряється наявність вхідних даних у серійному порту за допомогою Serial.available(). Якщо дані є, вони зчитуються як рядок до символу нового рядка (\n) за допомогою Serial.readStringUntil('\n') і зберігаються у змінну inputString.

Після цього перевіряється, чи зчитаний рядок (inputString) дорівнює "inputKey". Якщо умова виконується, в порт відправляється повідомлення про успішне підключення (Serial.println("ok")) та змінна _setup змінюється значення на false, що завершує цикл. Одразу після цього викликається команда return, яка припиняє виконання функції незалежно від умови циклу.

Якщо дані в серійному порту відсутні або отриманий рядок не відповідає "ok", цикл повторюється після затримки у 500 мілісекунд, заданої функцією delay(500)

Далі буде продемонстровано реалізацію керування кроковими двигунами.

```
#define home_speed 400
#define steps 1000
int _accel = 0;
long stepper1_speed = 100, stepper2_speed = 500;
int xValue = -1;  // Змінна для зберігання значення x
unsigned long aValue = 0;  // Змінна для зберігання значення a
bool flag_m1 = false;
bool flag_m2 = false;
int m1_target = steps/10*1.75;
int error = 0;

#include "TimerOne.h"
#include "Stepper.h"

Stepper<STEPPER2WIRE> stepper1(steps, 5, 6, 7);    // Каретка
Stepper<STEPPER2WIRE> stepper2(steps, 3, 2, 22); // Бабіна
```

Рис. 4.18. Ініціалізація потрібних змінних та об’єктів крокових двигунів

Представлений на рис.4.2.5 код описує налаштування параметрів і підключення двох крокових двигунів для роботи з системою керування на базі Atmega2560. Код включає визначення констант, змінних і об’єктів, необхідних для керування двигунами.

На початку оголошуються константи:

home_speed, яка задає швидкість руху двигунів у процесі повернення в початкову позицію, встановлена на 400.

steps, що задає кількість кроків двигуна, встановлена на 1000.

Далі оголошуються змінні:

_accel — початкове прискорення двигунів, встановлене на 0.

stepper1_speed і stepper2_speed — швидкості двох двигунів, встановлені відповідно на 100 і 500.

xValue і aValue — змінні для збереження певних параметрів.

flag_m1 і flag_m2 — булеві змінні, що відстежують стан першого і другого двигунів, спочатку встановлені на false.

m1_target — цільова кількість кроків для першого двигуна, обчислена як steps / 10 * 1.75.

error — змінна для відстеження помилок, початково дорівнює 0.

Код підключає дві бібліотеки. Бібліотека TimerOne підключена для можливості реалізації апаратного таймера, адже код був написаний у редакторі VS code, і він не містить всіх потрібних для роботи Atmega2560 бібліотек.

"Stepper.h" — для керування кроковими двигунами. Бібліотека має ліцензію MIT License — це одна з найпоширеніших і найпростіших ліцензій у світі відкритого програмного забезпечення. Вона забезпечує розробникам максимальну свободу у використанні, зміні та розповсюдженні програмного забезпечення, зберігаючи лише кілька базових умов.

У розділі підключення крокових двигунів створюються два об'єкти:

stepper1 відповідає за керування кареткою, підключений до пінів 5, 6 та 7.

stepper2 відповідає за керування бабіною (котушкою), підключений до пінів 3, 2 та 22.

Таким чином, цей код забезпечує базову конфігурацію для керування двома кроковими двигунами у системі, задаючи параметри їх роботи, цілі руху та пін-коди підключення.

```
stepper1.setRunMode(FOLLOW_POS);
stepper2.setRunMode(FOLLOW_POS);
stepper1.setAcceleration(_accel);
stepper2.setAcceleration(_accel);
stepper1.setMaxSpeed(stepper1_speed);
stepper2.setMaxSpeed(stepper2_speed);
```

Рис. 4.19. Початкові налаштування крокових двигунів.

