Міністерство освіти і науки України
Український державний університет науки і технологій
Навчально-науковий інститут
«Український державний хіміко-технологічний університет»

Факультет комп'ютерних наук та інженерії
(повна назва факультету)

Кафедра комп'ютерно-інтегрованих технологій та робототехніки
(повна назва кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавра
(освітній рівень)

на тему Розробка та реалізація робота-маніпулятора з механічним захопленням з дистанційним керуванням

Виконав: студент 4 курсу, групи 4-АВП-22

спеціальності 151 Автоматизація та комп'ютерно-інтегровані технології
(шифр і назва напряму підготовки (професійна спрямованість), спеціальності)

СОЛОВЙОВ Д.П.
(прізвище та ініціали) (підпис)

Керівник ЛОСІХІН Д.А.
(прізвище та ініціали) (підпис)

Рецензент ЧЕРНЕЦЬКИЙ Є.В.
(прізвище та ініціали) (підпис)

Дніпро – 2026 року

Міністерство освіти і науки України
Український державний університет науки і технологій
(повне найменування вищого навчального закладу)

Навчально-науковий інститут
«Український державний хіміко-технологічний університет»
(назва навчально-наукового інституту)

Факультет, відділення *Комп'ютерних наук та інженерії*
Кафедра *Комп'ютерно-інтегрованих технологій та автоматизації*
Освітньо-кваліфікаційний рівень *Бакалавр*
Галузь знань *15 "Автоматизація та приладобудування"*
(шифр і назва)
Спеціальність *151 "Автоматизація та комп'ютерно-інтегровані технології"*
(шифр і назва)
Освітня програма *Автоматизація та комп'ютерно-інтегровані технології*

ЗАТВЕРДЖУЮ
Завідувач кафедри *КІТтаА*

" " 20 р.

ЗАВДАННЯ

НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТУ

Соловйов Денис Петрович
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Розробка та реалізація робота-маніпулятора з механічним захопленням з дистанційним керуванням

керівник проекту (роботи) Лосіхін Дмитро Анатолійович, старший викладач
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від " 02" березня 2026 р. № 77

2. Строк подання студентом проекту (роботи) 05.06.2026
3. Вихідні дані до проекту (роботи) Звіт з практики, літературні джерела, відповідні сторінки інтернет-сайтів

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1 Вступ. 2 Апаратне забезпечення. 3 Розробка технічної документації та реалізація системи. 4 Програмне забезпечення. 5 Охорона труда та техніка безпеки. 6 Економічні розрахунки. 7 Висновки. Література. Додатки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Схема електрична принципова підключень компонентів
Схема електрична принципова з'єднань компонентів
Специфікація устаткування на засоби автоматизації

6. Консультанти розділів проекту (роботи)

Розділ	Ініціали, прізвище, та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	*Д. А. Лосіхін, старший викладач*	20.03.2026	20.05.2026
Економічна частина	*Д. А. Лосіхін, старший викладач*	20.03.2026	01.06.2026

7. Дата видачі завдання “ 02 ” *березня* 20 26 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Вступ	18.03.2026	
2	Апаратне забезпечення	04.04.2026	
3	Розробка технічної документації та реалізація системи	20.04.2026	
4	Програмне забезпечення	14.05.2026	
5	Охорона труда та техніка безпеки	20.05.2026	
6	Економічні розрахунки	30.05.2026	
7	Висновки і література	01.06.2026	

Студент _____ Д. П. Соловйов
(підпис) (ініціали, прізвище)

Керівник проекту (роботи) _____ Д. А. Лосіхін
(підпис) (ініціали, прізвище)

№ рядка	Формат	Позначення	Найменування	Кіл. листів	№ Екз.	Примітка
			Документація загальна			
			Знов розроблена			
1	А1	А26.14.АРТ.01.СБ	Схема принципова електрична	1	1	
			підключень компонентів			
			платформи			
2	А1	А26.14.АРТ.02.СБ	Схема принципова електрична	1	1	
			з'єднань компонентів			
			платформи			
3	А4	А26.14.АРТ .00.ПЗ	Пояснювальна записка	158	1	
4	А1	А26.14.АРТ.00.СО1	Специфікація устаткування	2	1	

				А26.14.АРТ.00.ВД		
				Дніпро, УДУНТ, ННІ УДХТУ		
Виконав	Соловйов		02.06	Розробка та реалізація робота маніпулятора з механічним захопленням з дистанційним керуванням	Стадія: ДП	Лист: 1 / Листів: 1
Перевірив	Лосіхін					
Рецензент	Чернецький					
Затвердив	Левчук			Відомість дипломного проекту	УДУНТ, ННІ УДХТУ, кафедра КІТтаР гр. 4-АВП-22	

РЕФЕРАТ

Пояснювальна записка дипломного проекту містить:

сторінок - 158;

таблиць - 32;

схем і рисунків – 13;

літературних джерел - 14.

Відповідно до завдання до кваліфікаційної роботи у спеціальній частині проекту: здійснений вибір технічних засобів; розроблено монтажну плату для розміщення компонентів системи; розроблено таблиці з’єднань, схеми електричні принципіальні з’єднань та підключень компонентів системи; розроблено програмне забезпечення системи; методика числового програмного управління.

У розділі “Охорона праці і техніка безпеки” виконаний опис заходів по створенню безпечних та здорових умов праці.

В економічній частині виконано розрахунок затрат на проектування та виготовлення прототипу системи, розрахунок точки беззбитковості. Після продажу мінімальної партії 551, шт., кожен наступний продаж приноситиме 1150 грн чистого прибутку.

Ключові слова: РОБОТ МАНІПУЛЯТОР, МЕХАНІЧНЕ ЗАХОПЛЕННЯ ДИСТАНЦІЙНЕ КЕРУВАННЯ, ЧИСЛОВЕ ПРОГРАМНЕ УПРАВЛІННЯ, МІКРОКОНТРОЛЕР, ПРОГРАМУВАННЯ C/C++.

				А26.14.АРТ.00.ПЗ			
				Дніпро, УДУНТ, ННІ УДХТУ			
Виконав	Соловйов		02.06	Розробка та реалізація робота маніпулятора з механічним захопленням з дистанційним керуванням	Стадія	Лист	Листів
Перевірив	Лосіхін				ДП	1	158
Рецензент	Чернецький						
				Пояснювальна записка	УДУНТ, ННІ УДХТУ, кафедра КІТтаР гр. 4-АВП-22		
Затвердив	Левчук						

Зміст

1 ВСТУП

Роботизовані маніпулятори є одним із найпоширеніших прикладів мехатронних систем, у яких поєднуються механічні вузли, електронні модулі керування та програмне забезпечення. Навіть малогабаритна роботизована рука дає можливість наочно дослідити принципи керування виконавчими ланками, позиціювання сервоприводів, обробки сигналів від органів ручного керування та передавання команд через бездротовий канал зв’язку. Саме тому такі пристрої широко використовуються в навчальних стендах, лабораторних роботах, прототипуванні простих автоматизованих механізмів і під час вивчення основ робототехніки.

Актуальність теми дипломного проєкту зумовлена потребою у створенні доступного робототехнічного комплексу, який можна не лише описати теоретично, а й реалізувати на реальній елементній базі. У межах роботи розглядається роботизована рука з механічним захопленням, побудована на основі Arduino-сумісної керуючої плати, чотирьох сервоприводів MG90S, плати спряження для підключення виконавчих механізмів, двох аналогових джойстиків і Bluetooth-модуля HC-05. Такий набір компонентів дозволяє реалізувати компактний навчально-дослідний стенд, придатний для перевірки різних способів керування маніпулятором.

Особливістю розроблюваної системи є підтримка кількох режимів роботи. Ручне керування здійснюється за допомогою двох джойстиків, що дозволяє оператору безпосередньо змінювати положення основи, плечових ланок і захвату. Бездротовий режим передбачає передавання команд через Bluetooth-з’єднання, що розширює можливості дистанційного керування. Окремо розглядається програмний режим, у якому положення маніпулятора можуть запам’ятовуватися та відтворюватися у вигляді заданої послідовності рухів. Це робить пристрій придатним не тільки для демонстрації ручного

керування, а й для відпрацювання елементарних сценаріїв автоматизованої роботи.

Об’єктом дослідження є роботизована рука як мехатронна система маніпуляційного типу. Предметом дослідження є апаратні та програмні засоби побудови системи керування роботизованою рукою на базі мікроконтролерної платформи Arduino, а також способи організації ручного, програмного та бездротового керування її виконавчими ланками.

Метою дипломного проєкту є розроблення роботизованої руки з механічним захопленням, яка забезпечує виконання базових маніпуляційних операцій і підтримує кілька режимів керування. Методичною основою роботи є принципи побудови вбудованих систем керування, методи програмування мікроконтролерів у середовищі Arduino IDE, основи керування сервоприводами за широтно-імпульсним сигналом, а також методи модульного проєктування мехатронних пристроїв. У процесі розроблення використовуються засоби експериментальної перевірки працездатності програмних модулів, повузлового тестування обладнання та послідовної інтеграції апаратних і програмних компонентів.

Практична цінність розробки полягає в можливості використання створеної роботизованої руки як навчального стенда для вивчення робототехніки, мікроконтролерної техніки та основ числового програмного управління. Розроблений пристрій може бути застосований для демонстрації принципів позиціювання ланок, відпрацювання алгоритмів захоплення та переміщення предметів, опанування технологій дистанційного керування та створення простих сценаріїв автоматичного виконання послідовностей дій. Крім того, отримані результати можуть слугувати основою для подальшої модернізації системи, наприклад за рахунок установлення додаткових датчиків, розширення кількості ступенів вільності або впровадження алгоритмів зворотного зв’язку.

2 АПАРАТНЕ ЗАБЕЗПЕЧЕННЯ

Апаратна частина роботизованої руки визначає її функціональні можливості, точність позиціювання, надійність роботи та зручність подальшої програмної реалізації. Під час проектування пристрою потрібно було підібрати такі компоненти, які забезпечують достатню вантажопідйомність ланок, можливість незалежного керування ступенями вільності, сумісність із поширеним середовищем розроблення та простоту інтеграції в єдину систему. Виходячи з цих вимог, апаратну платформу було сформовано на основі мікроконтролера сімейства Arduino, чотирьох сервоприводів, плати розподілу живлення та сигналів керування, двох аналогових джойстиків, Bluetooth-модуля та допоміжних конструктивних елементів.

Принцип побудови апаратної частини полягає в поділі системи на кілька функціональних рівнів: рівень керування, виконавчий рівень, рівень ручного введення команд, рівень бездротового зв’язку та рівень живлення. Такий підхід спрощує модернізацію пристрою, полегшує пошук несправностей і забезпечує можливість за потреби замінити окремі елементи без зміни загальної архітектури.

2.1Мікроконтролер

Як центральний елемент системи керування обрано мікроконтролер на AVR-архітектурі, сумісний із платформою Arduino UNO (рис. 2.1). Такий вибір зумовлений кількома причинами. По-перше, це сімейство мікроконтролерів є широко поширеним, що забезпечує наявність великої кількості готових бібліотек для роботи із сервоприводами, послідовним інтерфейсом UART і користувацькими пристроями введення. По-друге, мікроконтролер має достатню кількість цифрових і аналогових виводів для підключення всіх необхідних модулів роботизованої руки.

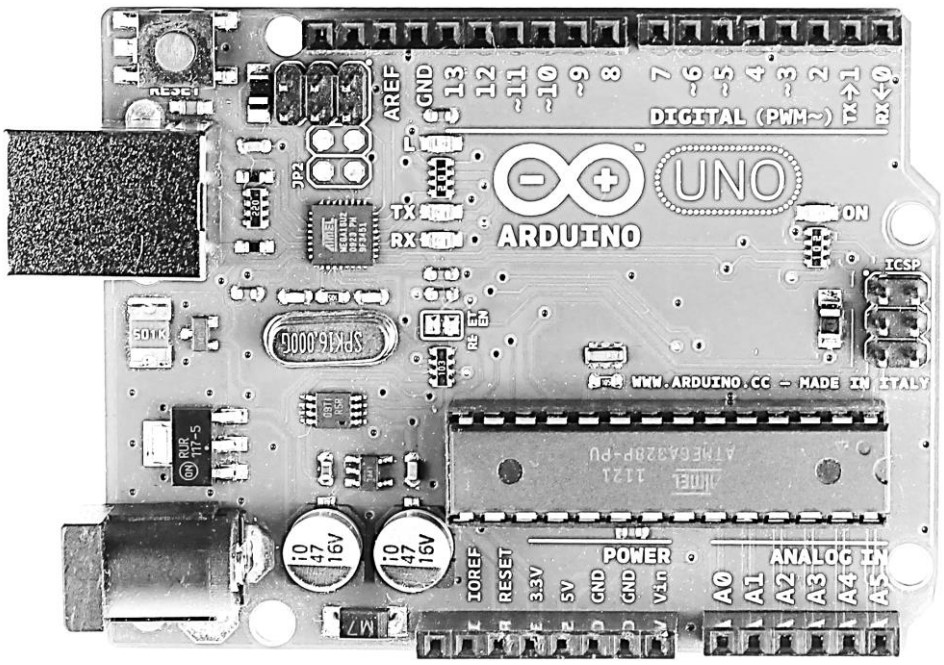


Рисунок 2.1 – плата Arduino Uno

Контролер виконує роль вузла координації. На нього покладаються функції зчитування сигналів із джойстиків, приймання команд бездротовим каналом зв’язку, генерації керуючих імпульсів для сервоприводів, збереження поточних положень ланок і реалізації сценаріїв руху. Тактова частота 16 МГц є достатньою для завдань цього класу, оскільки керування сервоприводами не потребує значних обчислювальних ресурсів, а алгоритми позиціонування та користувацького введення мають порівняно невисоку обчислювальну складність.

Додатковою перевагою Arduino-сумісного рішення є зручність налагодження. Підключення до персонального комп’ютера здійснюється через USB-інтерфейс, що створює можливість завантажувати керуючі програми, використовувати послідовний монітор для діагностики та швидко коригувати параметри системи. Завдяки цьому зменшується час розроблення та підвищується відтворюваність результатів під час виконання експериментів.

З погляду дипломного проекту такий мікроконтролер є раціональним компромісом між вартістю, функціональністю та простотою впровадження. Його можливостей достатньо для керування чотирьох ланковою

роботизованою рукою, а при переході до складніших алгоритмів архітектура системи може бути збережена із заміною лише обчислювального вузла.

В апаратній реалізації керуючого вузла основну обчислювальну функцію виконує мікросхема ATmega328P-PU, а обмін даними з комп'ютером і завантаження прошивки забезпечує перетворювач інтерфейсу USB–UART CP2102. Таким чином, загальні вимоги проекту щодо зручності розроблення, достатньої кількості інтерфейсів і стабільної роботи реалізуються через конкретний набір електронних компонентів.

Мікросхема ATmega328P-PU належить до 8-розрядних AVR-мікроконтролерів і широко застосовується у вбудованих системах керування. Її можливостей достатньо для розв'язання завдань навчально-дослідного та прикладного рівня, пов'язаних із керуванням робототехнічними пристроями малої та середньої складності. Основні характеристики мікроконтролера наведено в таблиці 2.1.

Наявність 14 цифрових ліній введення-виведення та 6 аналогових входів дозволяє підключити сервоприводи, джойстики, датчики, модулі зв'язку та додаткові виконавчі пристрої без суттєвого ускладнення схеми. Особливо важливою є наявність PWM-каналів, оскільки вони використовуються під час формування керуючих сигналів для приводів. Апаратні інтерфейси UART, SPI та I^2C створюють основу для підключення периферійних пристроїв – дисплеїв, датчиків положення, модулів бездротового зв'язку, пристроїв реєстрації даних та інших компонентів.

Таблиця 2.1 – Основні характеристики мікроконтролера ATmega328P-PU

Параметр	Значення
Тип мікроконтролера	8-разрядний AVR
Модель	ATmega328P-PU
Тактова частота	16 МГц
Робоча напруга	5 В
Flash-пам’ять програм	32 КБ
SRAM	2 КБ
EEPROM	1 КБ
Кількість цифрових ліній вводу-виводу	14
Кількість PWM-каналів	6
Кількість аналогових входів	6
Апаратні інтерфейси	UART, SPI, I²C (TWI)
Допустимість переривань	Підтримуються зовнішні та внутрішні переривання

Для програмування мікроконтролера й обміну службовими даними з комп’ютером у складі плати застосовується мікросхема CP2102. Вона виконує перетворення сигналів USB на послідовний інтерфейс UART і тим самим забезпечує завантаження програмного коду, виведення діагностичної інформації та налаштування параметрів роботи системи. Основні характеристики цього перетворювача наведено в таблиці 2.2.

Використання CP2102 позитивно впливає на зручність експлуатації роботизованої руки. Завдяки наявності USB-підключення спрощується процес програмування, не потрібен зовнішній програматор, а передавання діагностичних даних може здійснюватися безпосередньо до середовища розроблення або термінальної програми. Це особливо важливо на етапах налаштування положення приводів, перевірки алгоритмів руху та тестування взаємодії із зовнішніми модулями.

З погляду обчислювальної потужності обраний мікроконтролер є достатнім для реалізації алгоритмів керування роботизованою рукою з чотирьма ступенями вільності. Тактова частота 16 МГц забезпечує впевнене виконання циклів опитування органів керування, генерації команд керування і обміну даними без помітних затримок для користувача. Обсягу пам’яті

достатньо для розміщення програми керування, таблиць положень, параметрів калібрування та сервісних процедур.

Таблиця 2.2 – Основні характеристики перетворювача інтерфейсу CP2102

Параметр	Значення
Найменування	CP2102
Інтерфейс зі сторони ПК	USB 2.0
Інтерфейс зі сторони МК	UART/TTL
Підтримуються сигнали	TXD, RXD і службові лінії керування
Напруга	3,3 В / сумісність з TTL-інтерфейсами
Швидкість послідовного обміну	До 1 Мбіт/с
Наявність вбудованого генератора	Так
Підтримка драйверів	Windows, Linux, macOS
Призначення в проекті	Завантаження прошивки, налагодження, обмін даними з ПК

Наявний запас за кількістю виводів, вбудованими інтерфейсами та обсягом пам’яті забезпечує можливість в подальшому розширити функціональні можливості роботизованої руки. У перспективі це відкриває можливість підключити додаткові датчики, модуль індикації, засоби бездротової телеметрії, елементи зворотного зв’язку, систему запам’ятовування траєкторій або більш розвинені алгоритми керування без заміни базової архітектури.

2.2Електродвигуни виконавчих ланок

Для переміщення ланок роботизованої руки доцільно застосовувати не двигуни постійного струму загального призначення, а сервоприводи. У цій системі використано сервоприводи з робочим діапазоном повороту до 180°. Такий вибір пов’язаний із тим, що сервопривод уже містить усередині редуктор, схему керування та зворотний зв’язок за положенням вихідного валу. Це дозволяє отримувати заданий кут повороту на пристрої

безпосередньо за імпульсом керування без потреби проектувати окремий контур регулювання положення.

Кожен сервопривод виконує власну функцію в кінематичній схемі руки. Перший привід відповідає за обертання основи навколо вертикальної осі, другий керує підніманням основного плеча, третій – зміною положення передпліччя, а четвертий забезпечує розкриття та закриття захвату. Така конфігурація відповідає чотирьом ступеням вільності й дає можливість формувати типові траєкторії для захоплення та переміщення об’єктів.

Перевага сервоприводів виявляється також у спрощенні механічної частини. При цьому слід ураховувати, що реальні межі повороту кожного привода визначаються не лише його електричними характеристиками, а й геометрією конструкції. Тому в програмному забезпеченні мають бути задані обмеження кутів, які унеможливлюють заклинювання механізму та надмірне навантаження на вузли.

Як виконавчі приводи роботизованої руки обрано мікро-сервоприводи MG90S масою 14 г (рис. 2.2).

Рисунок 2.2 – мікро-сервопривод MG90S

Для конструкції, що розглядається застосування саме сервоприводів є більш раціональним, ніж використання звичайних двигунів постійного

струму, оскільки сервопривод поєднує електродвигун, редуктор, схему керування та вбудований контроль за положенням вихідного вала.

Використання MG90S виправдане тим, що роботизована рука належить до малогабаритних навчально-дослідних систем. У такій конструкції важливими є не лише зусилля й точність утримання положення, а й обмежена маса привода. Надмірно важкі двигуни збільшували б навантаження на сусідні ланки, погіршували б динаміку й потребували б жорсткішої та дорожчої механічної частини. Тому компактний сервопривод із металевим редуктором є збалансованим рішенням за масою, габаритами та моментом сили.

Основні характеристики сервоприводу MG90S наведено в таблиці 2.3.

Таблиця 2.3 – Основні характеристики сервопривода MG90S

Параметр	**Значення**
Модель	MG90S Micro Servo
Тип привода	Позиційний сервопривід постійного струму
Маса	14 г
Робоча напруга	4,8–6 В
Сигнал керування	PWM
Крутний момент	1,8 кг·см при 4,8 В; до 2,2 кг·см при 6 В
Швидкість	0,15 с/60° при 4,8 В; 0,12 с/60° при 6 В
Діапазон повороту	до 180°
Редуктор	Металевий
Габаритні розміри	близько 22,8 × 12,2 × 28,5 мм
Струм без навантаження	70–90 мА
Струм під робочим навантаженням	200 мА
Піковий струм / струм при заклинюванні	Орієнтовно 860 мА ±10% при 6 В

Для роботизованої руки ці характеристики є достатніми. Крутний момент порядку 1,8–2,2 кг·см дає можливість впевнено переміщувати легкі ланки механізму та виконувати операції захоплення невеликих об’єктів без потреби застосування масивніших приводів. Наявність металевого редуктора підвищує зносостійкість виконавчого вузла порівняно з пластиковими аналогами, що особливо важливо за багаторазових циклів відкриття захвату та переміщення плечових ланок.

Швидкість 0,12–0,15 с на 60° забезпечує достатньо швидке, але водночас кероване переміщення ланок. Для навчальної роботизованої руки це є перевагою, оскільки дозволяє реалізувати плавне позиціювання, не створюючи надмірних ударних навантажень на каркас і кріпильні елементи. Робоча напруга 4,8–6 В добре узгоджується з типовою низьковольтною схемою живлення, що застосовується спільно з Arduino-сумісним контролером і зовнішнім джерелом живлення.

Маса одного привода, що становить 14 г, також має принципове значення. Мала вага зменшує навантаження на сусідні ланки й основу, завдяки чому навіть за послідовного встановлення кількох сервоприводів конструкція залишається достатньо легкою та не потребує використання посилених механічних опор. Це спрощує виготовлення і знижує вартість усієї системи.

2.3Драйвер електродвигунів і плата спряження

Сервоприводи містять вбудовану електроніку керування, але для практичної реалізації системи потрібен вузол, який спрощує їх підключення до мікроконтролера та організовує розподіл живлення. Цю функцію виконує плата спряження, що встановлюється над контролером і надає роз’єми для сигнальних, живильних і спільних провідників.

Застосування такої плати має кілька переваг:

- спрощується комутація: сервоприводи підключаються до готових роз’ємів без виготовлення додаткової проводки;
- знижується ймовірність помилок монтажу, оскільки виводи живлення, землі та сигнального каналу згруповані у стандартному порядку;
- з’являється можливість компактно розмістити на одній платі додаткові інтерфейси, наприклад роз’єми для Bluetooth-модуля, джойстика або зовнішніх модулів керування.

Під час одночасної роботи кількох сервоприводів струм споживання системи помітно зростає, особливо в моменти старту, різкої зміни положення або утримання механічного навантаження. Тому плата спряження має забезпечувати підведення зовнішнього живлення до виконавчих механізмів і не перевантажувати стабілізатор самого мікроконтролера. Фактично цей вузол виконує роль силового та комутаційного центру апаратної частини.

У розглядуваній системі плата спряження (рис. 2.3) одночасно виконує дві задачі: розподіляє сигнали керування від виводів Arduino до сервоприводів і формує окрему силову шину живлення для виконавчих механізмів. Інакше кажучи, керувальний сигнал надходить з мікроконтролера на сигнальний контакт відповідного роз’єму, а живлення на сервоприводи подається вже через саму плату спряження від зовнішнього джерела. Таке рішення підвищує надійність роботи системи та знижує ймовірність просідання напруги на платі керування.

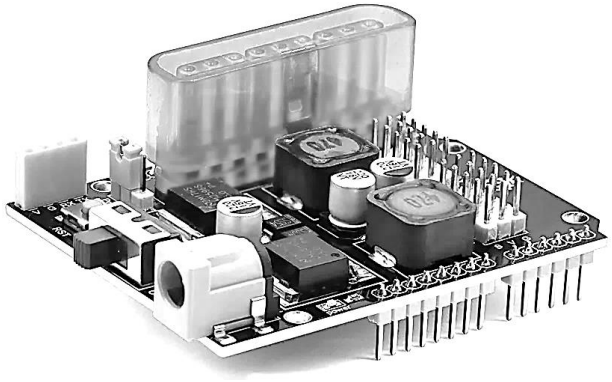


Рисунок 2.3 – плата спряження

До складу силової частини плати спряження входять імпульсні понижувальні стабілізатори на базі мікросхем LM2596S. Вони перетворюють вхідну напругу VIN на стабілізовану напругу 5 В, що використовується для живлення периферійних вузлів і сервоприводів. Застосування імпульсного перетворення є доцільнішим за лінійну стабілізацію, оскільки за помітних струмів навантаження забезпечується вищий ККД і менший нагрів плати.

Важливу роль у роботі понижувальних перетворювачів відіграють дроселі. Дросель є накопичувальним елементом імпульсного DC–DC-перетворювача: під час відкритого стану ключа він накопичує енергію в магнітному полі, а після закриття ключа передає її в навантаження та вихідний конденсатор. У досліджуваному шилді використовуються силові дроселі з маркуванням «470», що відповідає номіналу близько 47 мкГн. Цей елемент зменшує пульсації струму, стабілізує режим перетворення та підвищує стійкість роботи джерела живлення за змінного навантаження.

Крім того, у схемі застосовуються силові ключові елементи. У понижувальному перетворювачі ключовий режим роботи реалізовано всередині мікросхеми LM2596S, яка періодично комутує струм через дросель із робочою частотою 150 кГц. В інших вузлах плати використовуються польові транзистори, що виконують функції електронних ключів для узгодження та комутації сигналів. Використання ключового режиму забезпечує можливість зменшити теплові втрати та забезпечити живлення навантаження за порівняно великого струму.

Плата спряження є важливим компонентом, що підвищує технологічність складання, надійність з’єднань і зручність подальшої експлуатації роботизованої руки. Наявність стандартизованих роз’ємів, окремого зовнішнього живлення та вбудованих силових перетворювачів робить цей вузол центральним елементом апаратної інтеграції виконавчої системи.

Основні технічні характеристики плати наведено в таблиці 2.4.

Таблиця 2.4 – Основні характеристики плати спряження

Параметр	Значення
Вхідна напруга VIN	DC 7–9 В
Вхідний струм VIN	не більше 5 А
Вихідна напруга	5 В
Вихідний струм	до 3 А
Мікросхема стабілізатора	LM2596S
Частота перетворення	150 кГц
Робоча температура	0–50 °C

2.4Пристрої ручного керування

Для локального керування положенням роботизованої руки було обрано два аналогові джойстики. Кожен джойстик формує два аналогові сигнали за осями X і Y, а також цифровий сигнал від вбудованої кнопки. Таке рішення зручне тим, що оператор може одночасно змінювати положення кількох ланок, інтуїтивно співвідносячи рух рукоятки з рухом механізму.

Використання пари джойстиків дозволяє логічно розділити функції керування. Один джойстик може бути призначений для повороту основи та переміщення однієї з ланок, другий – для керування іншою ланкою і механічним захопленням . Подібна схема зменшує когнітивне навантаження на оператора та забезпечує плавніше керування порівняно з кнопковим інтерфейсом.

З апаратної точки зору джойстики зручні тим, що підключаються безпосередньо до аналогових входів мікроконтролера. Значення напруги зчитуються вбудованим аналого-цифровим перетворювачем і далі програмно перетворюються на команди зміни кута сервоприводів. Наявність кнопок у складі модулів додатково дозволяє реалізувати функції запису поз, запуску сценаріїв руху або перемикання режимів.

Принципово аналоговий джойстик (рис. 2.4) є електромеханічним модулем, до складу якого входять два потенціометри, розташовані за взаємно

перпендикулярними осями, і тактова кнопка, що спрацьовує під час натискання на рукоятку. У разі відхилення рукоятки змінюється положення повзунків потенціометрів, унаслідок чого змінюється вихідна напруга на каналах X і Y. У нейтральному положенні значення за обома осями близькі до середнього рівня, а під час зміщення в один із боків значення зростає або зменшується залежно від напряму відхилення.

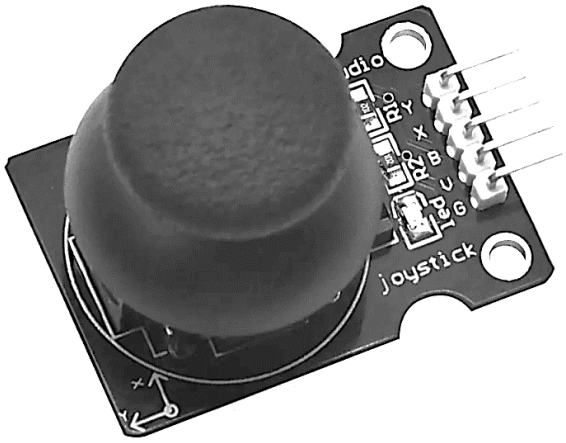


Рисунок 2.4 – аналоговий джойстик

Отже, джойстик забезпечує просте й наочне перетворення механічного руху оператора на електричні сигнали, які потім інтерпретуються мікроконтролером як команди керування відповідними сервоприводами роботизованої руки. Це робить цей тип пристрою зручним для ручного позиціювання маніпулятора, навчання рухів і реалізації напівавтоматичних режимів керування.