Двигуни налаштовані на режим режим плавного руху до заданої позиції з прискоренням та обмеженням швидкості(рис 4.2.6) що і продемонстровано в функції (Setup()) програми мікроконтролера.

Далі фрагмент коду для задання швидкості двигуна при певних умовах. Код представляє дві умови, які керують діями крокового двигуна залежно від значення змінної number. У першій умові (if (number < 40000)), якщо значення number менше 40000, з нього віднімається 30000, скидається значення помилки (error = 0), встановлюється цільова позиція двигуна за допомогою setTarget у відносному режимі, і задається максимальна швидкість через setMaxSpeed. Прапорець flag_m1 встановлюється у true, а функція завершується через return.

```
if (number < 40000 ){
  number -= 30000;
  //m1_target = number;
  error = 0;
  stepper1.setTarget(number,RELATIVE);
  stepper1.setMaxSpeed(stepper1_speed);
 //Serial.println(number);
  flag_m1 = true;
  return;
}

if (number < 50000 ){
  number -= 40000;
  //m1_target = (-1) * number;
  error = 0;
  stepper1.setTarget((-1) * number,RELATIVE);
  stepper1.setMaxSpeed(stepper1_speed);
  //Serial.println(number);
  flag_m1 = true;
  return;
}
```

Рис. 4.20. Код задання швидкості двигуну при певних умовах

У другій умові (if (number < 50000)), якщо number менше 50000, з нього віднімається 40000, обчислюється цільова позиція двигуна з протилежним знаком і виконується аналогічна конфігурація параметрів двигуна. Ця логіка використовується для керування рухом двигуна залежно від поточного значення number.

У розділі, присвяченому розробці програмного забезпечення для Raspberry Pi та Atmega2560, було виконано інтеграцію апаратних компонентів

із програмною логікою для забезпечення точного й ефективного управління намотувальним пристроєм. Програмне забезпечення розроблено з урахуванням двох основних частин системи: графічного інтерфейсу оператора, що працює на Raspberry Pi, і низькорівневого управління двигунами та обробки сигналів датчиків, реалізованого на мікроконтролері Atmega2560.

Для Raspberry Pi створено графічний інтерфейс оператора на основі бібліотеки PySide6, що забезпечує інтуїтивно зрозуміле налаштування параметрів роботи системи. Інтерфейс дозволяє оператору контролювати довжину намотаного філаменту, задавати параметри швидкості, діаметра й ширини філаменту, а також запускати чи зупиняти процес намотування за допомогою відповідних кнопок. Для забезпечення стабільності роботи розроблено функції для обробки натискань кнопок, відображення активного стану та передачі команд через серійний порт до Atmega2560.

На Atmega2560 реалізовано низькорівневу логіку управління, яка включає роботу з кроковими двигунами, обробку сигналів із датчиків та енкодерів і забезпечення точності руху механізмів. Використовуються бібліотеки, які дозволяють керувати двигунами через встановлення цільових позицій, швидкостей і прискорень. Окрема увага приділена процедурі повернення каретки в початкову позицію (homing), що забезпечує точність початкових налаштувань системи.

Система передачі даних між Raspberry Pi і Atmega2560 побудована на основі серійного протоколу, який дозволяє обмінюватися командами й отримувати відгуки в режимі реального часу. Логіка обробки серійних даних на мікроконтролері включає перевірку отриманих сигналів, зчитування параметрів, таких як довжина намотаного філаменту або стан кінцевих датчиків, та регулювання швидкості й положення механізмів у відповідь на команди.

Розроблене програмне забезпечення відзначається модульною структурою, що забезпечує його гнучкість і масштабованість. Завдяки використанню окремих бібліотек і підходів до багатопоточності в Raspberry Pi, а також асинхронної обробки сигналів на Atmega2560, система здатна

працювати ефективно та безперервно. Це рішення забезпечує стабільну роботу системи, точність у виконанні завдань і зручність для оператора. Усі компоненти програмного забезпечення були ретельно протестовані, щоб забезпечити коректну інтеграцію між апаратними й програмними модулями.

5. ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ТА ТЕСТУВАННЯ

5.1. Оцінка та інтеграція всіх підсистем

На цьому етапі проводиться перевірка коректної роботи всіх підсистем: механічних, електронних і програмних. Оцінюється правильність взаємодії між Raspberry Pi, Atmega2560, кроковими та серводвигунами, енкодерами й датчиками. Здійснюється інтеграція графічного інтерфейсу оператора з низькорівневим управлінням на мікроконтролері, тестується обмін даними через серійний порт. Перевіряється стабільність роботи механізмів, коректність реагування на команди та зворотний зв'язок від датчиків. Основним завданням цього підрозділу є забезпечення синхронної та безпомилкової роботи всіх компонентів системи.

Врезультаті тестувань було вивлно що швидкість передачі між Raspberry та Atmega2560 115200бод в умовах які були поставлені технічним завданням було недостатньо. Тому було прийнято рішення підняти швидкість бо значення 500000бод. При швидкості 500000 біт/с комунікація відбувається значно швидше, що забезпечує миттєву реакцію системи на зміни параметрів, таких як швидкість намотування чи положення каретки. Це особливо важливо для сценаріїв з високими вимогами до динамічності, коли дані повинні передаватися та оброблятися практично миттєво. Проте, при такій високій швидкості можуть виникати ризики втрати даних або помилок через електромагнітні завади, якщо кабелі чи апаратне середовище не оптимально налаштовані.

Порівняння:

115200 біт/с: Забезпечує стабільність і надійність для більшості задач із середніми вимогами до швидкості. Ідеально підходить для стандартних задач керування.

500000 біт/с: Забезпечує максимальну швидкість реакції та передачі даних, але може бути менш стабільною в умовах завад. У випадку та обставинах які обумовлені умовами де знаходиться система, такої швидкості достатньо. Та вона забезпечує стабільне з'єднання протягом усього робочого процесу.

Також експерементальним шляхом було визначено, що швидкість та прискорення каретки занадто висока. Було прийнято рішення знизити ці показники вдвоє.

5.2. Тестування працездатності пристрою

Головним етапом - перевірити ефективність роботи системи автоматизованого намотування філаменту при швидкості намотування 50 м/хв і визначити оптимальні параметри швидкості роботи компенсуючого пристрою та намотувального механізму.

В процесі роботи було виявлено хибні спрацювання на пінах мікроконтролера. У таблиці наведено дані про хибні спрацювання системи до та після встановлення фільтрів для зменшення електромагнітних перешкод від драйвера серводвигуна:

Тип помилок Кількість помилок до встановлення фільтрів Кількість помилок після встановлення фільтрів

Табл. 5.1

Тип помилок	Кількість помилок до встановлення фільтрів	Кількість помилок після встановлення фільтрів
Збої у передачі даних між Raspberry Pi та Atmega2560	2	0
Хибні спрацювання кнопок	56	0
Переривання роботи двигунів	3	0

До встановлення фільтрів драйвер серводвигуна спричиняв значні електромагнітні перешкоди, що призводило до численних збоїв у роботі системи, зокрема у передачі даних, хибних спрацюваннях на пінах мікроконтролерів та переривання роботи двигунів. Встановлення фільтрів

значно зменшило кількість помилок, покращивши стабільність і надійність системи.

Також продемонструю характер завада на осцилографі до встановлення фільтру. Чітко видно що при переходах сигналу з 0 в 1 відбувається аномальний шум на пінах. Все це заважало нормальній роботі системи



Рис. 5.1. Покази осцилографа до встановлення фільтру

За час тестування(більше 12год) в межах виробництва було проведено дослід з виявленя причин. Однією з причин були шумові викиди в мережу 220V потужним частотним перетворювачем та драйвером серводвигуна. Рішенням було встановленням фільтру на блок живлення для мікроконтролерів.

Система була протестована при швидкості намотування 50 м/хв, що відповідало стандартним умовам роботи пристрою. Під час тестування моніторилися робота компенсуючого механізму, відповідь системи на зміну швидкості філаменту, а також коректність укладки матеріалу на котушку.