Таблиця 2.5 – Основні характеристики аналогового джойстика

Параметр	Характеристика
Тип пристрою	Аналоговий двовісний джойстик із кнопкою
Основні елементи	Два потенціометри за осями X і Y, тактова кнопка
Кількість керованих осей	2
Вихідні сигнали	Два аналогові сигнали та один цифровий сигнал
Підключення	Безпосередньо до аналогових і цифрових входів мікроконтролера
Принцип роботи	Зміна положення змінює опір потенціометрів і вихідну напругу на каналах X і Y
Призначення в системі	Ручне керування ланками маніпулятора, захватом і додатковими функціями

2.5 Бездротовий інтерфейс Bluetooth

Для розширення можливостей керування до апаратної структури введено Bluetooth-модуль з UART-інтерфейсом. Його призначення полягає в організації бездротового обміну даними між роботизованою рукою та зовнішнім пристроєм, наприклад смартфоном або планшетом. Бездротовий канал є зручним у випадках, коли необхідно керувати механізмом на відстані, демонструвати роботу пристрою або використовувати графічний інтерфейс мобільного застосунку.

Вибір Bluetooth-інтерфейсу зумовлений його поширеністю, низьким енергоспоживанням і простотою інтеграції з мікроконтролером. Для обміну командами достатньо послідовного з’єднання по лініях RX і TX. Це спрощує схемотехнічну частину та дозволяє передавати окремі керувальні символи або короткі пакети даних, що кодують дії виконавчих ланок.

Наявність Bluetooth-модуля робить апаратну платформу гнучкішою. Надалі це забезпечує можливість перейти від прямого ручного керування до віддаленого керування через застосунок, а також використовувати модуль як канал налаштування, діагностики або обміну з зовнішньою обчислювальною системою.

Як конкретне рішення для реалізації бездротового каналу зв’язку обрано модуль HC-05 (рис 2.5). Цей модуль широко застосовується в навчальних і аматорських проєктах на базі Arduino, оскільки забезпечує просте підключення через послідовний інтерфейс UART і не потребує складного налаштування на початковому етапі. HC-05 працює в режимі веденого пристрою, що зручно для системи роботизованої руки, де керувальним пристроєм виступає смартфон або планшет, а сам модуль приймає команди й передає їх мікроконтролеру.

У цій системі HC-05 використовується як послідовний бездротовий канал: отримані символи або короткі команди передаються мікроконтролеру, після чого програма змінює положення відповідних сервоприводів.

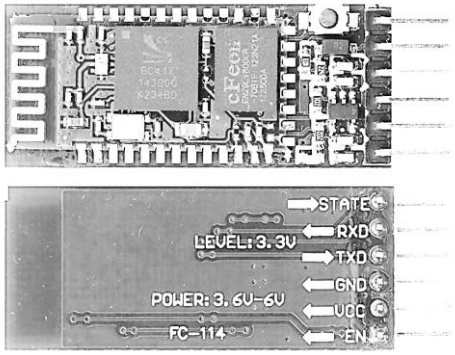


Рисунок 2.5 – Bluetooth модуль HC-05

Таблиця 2.6 – Основні характеристики Bluetooth-модуля HC-05

Параметр	**Значення**
Тип інтерфейсу	UART (TTL)
Режим роботи	Master / Slave
Стандарт Bluetooth	Bluetooth 2.0 + EDR
Робочий діапазон частот	2,4 ГГц
Напруга живлення модуля	3,6–6 В
Логічний рівень RX	3,3 В
Швидкість обміну за замовчуванням	9600 бод
Формат послідовного обміну	8 біт даних, 1 стоп-біт, без контролю парності (8-N-1)
Дальність зв’язку	до 10 м у типових умовах
Типове споживання струму	близько 30–40 мА
Основне призначення	Бездротова передача керувальних команд від смартфона до мікроконтролера

2.6Система живлення та конструктивні елементи

Надійна робота роботизованої руки значною мірою залежить від правильно організованого живлення. Мікроконтролер і модулі введення можуть споживати відносно невеликий струм, тоді як сервоприводи потребують значно більшого енергоспоживання, особливо під час різких переміщень або утримання вантажу. Тому для системи необхідно передбачити зовнішнє джерело живлення з напругою, що відповідає вимогам контролера та плати спряження.

Під час проектування також враховуються конструктивні елементи – основа, стійки, важелі, кріплення, майданчики для монтажу електроніки та з'єднувальні проводи. Саме механічна частина визначає підсумкову жорсткість конструкції, плавність руху та допустиме навантаження на приводні вузли. Для зменшення маси можуть використовуватися акрилові або інші легкі матеріали, а для монтажу електроніки – стійки та гвинтові з'єднання, що забезпечують ремонтопридатність конструкції.

Окрему увагу слід приділяти прокладанню проводів. Під час руху ланок вони не повинні натягуватися, потрапляти в зону захоплення або обмежувати робочий хід механізмів. Тому під час складання необхідно застосовувати стяжки, гнучкі з'єднувачі та продумане розміщення кабелів уздовж рухомих елементів.

Для живлення сервоприводів за наявності плати спряження використовується зовнішнє джерело з напругою 7–8 В, яке має забезпечувати струм, достатній для одночасної роботи виконавчих механізмів. Для цього вистачить двох послідовно з'єднаних акумуляторних елементів типорозміру 18650 у відповідному утримувачі з конектором для підключення до гнізда живлення (рис 2.6)

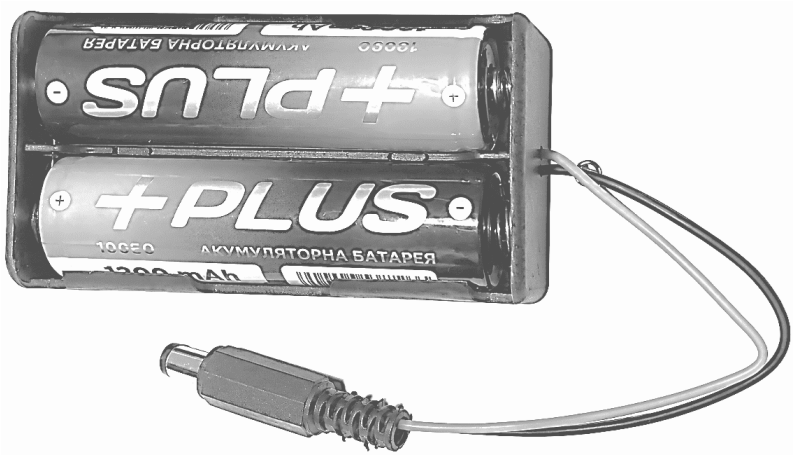


Рисунок 2.6 – утримувач з двома послідовно з’єднаними елементами 18650

Для поєднання усіх елементів виробу будемо застосовувати готовий набір шасі для робота-маніпулятора із захватом на акриловій основі. До набору входять необхідні кріпильні елементи – болти, муфти, гайки тощо.

Електричні з’єднання будемо виконувати за допомогою кабелів DuPont Female-Female та Female-Male, що є типовим способом з’єднання для проєктів на базі Arduino.

3 РОЗРОБКА ТЕХНІЧНОЇ ДОКУМЕНТАЦІЇ ТА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1Пульт прямого керування

Призначення виводів аналогового джойстику наведено у таблиці 3.1.

Таблиця 3.1 – Призначення виводів аналогового джойстику

Позначення	Призначення
V	Живлення 5В
G	Ground, земля
B	Button, сигнал кнопки
X	Аналоговий сигнал положення по горизонталі
Y	Аналоговий сигнал положення по вертикалі

```
const byte PIN_B = A0;
const byte PIN_X = A1;
const byte PIN_Y = A2;

void setup() {
  Serial.begin(9600);
  pinMode(PIN_B, INPUT_PULLUP);
}
void loop() {
  Serial.print("X=");
  Serial.print(analogRead(PIN_X));
  Serial.print(" Y=");
  Serial.print(analogRead(PIN_Y));
  Serial.print(" B=");
  Serial.println(digitalRead(PIN_B) == LOW ? 1 : 0);
  delay(100);
}
```

Рисунок 3.1 – скетч для перевірки працездатності джойстиків.

Перевірку працездатності джойстиків виконано до остаточного монтажу пульта. Кожен модуль по черзі підключався безпосередньо до плати контролера, після чого за допомогою тестового скетча (рис. 3.1) контролювалися значення аналогових осей і стан кнопки. Під час переміщення

рукоятки за кожною віссю значення, що виводилися в послідовний монітор, змінювалися в межах від 0 до 1023. Натискання кнопки змінювало її цифровий стан з 0 на 1 і навпаки, що підтвердило справність модулів і правильність підключення сигнальних ліній.

Збірку маніпулятора виконано на акриловій платі із підключенням до контактів за допомогою шлейфів кабелю F-F (рис. 3.2).

Рисунок 3.2 – пульт у зборі

Плату спряження встановлено поверх контролера, після чого виконано підключення готового пульта відповідно до таблиці 3.2.

Таблиця 3.2 – Підключення пульту до плати спряження

Джойстик	**Контакт на платі спряження**	**Рядок контактів**
Правий джойстик		
V	V	A3 (S)
G	G	A3 (S)
B	7	7 (S)
X	A2	A2 (S)
Y	A5	A5 (S)
Лівий джойстик		
V	V	7 (S)
G	G	7 (S)
B	6	6 (S)
X	A3	A3 (S)
Y	A4	A4 (S)

Схема з’єднань показана на рис. 3.3.

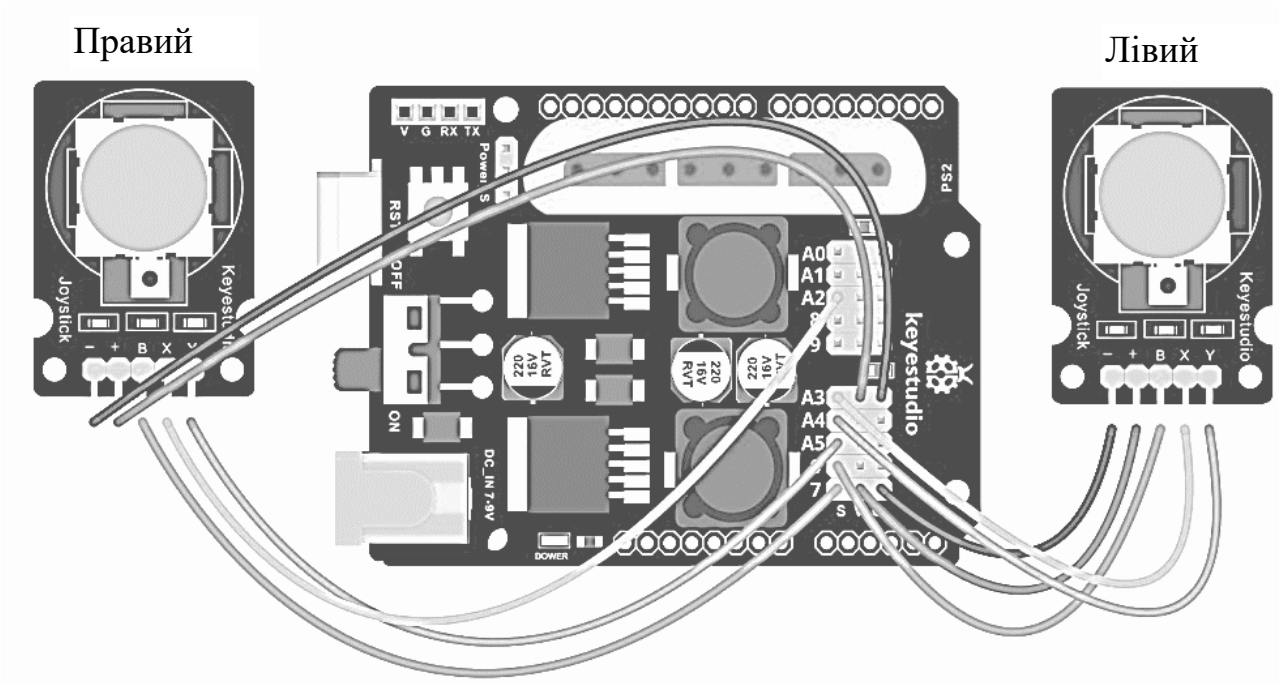


Рисунок 3.3 – підключення пульту до плати спряження

Така схема підключення обрана щоб дроти пульту займали переважно нижню частину контактів плати спряження і не заважали підключенню сервомоторів.

3.2 Виконавча частина маніпулятора

Складання виконавчої частини розпочато з установлення на акрилову основу утримувача акумуляторів, контролера та плати спряження. Після цього виконано послідовну перевірку кожного сервопривода. Перед монтажем у шасі кожен сервопривод було встановлено в початкове положення відповідно до таблиці 3.4. Така операція потрібна для правильного суміщення положення вала сервопривода з механічним положенням ланки маніпулятора та для запобігання заклинюванню механізму під час першого ввімкнення.

Призначення виводів сервопривода наведено у таблиці 3.3.

Таблиця 3.3 – Призначення виводів сервоприводу

Колір і позначення	Призначення
Сірий, G	Ground, земля
Червоний, V	Живлення
Жовтий, S	S, сигнал керування

```
#include <Servo.h>
int pin = A0, angle = 90;
Servo s;

void setup() {
  s.attach(pin);
  s.write(0);
  s.write(180);
  s.write(angle);
}
void loop() {}
```

Рисунок 3.4 – скетч для перевірки працездатності і позиціювання сервоприводів

Таблиця 3.4 – Підключення сервоприводів

Сервопривод	Кут, °	G/V/S, рядок	Пояснення по встановленню
Servo1, поворот	90	A1	Знизу, у нейтральному положенні повороту
Servo2, підйом	90	A0	Лівий, опущений, ручка вертикально
Servo3, висування	90	8	Правий, ручка вертикально, висунутий наполовину
Servo4, захоплення	0	9	Захват повністю закритий. Додатково підключається через дроти DuPont M-F, щоб подовжити площу досяжності.

Після встановлення шасі закріплене на акриловій платформі на підйомних муфтах. Виконано підключення сервоприводів до плати спряження. Виконано підключення живлення батарейного відсіку до плати спряження.

Дроти зібрані нижче зони роботи маніпулятора за допомогою пластикових стяжок. Дріт сервомотора захоплення ізольований від механічних пошкоджень за допомогою гнучкої шини. Виріб набув кінцевого виду (рис. 3.5).

Рисунок 3.5 – зібрана виконавча частина з під’єднаним пультом

Слід зазначити, що при подальшому використовуванні робота не можна використовувати живлення тільки за USB, слід вмикати живлення на платі спряження з живленням від акумуляторів.

3.3 Підключення Bluetooth

Підключення Bluetooth модуля HC-05 виконується згідно до таблиці 3.5. На платі спряження є окремий набір контактів саме для Bluetooth модуля.

Під час програмування контролера модуль слід виймати з плати, оскільки прошивка нової програми використовує ті ж самі лінії TX/RX, що і модуль Bluetooth.

Таблиця 3.5 – Підключення Bluetooth модуля HC-05

Контакт	Контакт на платі спряження
RX	TX
TX	RX
G	G
V	V

Працездатність модуля Bluetooth перевірена за допомогою скетча (рис. 3.6), та додатку для смартфону «Serial Bluetooth Terminal» [10].

```
String inputLine = "";

void setup() {
  Serial.begin(9600);
}
void loop() {
  while (Serial.available()) {
    char ch = Serial.read();

    if (ch == '\r') continue;       // Ігноруємо CR
    if (ch != '\n') inputLine += ch; // Додаємо символ до рядка
    else if (inputLine.length()) {   // Кінець рядка
      Serial.print("[RECEIVED] ");
      Serial.println(inputLine);
      inputLine = "";
    }
  }
}
```

Рисунок 3.6 – скетч для перевірки Bluetooth

Кроки для перевірки, треба:

- витягнути модуль HC-05;
- завантажити скетч;
- встановити модуль HC-05;
- увімкнути Bluetooth на смартфоні;
- у додатку обрати меню->Devices;
- натиснути SCAN;
- обрати HC-05 і дозволити поєднання с пристроєм;
- відправити будь яку строку і впевнитися що прийшов відгук (рис. 3.7).

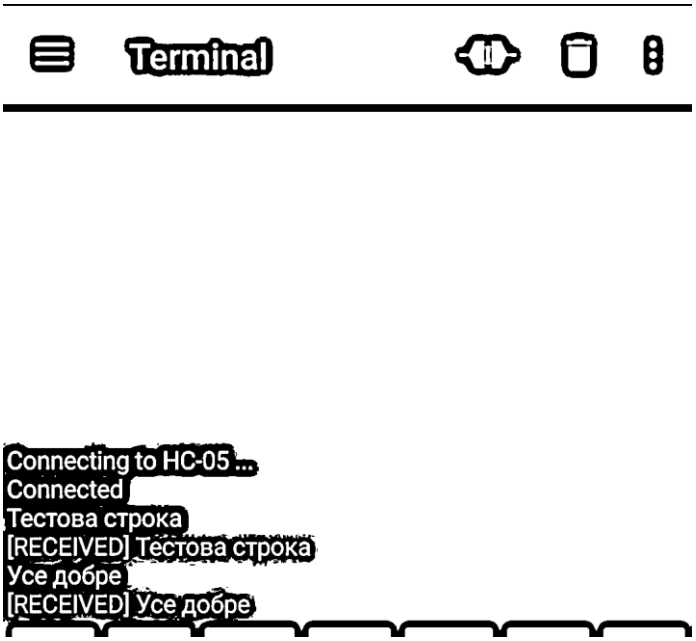


Рисунок 3.7 – результат перевірки модуля Bluetooth

3.4Розробка документації

Для підготовки графічної частини технічної документації використано середовище Fritzing, яке дозволяє створювати монтажні схеми підключення та принципові електричні схеми.

На вкладці «Макетна плата» розташовуються компоненти і виконуються з’єднання.

На вкладці «Схема» відображаються принципові електричні елементи, де можна впорядкувати підключення, які переносяться зі вкладки «Макетна плата».

Схема принципова електрична підключень компонентів платформи наведена на листі А26.14.АРТ.01.СБ.

Схема принципова електрична з’єднань компонентів платформи наведена на листі А26.14.АРТ.02.СБ.

Специфікація устаткування наведена на листі А26.14.АРТ.00.СО1.

4 ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

Програмне забезпечення є центральною частиною системи керування роботизованою рукою, оскільки саме воно поєднує апаратні компоненти в єдиний функціональний пристрій. Мікроконтролер приймає сигнали від джойстиків, обробляє команди з послідовного інтерфейсу, керує положеннями сервоприводів, виконує програмні сценарії руху та забезпечує збереження користувацьких програм у пам’яті.

Під час розроблення програмної частини було враховано кілька основних вимог. По-перше, система повинна забезпечувати безпечне ручне керування всіма віссями маніпулятора з урахуванням механічних обмежень. По-друге, керування має бути доступним не тільки з пульта, а й дистанційно через USB або Bluetooth-з’єднання. По-третє, програмне забезпечення повинно підтримувати відтворення окремих положень, завантаження послідовностей руху, повторення кадрів, запуск програм у різних режимах та збереження підготовлених сценаріїв для подальшого автономного використання.

У розділі послідовно розглянуто основні режими роботи програмного забезпечення. Спочатку описано безпосереднє ручне керування маніпулятором за допомогою двох аналогових джойстиків. Далі наведено принцип дистанційного керування через текстові команди STATE та IMMEDIATE. Після цього розглянуто режим числового програмного управління, у якому роботизована рука виконує завантажені послідовності кадрів. Завершальна частина розділу присвячена збереженню програм в EEPROM, роботі з іменами програм і використанню вбудованих сценаріїв руху.

4.1Безпосереднє ручне керування

Безпосереднє ручне керування є базовим режимом роботи роботизованої руки. У цьому режимі оператор змінює положення ланок за допомогою двох аналогових джойстиків, підключених до входів мікроконтролера. Аналогові осі джойстиків використовуються для керування сервоприводами, а кнопки надалі застосовуються для запуску програмного режиму.

У межах цього підрозділу розглядаються лише ті програмні елементи, які забезпечують безпосереднє ручне керування. Функції збереження програм, їх відтворення, Bluetooth-протокол і програмний запуск розглядаються в наступних підрозділах.

Ручне керування реалізовано в основному файлі DP.ino (Додаток А) і модулі ArmServo.h (Додаток Б). Основний файл виконує ініціалізацію системи, зчитування значень джойстиків і виклик методів зміни положення сервоприводів. Модуль ArmServo.h містить класи:

- для роботи з окремим сервоприводом;
- роботизованою рукою як сукупністю чотирьох вісей.

Таблиця 4.1 – Основні програмні модулі керування маніпулятором

Програмний елемент	**Модуль**	**Призначення**
DP.ino	DP.ino	Основний файл програми. Забезпечує ініціалізацію, опитування джойстиків і виклик команд зміни положень сервоприводів.
ArmServo	ArmServo.h	Клас окремого сервоприводу. Зберігає поточний кут, допустимі межі та виконує безпечну зміну положення.
RoboArm	ArmServo.h	Клас, що об’єднує чотири осі маніпулятора в один об’єкт для зручного звернення до всієї руки.

Основним елементом для роботи з окремою віссю є клас ArmServo. Його призначення полягає не в прямому формуванні всіх сигналів керування, а в організації безпечної зміни кута сервоприводу, основні властивості вказані у таблиці 4.2.

Основна логіка ArmServo полягає в тому, що зміна положення сервоприводу виконується не напряму, а через перевірку допустимого діапазону. Завдяки цьому ручне керування не може вивести механізм за межі безпечного ходу.

Таблиця 4.2 – Основні властивості класу керування маніпулятором ArmServo

Атрибут/метод	Призначення
angle	Поточне положення сервопривода в градусах. Це значення змінюється під час ручного керування.
minAngle, maxAngle	Межі допустимого руху осі. Вони не дозволяють передати на сервопривод небезпечне значення кута.
defaultAngle	Початкове положення осі після запуску системи.
write()	Встановлює нове положення сервопривода з урахуванням мінімального та максимального кута.
addAngle()	Змінює поточний кут на задану величину під час обробки відхилення джойстика.
writeDefault()	Переводить сервопривод у початкове положення.
getAngle()	Повертає поточний кут сервопривода для контролю стану осі.

Під час ручного керування метод addAngle() використовується для поступової зміни положення осі залежно від напряму відхилення джойстика.

Клас RoboArm (табл. 4.3) об’єднує чотири об’єкти ArmServo в одну структуру. Це спрощує звернення до маніпулятора, оскільки в основному файлі програми можна працювати з єдиним об’єктом руки та його окремими осями.

У межах прямого ручного керування RoboArm використовується як зручне об’єднання всіх виконавчих осей. Безпосередня зміна положень виконується через відповідні об’єкти ArmServo, а RoboArm забезпечує загальну структуру для роботи з маніпулятором як з єдиною системою.

Таблиця 4.3 – Основні властивості класу керування маніпулятором RoboArm

Атрибут/метод	Призначення
Rotate	Вісь повороту основи маніпулятора, Servo1
Elevation	Вісь піднімання та опускання лівої ланки, Servo2
Stretch	Вісь висування та втягування правої ланки, Servo3

Атрибут/метод	Призначення
Claw	Вісь відкриття та закриття захвату, Servo4
writeDefaults()	Встановлює всі осі маніпулятора в початкові положення.
printState()	Формує текстове подання поточних положень усіх вісей.

Основні налаштування прямого ручного керування задаються у основному модулю програми DP.ino у вигляді констант. Такий підхід спрощує зміну підключення, меж руху та параметрів реакції на джойстики без перероблення загальної логіки програми.

Таблиця 4.4 – Основні налаштування програми

Група параметрів	Призначення
Піни сервоприводів	Визначають підключення осей Rotate, Elevation, Stretch і Claw до плати спряження.
Піни джойстиків	Визначають входи для осей X/Y та кнопок правого і лівого джойстика.
Початкові кути	Задають положення маніпулятора після ввімкнення.
Мінімальні та максимальні кути	Обмежують рух сервоприводів відповідно до механічної конструкції.
Нейтральна зона джойстика	Визначає діапазон, у якому випадкові коливання сигналу не викликають руху.
Крок зміни кута	Визначає, на скільки градусів змінюється положення сервопривода за один цикл керування.

Під час роботи основний цикл програми постійно зчитує значення з аналогових входів джойстиків. Перед початком роботи виконується калібрування середнього положення джойстиків, тому подальше керування здійснюється не відносно фіксованого значення, а відносно фактично виміряного центру кожної осі.

Якщо відхилення рукоятки перебуває в межах нейтральної зони, відповідний сервопривід залишається нерухомим. Якщо джойстик відхилено за межі цієї зони, програма визначає напрям руху та швидкість переміщення. Швидкість залежить від величини відхилення: за малого відхилення

сервопривід рухається повільно, за середнього – швидше, а за повного відхилення – з найбільшою швидкістю. Такий підхід дає оператору змогу не тільки задавати напрям руху, а й плавно керувати інтенсивністю переміщення ланок.

Загальний алгоритм прямого ручного керування має такий вигляд:

1. ініціалізація сервоприводів і встановлення початкових положень;
2. калібрування середнього положення джойстиків;
3. зчитування значень осей джойстиків;
4. визначення відхилення від відкаліброваного центру;
5. перевірка нейтральної зони;
6. вибір швидкості залежно від величини відхилення;
7. визначення напряму руху для кожної осі;
8. зміна кута відповідного сервопривода з урахуванням вибраної швидкості;
9. перевірка мінімальних і максимальних меж;
10. передавання нового положення на сервопривод.

Для реалізації такого керування в програмі використовується функція визначення швидкості за відхиленням джойстика. Якщо відхилення менше заданої нейтральної зони, рух не виконується. Якщо відхилення перевищує нейтральну зону, але залишається невеликим, обирається повільний режим. При більшому відхиленні використовується середня або висока швидкість. Безпосереднє керування сервоприводом виконується через функцію обробки вісі джойстика, яка визначає напрям руху та передає відповідну команду збільшення або зменшення кута.

Програмні обмеження кутів є важливою частиною ручного керування. Вони запобігають виходу сервоприводів за допустимі механічні межі, зменшують ризик заклинювання ланок і захищають проводи від натягування під час руху. Наприклад, для сервопривода захвату встановлюється мінімальний безпечний кут, нижче якого програма не дозволяє опускати значення.

4.2 Дистанційне керування шляхом відтворення положень маніпулятора

Після реалізації безпосереднього ручного керування було додано можливість дистанційної взаємодії з маніпулятором. У цьому режимі оператор керує системою не через пульт, а шляхом передавання текстових команд через послідовний інтерфейс UART. Фізично такий обмін може виконуватися через USB-з'єднання з комп'ютером або через Bluetooth-модуль HC-05, який передає отримані зі смартфона чи термінальної програми символи на апаратні лінії RX/TX мікроконтролера.

Дистанційне керування побудовано на простому текстовому протоколі. Такий підхід обрано через зручність налагодження: команди можна вводити вручну з термінала, одразу бачити відповідь контролера та перевіряти роботу окремих функцій без створення окремого застосунку. У межах цього підрозділу розглядаються команди:

- STATE – отримання поточного положення маніпулятора;
- IMMEDIATE – негайне відтворення заданого положення.

Розширені команди завантаження, збереження та циклічного виконання програм розглядаються в наступних підрозділах.

Основна ідея дистанційного керування полягає в тому, що зовнішній пристрій передає в контролер команду у текстовому форматі, а програма розпізнає її, перетворює на внутрішню структуру кадру та виконує переміщення сервоприводів. При цьому безпосереднє ручне керування тимчасово блокується, щоб команди з джойстиків не конфліктували з виконанням отриманого положення.

Таблиця 4.5 – Основні програмні елементи дистанційного керування

Програмний елемент	**Модуль**	**Призначення**
BluetoothUploadParser	BluetoothUploadParser.h	Приймає символи з послідовного порту, розпізнає текстові команди STATE та IMMEDIATE і формує запит на виконання відповідної дії.

RoboActionFrame	Robo.h	Описує один кадр положення маніпулятора, тобто набір цільових положень осей і затримку після виконання.
RoboAxisDestination	Robo.h	Описує цільове положення окремої осі в кадрі, зокрема кут, швидкість переміщення та ознаку активності осі.

Обмін даними виконується через об’єкт Serial. У головному циклі програми всі доступні символи з послідовного порту передаються до BluetoothUploadParser. Парсер накопичує отримані символи, розпізнає завершену команду та встановлює службовий запит. Після цього основна програма перевіряє тип запиту і виконує відповідну дію: виведення стану або запуск негайного переміщення.

Класи ArmServo та RoboArm у цьому підрозділі не розглядаються повторно, оскільки їх загальне призначення вже описано раніше. Для дистанційного керування важливо лише те, що окрема вісь може бути переміщена до заданого кута методом moveTo(). На відміну від ручного режиму, де кут змінюється залежно від відхилення джойстика, у режимі IMMEDIATE цільове положення задається текстовою командою.

Для отримання поточного положення маніпулятора використовується команда STATE. Вона не змінює положення сервоприводів, а лише виводить у послідовний порт значення поточних кутів усіх осей. Відповідь формується у тому самому форматі, який надалі може бути використаний для команди негайного встановлення положення.