В процесі спостережень було виявлено, що компенсуючий пристрій завжди намагався підтримувати свою швидкість на рівні близько 130% від заданої базової швидкості намотувального пристрою. Це свідчить про те, що компенсуючий механізм автоматично адаптується до змін в системі, намагаючись забезпечити оптимальний натяг філаменту та уникнути провисання.

На основі отриманих даних було вирішено збільшити базову швидкість роботи намотувального пристрою до 80 м/хв, щоб компенсуючий механізм працював у більш ефективному режимі і не перевантажував систему, працюючи на надлишкових швидкостях.

Результатами при збільшенні швидкості до 80 м/хв компенсуючий пристрій демонстрував стабільну роботу з меншою амплітудою регулювання швидкості, забезпечуючи натяг філаменту без перевантаження.

Зміна базової швидкості покращила узгодженість роботи всієї системи та підвищила її продуктвність.

Тестування підтвердило, що підвищення базової швидкості намотувального пристрою до 80 м/хв дозволяє зменшити надмірні навантаження на компенсуючий пристрій і забезпечити більш ефективну та стабільну роботу всієї системи.

Далі фото з реального тестування на виробництві

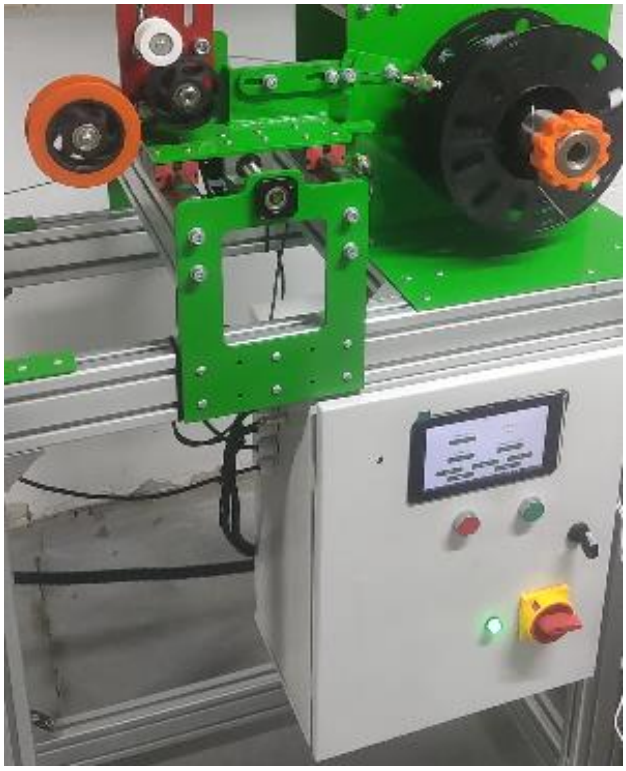


Рис. 5.2. Фото протестованої установки в межах виробництва.

ВИСНОВОК

Виконана дипломна робота демонструє комплексний та системний підхід до створення автоматизованої системи управління намотуванням філаменту, яка забезпечує високу точність, стабільність і продуктивність роботи. Інтеграція Raspberry Pi та ATmega2560 дозволила реалізувати ефективне керування механізмами системи, поєднуючи високорівневе управління з низькорівневою обробкою сигналів. Завдяки розробленому графічному інтерфейсу оператор отримує зручний інструмент для налаштування та контролю роботи пристрою в реальному часі.

Проведені експериментальні дослідження показали, що вибір оптимальної швидкості намотування (80 м/хв) і впровадження системи фільтрації електромагнітних перешкод дозволили досягти значного зменшення хибних спрацювань і покращення точності намотування. Результати тестування підтвердили, що система здатна адаптуватися до змін у процесі роботи та забезпечувати стабільну оботу при високій швидкості.

Робота має значний практичний зміст, адже впроваджена на виробництві 3D-друкарських матеріалів. Система, розроблена в цій роботі, є прикладом ефективного використання сучасних апаратних і програмних рішень для вирішення завдань промислової автоматизації.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Автоматизація виробничих процесів: навч. посібник / Федінець В.О.,Васильківський І.С., Николин Г.А.-Львів: СПОЛОМ, 2023. 120-150 с.

2. Л.М. Артюшин, О.А. Машков,Б.В. Дурняк,М.С. Сівов. Теоріяавтоматичного керування. — Львів: Видавництво УАД, 2004. 80-120с.

3. Бабіченко А.К. Промислові засоби автоматизації, Ч. 2 «Регулювальні і виконавчі пристрої»/А.К. Бабіченко, В.І. Тошинський, В.С. Михайлов та ін.– Х.: НТУ«ХПІ», 2003. 150-200 с.

4. Бабіченко А.К. Практикум з вимірювань та технічних засобів автоматизації/ А.К. Бабіченко, В.І. Тошинський, І.Л. Красніков та ін. – Х.: НТУ «ХПІ», 2009. 20-40 с.

5. Барало О.В. Автоматизація технологічних процесів і системи автоматичного керування: навчальний посібник / О.В. Барало, П.Г. Самойленко, С.Є.Гранат, В.О. Ковальов. – К.: Аграрна освіта, 2010. 100-350 с.

6. Bolton W. Mechatronics and Control of Electromechanical Systems. Pearson Education, 2015. С. 50-78 (інтеграція сенсорів та актуаторів), С. 200-225 (застосування мікроконтролерів у системах).

7. Hughes T. A. Industrial Automation and Control. ISA - The Instrumentation, Systems, and Automation Society, 2006. С. 120-150 (основи автоматизації), С. 300-320 (промислові стандарти керування).

8. Trzynadlowski A. M. Control of Induction Motors. Academic Press, 2000. С. 180-200.

9. Ott H. W. Electromagnetic Compatibility Engineering. John Wiley & Sons, 2009. С. 400-450 (фільтрація електромагнітних перешкод), С. 600-650.

10. Redwood B., Schöffer F., Garret B. The 3D Printing Handbook: Technologies, Design and Applications. 3D Hubs, 2017. С. 80-100 (матеріали для 3D-друку), С. 150-180 (вимоги до якості матеріалу).

11. Datasheet ATmega2560 Microcontroller. Atmel Corporation, 2014. С. 10-30 (порти вводу/виводу, апаратні переривання).

12. Bolton W. Programmable Logic Controllers. Elsevier, 2015. С. 75-95 (застосування програмованих контролерів у автоматизації), С. 200-230 (діагностика помилок у системах).

13. Dorf R. C., Bishop R. H. Modern Control Systems. Pearson, 2011. С. 320-350 (системи зворотного зв’язку), С. 500-540 (дискретні системи керування).

14. Groover M. P. Automation, Production Systems, and Computer-Integrated Manufacturing. 4th ed. Pearson, 2014. С. 150-170 (автоматизація виробничих процесів), С. 300-340 (системи управління якістю).

15. Ogata K. Modern Control Engineering. Prentice Hall, 2010. С. 100-130 (аналітичне моделювання), С. 400-450 (PID-регулювання).

16. Frank D. Petruzella. Electric Motors and Control Systems. McGraw-Hill, 2010. С. 50-80 (робота з серводвигунами), С. 150-170 (регулювання швидкості).

17. Hanselman D. Brushless Motors: Magnetic Design, Performance, and Control. E-Man Press LLC, 2006. С. 100-130 (дизайн безколекторних двигунів), С. 250-280 (контроль продуктивності).

18. Esposito A. Fluid Power with Applications. Pearson, 2013. С. 220-250 (гідравлічні та пневматичні системи в автоматизації), С. 450-480 (сенсори у системах).

19. Dixon J. C. The Shock Absorber Handbook. John Wiley & Sons, 2007. С. 300-330 (аналіз компенсуючих механізмів), С. 450-470 (оптимізація роботи пристроїв стабілізації).