Формат виведення поточного стану:

```
r:<кут>;
c:<кут>;
e:<кут>;
s:<кут>;
```

де r – вісь повороту основи Rotate, c – захват Claw, e – вісь піднімання Elevation, s – вісь висування Stretch.

Додатково програма формує рядок у форматі команди IMMEDIATE. Це зручно під час налагодження: оператор може вручну встановити потрібне положення маніпулятора джойстиками, виконати команду STATE, скопіювати сформований рядок IMMEDIATE і надалі використовувати його як готовий кадр положення.

Команда IMMEDIATE призначена для негайного переходу маніпулятора до заданого положення. Вона приймає набір осей і цільових кутів у текстовому форматі. Наприклад:

```
IMMEDIATE r:90; c:19; e:130; s:60; delay:0;
```

У цьому прикладі для осі Rotate встановлюється кут 90°, для Claw – 19°, для Elevation – 130°, для Stretch – 60°. Параметр delay задає затримку після завершення переміщення. Якщо в команді вказано не всі осі, змінюються лише ті положення, які були передані в команді. Інші вісі залишаються в поточному положенні.

Таблиця 4.6 – Позначення вісі у дистанційних командах

Позначення	Вісь маніпулятора	Призначення
r	Rotate	Поворот основи маніпулятора.
c	Claw	Відкриття та закриття захвату.
e	Elevation	Піднімання та опускання лівої ланки.
s	Stretch	Висування та втягування правої ланки.
delay	Затримка	Пауза після завершення переміщення.

Для внутрішнього подання команди IMMEDIATE використовується структура кадру RoboActionFrame. Кадр містить не просто чотири кути, а набір цільових положень, кожне з яких може бути активним або неактивним. Це дозволяє змінювати як положення всієї руки, так і часткову команду, наприклад тільки відкриття захвату або тільки поворот основи.

Таблиця 4.7 – Основні властивості структури RoboActionFrame

Властивість	Призначення
Rotate	Цільове положення осі повороту основи.
Claw	Цільове положення захвату.
Elevation	Цільове положення осі піднімання.
Stretch	Цільове положення осі висування.
delay	Затримка після виконання кадру.

Кожна вісь усередині кадру описується окремою структурою RoboAxisDestination. Її основне завдання – зберегти цільовий кут, ознаку активності осі та параметр швидкості виконання

Таблиця 4.8 – Основні властивості структури RoboAxisDestination

Властивість	Призначення
angle	Цільовий кут відповідної осі.
slowness	Затримка між кроками переміщення, яка визначає швидкість руху.
isEnabled()	Вказує, чи потрібно виконувати переміщення цієї осі в поточному кадрі.

Після приймання команди IMMEDIATE парсер формує кадр і передає його основній програмі. Основний файл зупиняє поточне програмне або ручне переміщення, зберігає отриманий кадр як активний і запускає його виконання. Поки кадр виконується, значення джойстиків не використовуються для керування осями.

Виконання кадру відбувається поступово. Для кожної активної осі програма перевіряє, чи досягнуто цільового кута. Якщо ні, відповідний сервопривід робить один крок у напрямку цільового значення. Після цього програма повертається в основний цикл і продовжує переміщення на наступних ітераціях. Такий підхід не блокує роботу контролера довгими затримками та дозволяє паралельно обробляти інші службові команди.

Загальний алгоритм виконання команди IMMEDIATE має такий вигляд:

1. приймання символів команди через UART або Bluetooth;
2. передавання символів у BluetoothUploadParser;

3. розпізнавання команди IMMEDIATE;
4. формування внутрішнього кадру RoboActionFrame;
5. зупинка поточного ручного або програмного руху;
6. блокування керування від джойстиків на час виконання кадру;
7. поступове переміщення активних осей до заданих кутів;
8. виконання затримки delay після досягнення положення;
9. повернення системи до режиму очікування команд.

Важливою особливістю такого підходу є те, що команда IMMEDIATE працює не як різке встановлення сервоприводів у нове положення, а як кероване переміщення до заданих кутів. Швидкість руху визначається параметром slowness. Чим менше значення затримки між кроками, тим швидше сервопривід досягає заданого положення. Якщо параметр швидкості не вказано, використовується стандартне значення, задане у програмі.

Команда STATE доповнює IMMEDIATE і використовується як засіб контролю та підготовки кадрів. За її допомогою можна отримати фактичне положення маніпулятора, перевірити правильність роботи сервоприводів і сформувати команду для повторного відтворення поточного положення. Таким чином, оператор може спочатку виставити положення з пульту, потім отримати її координати через STATE, а після цього відтворити це положення дистанційно через IMMEDIATE.

Дистанційне керування шляхом відтворення положень маніпулятора є проміжним етапом між ручним керуванням і повноцінним числовим програмним управлінням. На цьому рівні система вже приймає команди ззовні, формує внутрішні кадри руху та відтворює задані положення без участі джойстиків. Надалі ця ж логіка використовується для виконання послідовностей кадрів, повторень, збереження програм у пам’яті та запуску автоматичних сценаріїв роботи.

Код модулів наведено в додатках Robo.h у додатку В, BluetoothUploadParser.h у додатку Г.

4.3Дистанційне керування за допомогою ЧПУ

Після реалізації дистанційного відтворення окремого положення програмне забезпечення було розширено до режиму числового програмного управління. Якщо команда IMMEDIATE виконує тільки один кадр, то режим числового програмного управління дозволяє завантажити в контролер повну програму руху. Така програма складається з послідовностей, а кожна послідовність містить один або кілька кадрів.

У межах цього підрозділу розглядаються команди, які пов’язані із завантаженням програми в оперативний буфер, її переглядом, очищенням і запуском на виконання. Команди збереження програми в EEPROM та завантаження з пам’яті розглядаються в наступному підрозділі.

Основними командами цього етапу є:

- START UPLOAD – початок передавання тексту програми;
- END UPLOAD – завершення передавання програми;
- PRINT – виведення текстового подання поточної програми;
- INFO – виведення службової інформації про стан програми;
- CLEAR – очищення буфера програми;
- ONCE – одноразове виконання завантаженої програми;
- LOOP – циклічне виконання завантаженої програми;
- STOP – зупинка поточного виконання.

На цьому етапі раніше описані структури RoboActionFrame та RoboAxisDestination використовуються як основа для окремого кадру. Новими програмними елементами є структури, які об’єднують кадри в послідовності, зберігають програму в буфері та виконують її відтворення.

Кадр залишається мінімальною командою руху. Він може містити положення однієї або кількох осей, параметр уповільнення та затримку після завершення руху. У програмі підтримується як багаторядковий, так і однорядковий формат запису кадру.

Таблиця 4.9 – Програмні елементи числового програмного управління

Програмний елемент	Модуль	Призначення
RoboSequence	Robo.h	Описує одну послідовність кадрів і кількість її повторень.
RoboProgramBuffer	Robo.h	Зберігає завантажену програму як набір послідовностей і спільний буфер кадрів.
Playback	Playback.h	Виконує відтворення завантаженої програми в режимах ONCE або LOOP.
BluetoothUploadParser	BluetoothUploadParser.h	Приймає текст програми, розпізнає службові команди та заповнює буфер програми.

Багаторядковий формат зручний для ручного редагування складніших програм:

```
r:90/10;
c:19/5;
e:130/10;
s:60/10;
delay:1;
```

Однорядковий формат зручний для швидкого введення в терміналі або для копіювання положень, отриманих після команди STATE:

```
r:90/10; c:19/5; e:130/10; s:60/10; delay:1;
```

Після позначення осі може вказуватися параметр уповільнення у форматі r:<кут>/<уповільнення>. Кут задає цільове положення сервопривода, а уповільнення визначає затримку між окремими кроками переміщення. Якщо уповільнення не вказане, використовується стандартне значення. Менше значення уповільнення відповідає швидшому переміщенню, більше значення – повільнішому.

У тексті програми також підтримуються коментарі. Коментар починається символами // і використовується тільки для пояснення дій у програмі. Він не перетворюється на кадр і не впливає на рух маніпулятора.

```
// захоплення об’єкта
c:19/5; delay:0.5;

// підняття руки
e:120/10; s:80/10; delay:1;
```

Для групування кадрів використовується поняття послідовності. Послідовність – це набір кадрів, які виконуються один за одним. Якщо для послідовності задано повторення, вона виконується кілька разів. Для цього застосовується конструкція repeat.

Формат послідовності з повторенням:

```
repeat:3;
c:110/5; delay:0.5;
c:19/5; delay:0.5;
endrepeat;
```

У цьому прикладі створюється послідовність із двох кадрів. Вона буде виконана три рази. Це дозволяє описувати повторювані дії без дублювання однакових кадрів у тексті програми.

Таблиця 4.10 – Основні властивості та методи RoboSequence

Атрибут/метод	Призначення
frameCount	Зберігає кількість кадрів у послідовності.
repeats	Задає кількість повторень послідовності.
start()	Починає формування нової послідовності.
addFrame()	Додає кадр до поточної послідовності.
getFrame()	Повертає кадр за його номером під час відтворення.

Завантажена програма зберігається в об’єкті RoboProgramBuffer. Він містить масив послідовностей і спільний буфер кадрів. Такий підхід дозволяє зберігати кадри компактно: самі кадри розміщуються в одному загальному буфері, а послідовності визначають, які саме кадри до них належать і скільки разів їх потрібно повторити.

Таблиця 4.11 – Команди завантаження та керування програмою

Команда	Призначення
START UPLOAD	Переводить систему в режим приймання тексту програми.
END UPLOAD	Завершує приймання програми та формує її внутрішнє подання.
PRINT	Виводить текстове подання поточної завантаженої програми.
INFO	Виводить службову інформацію про кількість послідовностей, кадрів і стан буфера.
CLEAR	Очищує завантажену програму з оперативного буфера.
ONCE	Запускає одноразове виконання програми.
LOOP	Запускає циклічне виконання програми.
STOP	Зупиняє поточне виконання програми.

Передавання програми починається командою START UPLOAD. Після її отримання контролер переходить у режим приймання тексту програми. У цьому режимі рядки з координатами, delay, repeat, endrepeat та коментарями сприймаються як частини програми руху. Завершення передавання виконується командою END UPLOAD.

Приклад повного завантаження програми з кількома послідовностями:

```
START UPLOAD;

// --- s01: підготовка руки ---
repeat:1;
r:90/10; c:110/5; e:130/10; s:60/10; delay:1;
endrepeat;

// --- s02: повторне відкриття та закриття захвату ---
repeat:3;
c:110/5; delay:0.5;
c:19/5; delay:0.5;
endrepeat;

// --- s03: переміщення в інше положення ---
repeat:1;
r:120/10; e:110/10; s:80/10; delay:1;
c:110/5; delay:0.5;
endrepeat;

END UPLOAD;
```

У цьому прикладі програма складається з трьох послідовностей. Перша послідовність переводить маніпулятор у початкове положення, друга тричі відкриває та закриває захват, а третя переміщує руку в інше положення і

відкриває захват. Коментарі після символів // використовуються лише для пояснення структури програми і не впливають на виконання рухів.

Після команди END UPLOAD програма завершує формування буфера. Якщо послідовність була відкрита, вона має бути коректно завершена. Якщо буфер кадрів або позслідовностей переповнений, некоректні дані не повинні використовуватися для виконання руху.

Команда PRINT використовується для перевірки текстового подання поточної завантаженої програми. Вона дозволяє переглянути програму у зрозумілому для оператора вигляді та переконатися, що кадри, послідовності, повторення і затримки були прийняті правильно.

Команда INFO виводить службову інформацію про стан програми, зокрема кількість послідовностей, загальну кількість кадрів і використання буфера. Команда CLEAR очищує поточний буфер програми і використовується перед завантаженням нової програми або після помилкового введення.

Команда ONCE запускає одноразове виконання програми. У цьому режимі всі послідовності виконуються по черзі. Якщо всередині послідовності вказано repeat, повторення виконується лише в межах цієї послідовності. Після завершення останньої послідовності система повертається до режиму очікування команд.

Команда LOOP запускає циклічне виконання всієї програми. Після завершення останньої послідовності програвач повертається до першої послідовності та починає виконання знову. Такий режим зручний для демонстраційних сценаріїв або багаторазового відпрацювання однакової траєкторії.

Команда STOP припиняє поточне виконання. Вона може бути передана через UART або Bluetooth під час роботи програми. Після зупинки система повертається до режиму очікування, а оператор може знову перейти до ручного керування або запустити іншу команду.

Відтворення завантаженої програми виконує клас Playback. Він не розбирає текстові команди, а працює з уже підготовленим буфером RoboProgramBuffer. Його завдання – послідовно проходити по послідовностях, кадрах і повтореннях, передавати цільові положення сервоприводам та контролювати затримки після кадрів.

Таблиця 4.12 – Основні методи класу Playback

Атрибут/метод	**Призначення**
begin()	Підключає до програвача буфер програми та об’єкт маніпулятора.
startOnce()	Запускає одноразове виконання програми.
startLoop()	Запускає циклічне виконання програми.
stop()	Зупиняє поточне відтворення.
isActive()	Показує, чи виконується програма в цей момент.
update()	Виконує черговий крок відтворення без блокування основного циклу.

Крім дистанційних команд, завантажена програма може запускатися з пульта оператора. Для цього використовуються кнопки обох джойстиків. Одночасне одноразове натискання двох кнопок запускає одноразове виконання програми, тобто є аналогом команди ONCE. Подвійне одночасне натискання запускає циклічне виконання програми, тобто є аналогом команди LOOP. Якщо програма вже виконується, натискання кнопок може використовуватися для її зупинки, що відповідає дії команди STOP.

Під час виконання програми ручне керування осями блокується. Це важливо для безпеки та стабільності руху: оператор може натискати кнопки пульта для запуску або зупинки сценарію, але випадкове відхилення рукояток джойстиків у цей момент не змінює положення сервоприводів. Завдяки цьому програмна траєкторія не порушується, а маніпулятор виконує саме ті кадри, які були завантажені в буфер.

Такий підхід дозволяє завантажити програму через Bluetooth або USB-термінал, а потім запускати її без підключення до зовнішнього пристрою. Пульт у цьому випадку використовується не для безпосереднього

переміщення осей, а як простий орган керування режимами виконання програми.

Загальний алгоритм роботи числового програмного управління має такий вигляд:

1. приймання команди START UPLOAD;
2. підготовка буфера програми;
3. приймання кадрів, коментарів і repeat-блоків;
4. формування послідовностей RoboSequence;
5. запис кадрів у RoboProgramBuffer;
6. завершення завантаження командою END UPLOAD;
7. перевірка програми командами PRINT та INFO;
8. запуск програми командою ONCE, LOOP або з пульта;
9. покрокове виконання кадрів через Playback;
10. зупинка після завершення програми або за командою STOP.

У режимі числового програмного управління маніпулятор переходить від виконання одиничних дистанційних команд до відтворення повноцінних сценаріїв руху. Програма може містити окремі кадри, однорядкові записи положень, коментарі та повторювані послідовності. Це робить систему придатною для автоматизованого виконання простих технологічних або демонстраційних операцій без постійного ручного керування оператором.

Код модулю Playback наведений у Додаток Д.

4.4 Збереження програм та вбудовані програми

У попередньому підрозділі було розглянуто завантаження програми руху в оперативний буфер, її перегляд, очищення та запуск на виконання. Такий підхід зручний для налагодження, але після вимкнення живлення програма, що зберігається тільки в оперативній пам’яті, втрачається. Для усунення цього обмеження в програмному забезпеченні передбачено збереження програми в енергонезалежну пам’ять EEPROM, а також завантаження вбудованих програм, записаних безпосередньо в пам’ять програм мікроконтролера.

У межах цього підрозділу розглядаються команди збереження, завантаження та роботи з вбудованими програмами. Команди START UPLOAD, END UPLOAD, PRINT, INFO, CLEAR, ONCE, LOOP і STOP повторно детально не розглядаються, оскільки їх призначення описано в підрозділі 4.3. Тут вони враховуються лише як частина загального процесу: програма спочатку завантажується в буфер, після цього може бути збережена в EEPROM або замінена програмою з EEPROM чи з модуля DefaultProg.h.

Розглянуті команди:

- SAVE – збереження поточної програми в EEPROM;
- LOAD – завантаження програми з EEPROM;
- BUILTIN – виведення списку вбудованих програм;
- DEFLOAD – завантаження вбудованої програми за номером.

Таблиця 4.13 – Програмні елементи збереження та завантаження програм

Програмний елемент	**Модуль**	**Призначення**
RoboProgramManager	RoboProgramManager.h	Керує збереженням, завантаженням, іменем програми та роботою з EEPROM і вбудованими програмами.
RoboEepromReader	Robo.h	Забезпечує послідовне читання байтів з EEPROM.

Програмний елемент	Модуль	Призначення
RoboProgmemReader	Robo.h	Забезпечує послідовне читання байтів із пам’яті програм.
DefaultProg.h	DefaultProg.h	Містить вбудовані програми та їхні імена.
RoboVersion.h	RoboVersion.h	Містить descriptor і номер версії формату програми.

Основним елементом цього підрозділу є RoboProgramManager. Він не відповідає за ручне керування або покрокове відтворення кадрів. Його завдання полягає в тому, щоб працювати з готовою програмою як з цілісним об’єктом: зберегти її, завантажити, призначити ім’я, перевірити службовий опис і надати інформацію про поточну програму.

Таблиця 4.14 – Основні методи RoboProgramManager

Атрибут/метод	Призначення
save()	Зберігає поточну програму з оперативного буфера в EEPROM.
load()	Завантажує програму з EEPROM у робочий буфер.
loadBuiltin()	Завантажує вбудовану програму за її номером.
printBuiltinList()	Виводить список доступних вбудованих програм.
getProgramName()	Повертає назву поточної програми.
setUploadedName()	Задає автоматичну назву після завантаження через UPLOAD.
clearProgramName()	Скидає назву поточної програми.

Команда SAVE використовується після того, як програма вже була завантажена в оперативний буфер через START UPLOAD та END UPLOAD. Вона записує поточну програму в EEPROM, щоб після вимкнення живлення її можна було відновити командою LOAD або використати під час запуску з пульта.

Формат команди:

```
SAVE <назва>
```

Якщо після SAVE передано назву, вона зберігається разом із програмою. Якщо назва не вказана, формується автоматична назва виду Unnamed.<число>,

де число утворюється на основі поточного часу роботи контролера. Максимальна довжина назви становить 25 символів.

Команда LOAD завантажує програму з EEPROM у робочий буфер:

```
LOAD
```

Після успішного завантаження програма може бути запущена командами ONCE або LOOP, а також з пульта оператора кнопками джойстиків. Під час виконання команди INFO програма виводить ім’я поточної програми, тому оператор може перевірити, яка саме програма завантажена.

Команда BUILTIN виводить список вбудованих програм:

```
BUILTIN
```

У відповідь контролер виводить номери програм, їхні імена та версію формату. У модулі DefaultProg.h передбачено три вбудовані програми: Default pickup, Claw sweep і Rotate sweep. Їхні імена зберігаються в тому самому форматі, що й ім’я користувацької програми.

Команда DEFLOAD завантажує одну з вбудованих програм за номером:

```
DEFLOAD <номер>
```

Наприклад, команда DEFLOAD 1 завантажує першу вбудовану програму. Після цього вона потрапляє в той самий робочий буфер, що й програма, прийнята через START UPLOAD. Надалі її можна запускати, переглядати, зберігати в EEPROM або замінювати іншою програмою.

Після завантаження програми через START UPLOAD та END UPLOAD їй автоматично призначається ім’я виду Uploaded.<число>. Це ім’я дозволяє відрізняти програму, щойно прийняту через Bluetooth або USB-термінал, від програми, завантаженої з EEPROM чи з модуля DefaultProg.h. Після команди CLEAR ім’я програми скидається до значення Unnamed.

Для зберігання програми використовується компактний двійковий формат. Текст програми не записується в EEPROM у вихідному вигляді,

оскільки такий запис займав би надто багато пам’яті. Наприклад, текстовий рядок з кількома осями, параметрами швидкості та затримкою містить десятки символів, тоді як у внутрішньому форматі зберігаються тільки числові значення та службові прапори.

Формат збереження програми складається з двох частин: заголовка програми та двійкових даних програми. Заголовок містить службовий descriptor, ім’я програми та дані для перевірки сумісності. Descriptor має вигляд RoboArmXX.YY, де XX – основна версія формату, а YY – молодша версія. У поточній версії використовується descriptor RoboArm01.02.

Таблиця 4.15 – Формат заголовка програми

Поле	Розмір	Призначення
descriptor	12 байтів	Службовий підпис і версія формату програми.
nameLength	1 байт	Довжина імені програми.
nameBuffer	25 байтів	Фіксоване поле для імені програми.
Разом	38 байтів	Повний розмір заголовка.

Ім’я програми зберігається у фіксованому полі довжиною 25 байтів. Окремий байт nameLength показує, скільки символів імені фактично використано. Якщо ім’я коротше за 25 символів, решта поля заповнюється нулями. Такий формат спрощує читання заголовка і дозволяє швидко перейти до двійкових даних програми.

Після заголовка зберігається сама програма. Її структура відповідає внутрішньому формату RoboProgramBuffer: спочатку записується кількість послідовностей, далі для кожної послідовності записуються repeats і frameCount, а після цього зберігаються кадри цієї послідовності.

Компактність формату забезпечується використанням прапорів активних осей. У кожному кадрі зберігається байт flags, де для кожної вісі виділено два біти:

- один біт показує наявність кута;

- другий – наявність параметра slowness.

Якщо певна вісь у кадрі не використовується, її кут і slowness взагалі не записуються.

Порядок осей у двійковому кадрі такий: Stretch, Elevation, Claw, Rotate. Структура одного кадру містить flags, значення кутів тільки для активних осей, значення slowness тільки для тих осей, де воно відрізняється від стандартного, і delay10 – затримку після кадру в десятих частках секунди.

Таблиця 4.16 – Оцінка розміру одного кадру

Варіант кадру	Склад даних	Розмір
Мінімальний кадр	flags, один кут, delay10	3 байти
Кадр з кількома осями без нестандартної швидкості	flags, кути активних осей, delay10	3–6 байтів
Повний кадр з усіма осями та slowness	flags, 4 кути, 4 значення slowness, delay10	10 байтів

Найбільший можливий кадр займає 10 байтів. Це відбувається тоді, коли в одному кадрі задано всі чотири вісі і для кожної вісі вказано власне значення slowness. У більшості практичних кадрів розмір буде меншим, оскільки не всі вісі змінюються одночасно і не завжди потрібно задавати нестандартну швидкість.

Кожна послідовність додатково займає 2 байти службової інформації: repeats – кількість повторень і frameCount – кількість кадрів у послідовності. Для збереження кількості послідовностей у програмі використовується ще 1 байт.

Тому максимальний розрахунковий розмір двійкових даних програми становить:

```
40 кадрів × 10 байтів = 400 байтів
25 послідовностей × 2 байти = 50 байтів
1 байт кількості послідовностей = 1 байт
400 + 50 + 1 = 451 байт
```

До цих 451 байтів додається заголовок розміром 38 байтів:

```
451 + 38 = 489 байтів
```

Програма зберігається в EEPROM, починаючи з адреси 512. Загальний обсяг EEPROM мікроконтролера становить 1 КБ, тобто доступний діапазон адрес становить 0–1023. Для користувацької програми виділено другу половину EEPROM – від адреси 512 до кінця пам’яті, тобто 512 байтів.

У найгіршому випадку програма займає до 489 байтів із доступних 512 байтів. Запас становить:

```
512 – 489 = 23 байти
```

Саме тому обмеження вибрано на рівні 40 кадрів і 25 послідовностей. Навіть якщо кожен кадр буде максимального розміру, програма разом із заголовком, іменем і службовими даними поміститься у виділену область EEPROM.

Таблиця 4.17 – Обґрунтування обмежень програми

Обмеження	Обґрунтування
40 кадрів	У найгіршому випадку займають до 400 байтів.
25 послідовностей	Дозволяють розбивати програму на логічні блоки і займають до 50 байтів службових даних.
512 байтів EEPROM	Виділена область для користувацької програми, починаючи з адреси 512.
38 байтів заголовка	Descriptor, довжина імені та фіксоване поле імені.
489 байтів максимум	Повна програма з 40 максимальними кадрами, 25 послідовностями та заголовком.
23 байти запасу	Резерв у межах виділеної області EEPROM.

Під час завантаження програми з EEPROM контролер перевіряє службовий descriptor, основну версію формату, ім’я програми та двійкові дані. Якщо дані несумісні з поточною прошивкою або пошкоджені, програма не використовується для запуску. Це потрібно для того, щоб маніпулятор не

виконував випадкові дані після зміни прошивки або після некоректного запису EEPROM.

Окремо реалізовано механізм вбудованих програм. Вони описані в модулі DefaultProg.h і зберігаються не в EEPROM, а в пам’яті програм мікроконтролера. Це означає, що вони не займають область EEPROM, призначену для користувацької програми, і не стираються під час виконання команд CLEAR або SAVE.

Формат вбудованої програми збігається з форматом програми в EEPROM: descriptor, nameLength, nameBuffer і programData. Завдяки цьому однаковий механізм читання може використовуватися як для EEPROM, так і для вбудованих програм. Для EEPROM використовується RoboEepromReader, а для пам’яті програм – RoboProgmemReader.

Вбудовані програми потрібні для таких випадків:

- швидка перевірка працездатності системи після прошивки;
- відновлення базового сценарію роботи;
- запуск демонстраційної програми без попереднього завантаження через Bluetooth;
- резервний варіант, якщо в EEPROM немає коректної програми.

Під час запуску програми з пульта діє окремий порядок вибору джерела програми. Якщо оператор натискає дві кнопки джойстиків для запуску ONCE або LOOP, система спочатку перевіряє, чи є програма в оперативному буфері. Якщо така програма вже завантажена, використовується саме вона. Якщо буфер порожній, програма намагається завантажити дані з EEPROM. Якщо в EEPROM немає коректної програми або вона не проходить перевірку, система завантажує резервну вбудовану програму. Номер резервної вбудованої програми задається константою FALLBACK_BUILTIN_PROGRAM.

Пріоритет запуску з пульта має такий вигляд:

1. програма, вже завантажена в оперативний буфер;
2. програма, збережена в EEPROM;
3. резервна вбудована програма з DefaultProg.h.

Такий порядок роботи зручний для практичного використання. Після завантаження нової програми через Bluetooth оператор може одразу запускати її з пульта. Після вимкнення і повторного ввімкнення живлення система може відновити програму з EEPROM. Якщо збереженої програми немає, залишається можливість виконати вбудований сценарій.

При цьому спосіб запуску залишається тим самим, що було описано в підрозділі 4.3.

Загальний алгоритм роботи збереження та вбудованих програм має такий вигляд:

1. завантаження програми через START UPLOAD та END UPLOAD;
2. перевірка структури програми командами PRINT та INFO;
3. збереження програми в EEPROM командою SAVE;
4. завантаження збереженої програми командою LOAD;
5. перегляд списку вбудованих програм командою BUILTIN;
6. завантаження вбудованої програми командою DEFLOAD;
7. запуск вибраної програми командами ONCE, LOOP або кнопками пульта;
8. автоматичний вибір резервної програми під час запуску з пульта, якщо буфер і EEPROM порожні або некоректні.

Отже, підсистема збереження програм розширює числове програмне управління і робить його придатним для автономної роботи. EEPROM використовується для збереження користувацької програми з ім’ям, descriptor і службовими даними, а DefaultProg.h забезпечує наявність вбудованих сценаріїв. Компактний двійковий формат дозволяє розмістити до 40 кадрів і до 25 послідовностей у межах виділеної області пам’яті.

Код модулю RoboProgramManager.h наведений у Додаток Е.

Код модулю RoboVersion.h наведений у Додаток Ж.

Код модулю DefaultProg.h наведений у Додаток З.

5 ОХОРОНА ПРАЦІ І ТЕХНІКА БЕЗПЕКИ

Під час розроблення, складання та налагодження роботизованої руки необхідно враховувати ризики, пов’язані з роботою за персональним комп’ютером, використанням низьковольтних джерел живлення, підключенням літій-іонних акумуляторів, випробуванням сервоприводів і рухомих ланок маніпулятора. Незважаючи на те, що система працює з відносно невисокими напругами, порушення правил підключення може призвести до короткого замикання, перегріву провідників, пошкодження електронних модулів або травмування оператора рухомими частинами.

У розроблюваному виробі використовуються Arduino-сумісна плата керування, плата спряження для сервоприводів, чотири сервоприводи MG90S, два аналогові джойстики, Bluetooth-модуль HC-05, утримувач із двома елементами 18650 та з’єднувальні проводи DuPont. Тому вимоги безпеки мають охоплювати не тільки електронну частину, а й механічні вузли, які переміщуються під час роботи програми.

5.1 Загальні вимоги безпеки під час виконання робіт

До складання та налагодження роботизованої руки допускаються особи, які ознайомлені з будовою пристрою, призначенням основних модулів, правилами підключення живлення та порядком аварійного вимкнення системи. Перед початком роботи необхідно підготувати робоче місце, перевірити справність обладнання та переконатися, що всі дії виконуються без поспіху.

Небезпечними та шкідливими факторами під час виконання робіт є:

- ризик короткого замикання під час неправильного підключення живлення або сигнальних ліній;
- перегрів провідників, плати спряження, стабілізаторів або акумуляторів у разі перевантаження;

- механічне травмування пальців рухомими ланками та захватом маніпулятора;
- пошкодження очей або рук під час різкого руйнування дрібних деталей чи від’єднання кріпильних елементів;
- підвищене зорове навантаження та статична втома під час тривалої роботи за комп’ютером.

Робоча зона має бути чистою, сухою та достатньо освітленою. На столі не повинно бути зайвих металевих предметів, відкритих рідин, легкозаймистих матеріалів або сторонніх кабелів, які можуть заважати підключенню та перевірці пристрою. Доступ дітей і домашніх тварин до робочого місця має бути обмежений.

Під час роботи з макетними з’єднаннями забороняється торкатися оголених контактів металевими предметами, змінювати підключення під напругою та залишати пристрій увімкненим без нагляду. Для аварійного вимкнення необхідно мати швидкий доступ до вимикача живлення, USB-кабелю або роз’єму акумуляторного відсіку.

5.2 Вимоги безпеки під час роботи за комп’ютером

Програмування мікроконтролера, підготовка скетчів, перевірка команд через послідовний монітор і робота з Bluetooth-терміналом виконуються за персональним комп’ютером або ноутбуком. Під час такої роботи необхідно дотримуватися правил організації робочого місця користувача ПК.

Екран має бути розташований на відстані приблизно 50–70 см від очей. Освітлення робочої зони повинно бути рівномірним, без різких відблисків на моніторі. Під час тривалої роботи необхідно робити короткі перерви тривалістю 5 хвилин після кожних 45 хвилин безперервної роботи, щоб зменшити зорову та статичну втому.

Кабелі USB та живлення слід розміщувати так, щоб вони не натягувалися, не перегиналися і не потрапляли під рухомі частини маніпулятора. Перед завантаженням прошивки необхідно переконатися, що Bluetooth-модуль HC-05 від'єднаний від плати, оскільки програмування контролера використовує ті самі лінії TX/RX.

5.3 Вимоги безпеки під час складання електронної частини

Перед підключенням електронних модулів необхідно перевірити відповідність схеми з'єднань технічній документації. Особливу увагу слід приділяти правильності підключення ліній G, V і S на роз'ємах сервоприводів, джойстиків і плати спряження. Переплутування живлення та сигнального контакту може призвести до виходу з ладу мікроконтролера або периферійного модуля.

Усі монтажні роботи необхідно виконувати при вимкненому живленні. Підключення або переставляння проводів DuPont під напругою не допускається. Перед подаванням живлення треба перевірити, чи немає випадкового замикання між плюсовою шиною та землею, чи правильно вставлено сервоприводи в роз'єми, а також чи не торкаються оголені контакти металевих деталей конструкції.

Оскільки в роботизованій руці застосовуються готові модулі та з'єднувальні кабелі, основні роботи виконуються без паяння. Якщо під час ремонту або модернізації виникає потреба у паянні провідників, ці роботи треба виконувати тільки на відключеній платі, у провітрюваному приміщенні, з використанням термостійкої підставки для паяльника та захисних окулярів. Забороняється паяти безпосередньо на підключених акумуляторах або поблизу легкозаймистих матеріалів.

5.3 Вимоги безпеки під час роботи з літій-іонними акумуляторами

Живлення виконавчої частини роботизованої руки здійснюється від утримувача з двома літій-іонними елементами типорозміру 18650. Такі акумулятори мають високу енергоємність, тому в разі неправильного використання можуть бути джерелом пожежної небезпеки.

Перед установленням акумуляторів необхідно виконати їх візуальний огляд. До роботи не допускаються елементи зі здуттям, вм’ятинами, слідами іржі, пошкодженою ізоляційною оболонкою або витіканням електроліту. Також бажано перевірити напругу мультиметром. Не слід використовувати елементи, які мають ознаки глибокого розряду або нестабільної роботи.

Під час роботи з акумуляторами забороняється:

– замикати їхні контакти металевими предметами;

– паяти провідники безпосередньо до полюсів елементів 18650;

– нагрівати акумулятори паяльником, феном або іншими джерелами тепла;

– використовувати елементи з пошкодженою ізоляцією;

– залишати пристрій з підключеними акумуляторами без нагляду під час першого запуску.

Для підключення акумуляторів у цьому проєкті використовується готовий батарейний відсік, що зменшує ризик помилки монтажу. При встановленні елементів необхідно суворо дотримуватися полярності. Якщо після ввімкнення з’являється запах гару, дим, шипіння або різке нагрівання акумулятора, живлення треба негайно вимкнути, а акумуляторний відсік від’єднати від пристрою.

Пошкоджені або відпрацьовані літій-іонні акумулятори не можна викидати разом із побутовими відходами. Їх необхідно передавати до спеціалізованих пунктів приймання батарейок та акумуляторів.

5.5 Вимоги безпеки під час першого ввімкнення та налагодження

Перше ввімкнення зібраного пристрою є відповідальним етапом, оскільки саме в цей момент можуть проявитися приховані помилки монтажу: неправильна полярність, коротке замикання, переплутані сигнальні дроти або механічне заклинювання ланок. Перед першим запуском необхідно виконати візуальний контроль усіх з’єднань і переконатися, що маніпулятор має вільний простір для руху.

Перед подаванням живлення треба перевірити:

- правильність підключення сервоприводів Rotate, Elevation, Stretch і Claw до відповідних роз’ємів;
- правильність підключення джойстиків до аналогових і цифрових входів;
- відсутність натягнутих або защемлених проводів у зоні руху ланок;
- надійність кріплення сервоприводів, важелів, основи та захвату;
- відсутність сторонніх предметів у робочій зоні маніпулятора.

Перше ввімкнення бажано виконувати поетапно. Спочатку перевіряється робота контролера через USB без навантаження сервоприводів. Після цього підключається зовнішнє живлення плати спряження і окремо перевіряється реакція кожного сервопривода на тестові кути. Забороняється тримати пальці в зоні захвату або між рухомими ланками під час виконання тестового скетча.

Під час налагодження не можна живити всю систему тільки від USB-порту комп’ютера, оскільки сервоприводи споживають значний струм. Для роботи виконавчої частини необхідно використовувати зовнішнє джерело живлення через плату спряження. Недостатнє живлення може спричинити просідання напруги, перезавантаження контролера, некеровані рухи сервоприводів або пошкодження USB-порту.

5.6 Вимоги безпеки під час роботи рухомих частин маніпулятора

Роботизована рука має кілька рухомих ланок і механічний захват, тому під час її роботи існує ризик защемлення пальців або пошкодження дротів. Перед запуском ручного або програмного режиму треба переконатися, що в зоні руху немає рук, інструментів, дрібних деталей або сторонніх предметів.

Під час випробування програмних сценаріїв руху оператор повинен стежити за положенням ланок і бути готовим зупинити виконання програми командою STOP або вимкненням живлення. Не допускається примусово утримувати ланки руками під час роботи сервоприводів, оскільки це створює підвищене навантаження на редуктор і може призвести до перегріву або механічного пошкодження привода.

Особливу увагу слід приділяти сервоприводу захвату. Програмні обмеження кута мають запобігати надмірному стисканню механізму, але під час налагодження оператор не повинен розміщувати пальці між губками захвату. Якщо ланка або захват упирається в механічну перешкоду, необхідно негайно зупинити роботу та скоригувати кутові межі у програмі.

5.7 Дії в аварійних ситуаціях

У разі появи диму, запаху горілої ізоляції, іскріння, різкого нагрівання плати або акумуляторів необхідно негайно припинити роботу та відключити живлення. Після вимкнення не слід одразу торкатися елементів плати, оскільки окремі компоненти можуть залишатися гарячими.

Якщо виникло механічне заклинювання ланок маніпулятора, спочатку потрібно вимкнути живлення, а вже потім усувати перешкоду. Забороняється намагатися розтиснути або провернути ланки руками під час подавання керуючого сигналу на сервоприводи.

При термічному опіку уражену ділянку необхідно охолодити проточною прохолодною водою протягом 10–15 хвилин і, за потреби, звернутися по

медичну допомогу. При потраплянні дрібних частинок або електроліту в очі необхідно негайно промити їх великою кількістю чистої води та звернутися до лікаря.

У разі загоряння електронних компонентів або акумулятора необхідно відключити живлення, не вдихати дим і застосувати відповідний засіб пожежогасіння. Для локалізації загоряння літій-іонного акумулятора можна використовувати сухий пісок або порошковий вогнегасник. Використання води для гасіння акумулятора небажане, оскільки це може ускладнити ситуацію.

5.8 Вимоги безпеки після закінчення робіт

Після завершення складання, налагодження або випробування роботизованої руки необхідно вимкнути живлення, від’єднати USB-кабель і, якщо пристрій не буде використовуватися тривалий час, вийняти акумулятори з утримувача. Перед зберіганням слід переконатися, що сервоприводи не перебувають під навантаженням, а рухомі ланки не затискають проводи.

Робоче місце треба привести до ладу: прибрати інструменти, дроти, кріпильні елементи та відходи монтажу. Акумулятори слід зберігати окремо від металевих предметів, а їхні контакти не повинні мати можливості випадково замкнутися. Документацію, схеми підключення та програмні файли доцільно зберігати разом із проєктом, щоб під час подальшої модернізації не порушити правильність підключення.

Дотримання наведених вимог дозволяє зменшити ризики під час складання, програмування та експлуатації роботизованої руки, забезпечити безпечне налагодження електронних модулів і запобігти пошкодженню апаратної частини під час першого запуску та подальших випробувань.

6 ЕКОНОМІЧНІ РОЗРАХУНКИ

Економічні розрахунки виконуються для визначення орієнтовної собівартості розроблення та виготовлення дослідного зразка роботизованої руки. У розрахунках враховуються витрати на проєктування, програмування, складання, налагодження, закупівлю електронних і механічних компонентів, а також можливі накладні витрати.

6.1 Оцінка витрат на проєктування

Витрати на проєктування включають оплату інженерної праці, розроблення алгоритмів керування, підготовку програмного забезпечення, налагодження протоколу обміну та оформлення технічної документації.

6.1.1 Розрахунок трудомісткості та фонду оплати праці

Для кожного етапу проєктування задається витрачений час t_i та умовна годинна ставка R_{hi}. Вартість робіт за окремим етапом визначається за формулою:

$$Z_i = t_i \cdot R_{hi},$$

(6.1)

де Z_i – вартість робіт за і-м етапом, грн.

Таблиця 6.1 – Оплата інтелектуальної праці

№	Час t_i, год	Ставка *Rhi*, грн/год	Сума *Zi*, грн	Етап проєктування
1	8	600	4800	Аналіз завдання, уточнення режимів роботи та вимог до системи
2	12	700	8400	Розроблення структури програмного забезпечення та алгоритму ручного керування
3	35	900	31500	Написання основної прошивки, модулів керування сервоприводами та відтворення програм
4	12	800	9600	Налагодження Bluetooth-зв’язку, текстового протоколу та команд керування
5	10	800	8000	Перевірка збереження програм в EEPROM і вбудованих сценаріїв
6	20	500	10000	Підготовка технічної документації та опису програмних модулів
	97		72300	Разом основна заробітна плата Z_{osn}

Основна заробітна плата визначається як сума витрат за всіма етапами проєктування:

$$Z_{osn} = \sum t_i \cdot R_{hi}. \quad (6.2)$$

Z_{osn} = 72300 грн.

Додаткова заробітна плата приймається як частка від основної заробітної плати:

$$Z_{dop} = k_{dop} \cdot Z_{osn}, \quad (6.3)$$

Z_{dop} = 0,10 · 72300 = 7230 грн.

Єдиний соціальний внесок розраховується від суми основної та додаткової заробітної плати:

$$S_{soc} = \left(Z_{osn} + Z_{dop}\right) \cdot k_{soc}, \quad (6.4)$$

S_{soc} = (72300 + 7230) · 0,22 = 17496,60 грн.

6.1.2 Розрахунок накладних витрат та амортизації

До накладних витрат відносять витрати на електроенергію, інтернет, знос персонального комп'ютера, інструменту, паяльного обладнання та використання програмних засобів. Для спрощеного розрахунку їх можна прийняти як частку від основної заробітної плати.

$$OVR = k_{ovr} \cdot Z_{osn}, \tag{6.5}$$

$OVR = 0{,}50 \cdot 72300 = 36150$ грн.

Загальні витрати на проєктування визначаються за формулою:

$$C_{dev} = Z_{osn} + Z_{dop} + S_{soc}. \tag{6.6}$$

$C_{dev} = 72300 + 7230 + 17496{,}60 + 36150 = 133176{,}60$ грн.

6.2 Витрати виготовлення прототипу

Витрати виготовлення прототипу охоплюють матеріальні витрати на покупні компоненти, конструктивні елементи, дроти, кріплення, а також витрати на складання, монтаж і первинне налагодження роботизованої руки.

6.2.1 Матеріальні витрати

Вартість кожного виду покупних компонентів визначається як добуток кількості елементів на ціну одиниці:

$$M_i = m_i \cdot c_i, \tag{6.7}$$

де m_i – кількість компонентів і-го виду, шт; c_i – ціна одиниці компонента, грн; M_i – загальна вартість компонентів і-го виду, грн.

Таблиця 6.2 – Витрати на покупні компоненти та доставку

№	Кількість m_i, шт	Ціна c_i, грн	Усього M_i, грн	Найменування компонента
1	1	350,00	350,00	Arduino-сумісна плата керування
2	1	450,00	450,00	Плата спряження / servo motor driver shield
3	4	119,00	476,00	Сервопривод MG90S
4	2	32,00	64,00	Аналоговий двовісний джойстик із кнопкою
5	1	179,00	179,00	Bluetooth-модуль HC-05
6	1	650,00	650,00	Акрилове шасі роботизованої руки із захватом
7	1	33,00	33,00	Утримувач для двох акумуляторів 18650
8	2	87,00	174,00	Акумулятор 18650
9	1	34,00	34,00	Комплект дротів DuPont Female-Female та Female-Male
10	1	30,00	30,00	Кріпильні елементи, стяжки, ізоляційні матеріали
11	1	150,00	150,00	Доставка компонентів
			2590,00	Разом вартість матеріалів M_{sum}

Разом вартість матеріалів визначається за формулою:

$$M_{sum} = \sum m_i \cdot c_i \ . \tag{6.8}$$

M_{sum} = 2590,00 грн.

6.2.2 Розрахунок витрат на складання та налагодження прототипу

Витрати на складання включають монтаж сервоприводів, підключення пульта, встановлення Bluetooth-модуля, перевірку живлення, тестування сервоприводів і первинне налагодження програмних режимів.

$$Z_{sb} = t_{sb} \cdot R_{hsb} \ , \tag{6.9}$$

де t_{sb} – час складання та налагодження прототипу, год; R_{hsb} – годинна ставка виконавця складальних робіт, грн/год; Z_{sb} – витрати на складання, грн.

t_{sb} = 6 год; R_{hsb} = 400 грн/год; Z_{sb} = 2400 грн.

Відрахування на оплату праці під час складання визначаються за формулою:

$$S_{sbsoc} = Z_{sb} \cdot k_{soc} . \quad (6.10)$$

S_{sbsoc} = 2400 · 0,22 = 528 грн.

Загальні витрати на виготовлення прототипу становлять:

$$C_{proto} = M_{sum} + Z_{sb} + S_{sbsoc} . \quad (6.11)$$

C_{proto} = 2590 + 2400 + 528 = 5518 грн.

6.3 Зведений кошторис проєкту

Повна собівартість розроблення та виготовлення дослідного зразка визначається як сума витрат на проєктування і витрат на створення прототипу.

$$C_{total} = C_{dev} + C_{proto} . \quad (6.12)$$

C_{total} = 133176,60 + 5518 = 138694,6 грн.

6.4 Розрахунок точки беззбитковості

Для оцінки потенційної комерційної доцільності проєкту витрати поділяють на постійні та змінні. Постійні витрати пов’язані з розробленням системи і не залежать від кількості виготовлених виробів. Змінні витрати припадають на виготовлення однієї одиниці продукції.

6.4.1 Вихідні дані для комерційного розрахунку

Таблиця 6.3 – Вихідні дані для розрахунку беззбитковості

Показник	Позначення	Значення	Коментар
Ринкова ціна виробу з ПДВ	P_{val}	4500,00 грн	Орієнтовна ціна навчального роботизованого маніпулятора
Ставка ПДВ	k_{PDV}	20 %	
Ціна без ПДВ	P	3750,00 грн	$P = P_{val} / 1{,}20$
Постійні витрати	FC	133176,60 грн	$FC = C_{dev}$
Змінні витрати на одиницю	VC	2600,00 грн	Серійна собівартість одного виробу без ПДВ
Плановий прибуток	PP	500000,00 грн	

Ціна без ПДВ визначається за формулою:

$$P = P_{val} / (1 + k_{PDV}). \quad (6.13)$$

P = 4500 / 1,20 = 3750 грн.

6.4.2 Розрахунок маржинального доходу на одиницю продукції

Маржинальний дохід показує, яка сума залишається після продажу одного виробу для покриття постійних витрат і формування прибутку:

$$MR = P - VC. \quad (6.14)$$

MR = 3750 − 2600 = 1150 грн.

6.4.3 Розрахунок точки беззбитковості

Точка беззбитковості у натуральному вираженні показує, скільки виробів потрібно продати, щоб повністю покрити витрати на розроблення:

$$BEP_{units} = FC / MR. \quad (6.15)$$

BEP_{units} = 133176,60 / 1150 = 116 од.

Точка беззбитковості у грошовому вираженні без ПДВ визначається за формулою:

$$BEP_{money} = BEP_{units} \cdot P. \quad (6.16)$$

$$BEP_{money} = 116 \cdot 3750 = 435000 \text{ грн.}$$

Точка беззбитковості у грошовому вираженні з ПДВ визначається за формулою:

$$BEP_{moneyval} = BEP_{units} \cdot P_{val}. \quad (6.17)$$

$$BEP_{moneyval} = 116 \cdot 4500 = 522000 \text{ грн.}$$

6.4.4 Розрахунок обсягу продажу для отримання планового прибутку

Необхідний обсяг продажу для отримання заданого планового прибутку визначається за формулою:

$$Q_{target} = (FC + PP) / MR. \quad (6.18)$$

$$Q_{target} = (133176{,}60 + 500000) / 1150 = 551 \text{ од.}$$

Таблиця 6.4 – Зведений фінансовий план проєкту

Показник	Значення	Коментар
Обсяг продажів	551	Цільовий обсяг продажу для отримання планового прибутку
Виручка загальна з ПДВ	2479500,00	$Q_{target} \cdot P_{val}$
ПДВ до сплати	413250,00	20 % від виручки з ПДВ
Виручка чиста без ПДВ	2066250,00	$Q_{target} \cdot P$
Постійні витрати на розроблення	133176,60	FC
Змінні витрати виробництва	1432600,00	$Q_{target} \cdot VC$
Орієнтовний чистий прибуток	500473,40	Виручка без ПДВ − FC − змінні витрати

Висновки. Щоб повністю окупити створення платформи необхідно продати мінімальну партію: 551 од. Кожен наступний продаж приноситиме 1150 грн прибутку (після вирахування податків та вартості деталей).

ВИСНОВКИ

У результаті виконання дипломного проєкту було розроблено та реалізовано роботизовану руку з механічним захопленням на базі Arduino-сумісної мікроконтролерної платформи. У роботі підібрано основні апаратні компоненти системи: плату керування, чотири сервоприводи MG90S, плату спряження, два аналогові джойстики, Bluetooth-модуль HC-05, систему живлення та конструктивні елементи маніпулятора.

Під час практичної реалізації було виконано складання виконавчої частини, підключення сервоприводів, пульта ручного керування, акумуляторного живлення та бездротового модуля. Перевірка окремих вузлів підтвердила працездатність обраної апаратної структури та можливість керування всіма основними віссями роботизованої руки.

Основним результатом роботи стало розроблення програмного забезпечення, яке підтримує ручне, дистанційне та програмне керування маніпулятором. У ручному режимі оператор може плавно змінювати положення ланок за допомогою джойстиків. У програмі передбачено калібрування нейтральних положень, вибір швидкості переміщення та обмеження кутів сервоприводів, що підвищує зручність і безпеку роботи пристрою.

Для дистанційного керування реалізовано обмін командами через послідовний інтерфейс і Bluetooth-з’єднання. Це дозволяє задавати положення маніпулятора, запускати підготовлені сценарії руху та зупиняти виконання за потреби. Додатково передбачено збереження програм в енергонезалежній пам’яті та використання вбудованих сценаріїв, що підвищує автономність системи після налаштування.

Особливе значення має розроблений спосіб програмування рухів. Рух маніпулятора описується через кадри, послідовності, повторення, затримки та параметри швидкості. Такий підхід є розширюваним і потенційно може бути використаний не лише для цієї роботизованої руки, а й як основа для інших

робототехнічних систем, де потрібно задавати послідовність дій виконавчих механізмів.

Розроблена роботизована рука може використовуватися як навчально-дослідний стенд для вивчення мікроконтролерного керування, роботи сервоприводів, Bluetooth-зв’язку та принципів програмного задання рухів. Запропонована програмна архітектура має потенціал для подальшого розвитку та адаптації до інших задач робототехніки.

ЛІТЕРАТУРА

1. Дорохов М.Ю. Конспект лекцій «Роботи та маніпулятори», Донбаська державна машинобудівна академія. Краматорськ, 2019 р., 53 с.
2. Кузнєцов Ю.М. Верстати з ЧПК та верстатні комплекси: / Навч. посібник К.: ТОВ «Замок»; Тернопіль, 2001. Т.1. 198 с., Т.2 . 298 с.
3. Дудюк Д.Л. Гнучке автоматизоване виробництво і роботизовані комплекси / Д.Л. Дудюк, С.С. Мазепа, М.М. Мисик / Навч. посібник. Львів:«Магнолія плюс», видавець СГД ФО В.М.Піча, 2005. – 278с.
4. О. Бішоп, Книга розробника роботів, Київ, “МК-Пресс”, 2010 о, 400с.
5. Офіційний сайт Arduino. - URL: https://www.arduino.com. (дата звернення: 11.05.2023).
6. Програмування ардуїно. - URL: https://doc.arduino.ua/ru/prog/ (дата звернення: 12.03.2026).
7. Ловейкін В.С., Ромасевич Ю.О. Навчальний посібник «Мехатроніка»: Київ, 2012 р., 357 с.
8. ДСТУ 1.5:2015. Правила розроблення, викладання та оформлення національних нормативних документів. URL: https://udhtu.edu.ua/wp-content/uploads/2018/03/DSTY_1_5_2015.pdf (дата звернення: 09.06.2023).
9. ДСТУ 3582-97. Інформація та документація. Скорочення слів в українській мові у бібліографічному описі. Загальні вимоги та правила. URL: https://uni-sport.edu.ua/sites/default/files/vseDocumenti/dokument2. pdf (дата звернення: 09.06.2023).
10. Додаток Android – термінал Bluetooth. Serial Bluetooth Terminal /Owner: Kai Morich https://play.google.com/store/apps/details?id=de.kai_morich.serial_bluetooth_terminal [Дата звернення: 01.05.2026].
11. Електробезпека [Текст]: підручник / С. В. Панченко, О. І. Акімов, М. М. Бабаєв та ін. Харків : УкрДУЗТ, 2018. 295 с.

12. Голінько В.І. Основи охорони праці : підручник. Донецьк : НГУ, 2014. 271 с.

13. Економіка підприємства: Підручник під ред.проф. О.М.Волкова.-М.:ІНФРА-М,1999.-416с.

14. Fritzing GmbH. Learning. Fritzing Documentation. Retrieved [04.05.2026], from https://fritzing.org/learning/

ДОДАТОК А – Основна програма управління роботом-маніпулятором, модуль DP.ino

```
#include <Servo.h>
#include <avr/pgmspace.h>

#include "Robo.h"
#include "ArmServo.h"
#include "Playback.h"
#include "DefaultProg.h"
#include "RoboProgramManager.h"
#include "BluetoothUploadParser.h"
#include "RoboVersion.h"

// Швидкості руху сервоприводів
const int SPEED_SLOW = 20;   // Мале відхилення
const int SPEED_MEDIUM = 15; // Середнє відхилення
const int SPEED_FAST = 5;    // Повне відхилення

// Зони відхилення джойстика від центру
const int JOYSTICK_DEADZONE = 80;
const int JOYSTICK_MEDIUM_ZONE = 280;
const int JOYSTICK_FAST_ZONE = 500;

// Параметри калібрування центру джойстиків
const int CALIBRATION_READS = 30;
const int CALIBRATION_DELAY = 5;

// Затримка від брязкоту для одночасного натискання двох кнопок
const unsigned long BOTH_KEYS_DEBOUNCE_MS = 60;

// Максимальна пауза між двома одночасними натисканнями для double-click
const unsigned long BOTH_KEYS_DOUBLE_CLICK_MS = 700;

// Номер вбудованої програми, яка використовується як останній запасний
варіант
const byte FALLBACK_BUILTIN_PROGRAM = 1;

// Піни сервоприводів
const int PIN_ROTATE = A1;    // Основа
const int PIN_STRETCH = 8;    // Правий сервопривод
const int PIN_ELEVATION = A0; // Лівий сервопривод
const int PIN_CLAW = 9;       // Захват

// Піни правого джойстика
const int RIGHT_X = A2;
const int RIGHT_Y = A5;
```

```
const int RIGHT_KEY = 7;

// Піни лівого джойстика
const int LEFT_X = A3;
const int LEFT_Y = A4;
const int LEFT_KEY = 6;

ArmServo Claw(PIN_CLAW, 19, 19, 110);
ArmServo Rotate(PIN_ROTATE, 90, 1, 180);
ArmServo Elevation(PIN_ELEVATION, 130, 0, 180);
ArmServo Stretch(PIN_STRETCH, 60, 0, 180);

// Роботизована рука, що працює з уже створеними сервоприводами
RoboArm Arm(Claw, Rotate, Elevation, Stretch);

// Буфер програми рухів
RoboProgramBuffer ProgramBuffer;

// Менеджер збереження, завантаження та вбудованих програм
RoboProgramManager ProgramManager(ProgramBuffer);

// Парсер команд з термінала / Bluetooth
BluetoothUploadParser Parser(ProgramBuffer, ProgramManager, Serial);

// Програвач програми
Playback Player;

// Значення правого джойстика
int rightX;
int rightY;
int rightKey;

// Значення лівого джойстика
int leftX;
int leftY;
int leftKey;

// Відкалібровані центри осей джойстиків
int rightXCenter;
int rightYCenter;
int leftXCenter;
int leftYCenter;

// Стан одиночного / подвійного кліку двома кнопками
bool lastBothKeysRawPressed = false;
bool stableBothKeysPressed = false;
unsigned long bothKeysRawChangedTime = 0;
bool waitingSecondBothKeysClick = false;
unsigned long firstBothKeysClickTime = 0;

// Стан негайного виконання одного кадру з команди IMMEDIATE
```

```
RoboActionFrame ImmediateFrame;
bool immediateActive = false;
bool immediateWaitingDelay = false;
unsigned long immediateDelayStart = 0;

void setup()
{
  Serial.begin(9600);

  initJoysticks();
  calibrateJoysticks();

  Arm.begin();

  readJoysticks();
  initBothKeysControl();

  Player.begin(ProgramBuffer, Arm);

  printStartupInfo();
}

void loop()
{
  readJoysticks();

  readSerialInput();
  handleParserPlaybackRequest();
  handleBothKeysPlaybackControl();

  if (Player.isActive())
  {
    // Під час програвання осі джойстиків заблоковані.
    // Працюють тільки STOP з термінала та одночасне натискання двох
кнопок.
    Player.update();
    Arm.update();
    return;
  }

  if (immediateActive)
  {
    // Під час IMMEDIATE осі джойстиків теж заблоковані.
    updateImmediateFrame();
    Arm.update();
    return;
  }

  controlClaw();
  controlRotate();
  controlElevation();
```

```
  controlStretch();

  Arm.update();
}

// Надрукувати коротку інформацію після старту
void printStartupInfo()
{
  Serial.print(F("RoboArm "));
  printTwoDigits(ProgramManager.getCurrentMajorVersion());
  Serial.print(F("."));
  printTwoDigits(ProgramManager.getCurrentMinorVersion());
  Serial.println();

  Serial.println(F("Commands: START UPLOAD, END UPLOAD, INFO, PRINT,
CLEAR, SAVE, LOAD, BUILTIN, DEFLOAD n, ONCE, LOOP, STOP, STATE,
IMMEDIATE"));
}

// Надрукувати число версії у два знаки
void printTwoDigits(byte value)
{
  Serial.print(value / 10);
  Serial.print(value % 10);
}

// Явна ініціалізація джойстиків
void initJoysticks()
{
  // Аналогові осі джойстиків
  pinMode(RIGHT_X, INPUT);
  pinMode(RIGHT_Y, INPUT);
  pinMode(LEFT_X, INPUT);
  pinMode(LEFT_Y, INPUT);

  // Кнопки джойстиків
  // INPUT_PULLUP: натиснуто = LOW, відпущено = HIGH
  pinMode(RIGHT_KEY, INPUT_PULLUP);
  pinMode(LEFT_KEY, INPUT_PULLUP);
}

// Калібрування середнього положення джойстиків
void calibrateJoysticks()
{
  long rightXSum = 0;
  long rightYSum = 0;
  long leftXSum = 0;
  long leftYSum = 0;

  for (int i = 0; i < CALIBRATION_READS; i++)
  {
```

```
    rightXSum += analogRead(RIGHT_X);
    rightYSum += analogRead(RIGHT_Y);
    leftXSum += analogRead(LEFT_X);
    leftYSum += analogRead(LEFT_Y);

    delay(CALIBRATION_DELAY);
  }

  rightXCenter = rightXSum / CALIBRATION_READS;
  rightYCenter = rightYSum / CALIBRATION_READS;
  leftXCenter = leftXSum / CALIBRATION_READS;
  leftYCenter = leftYSum / CALIBRATION_READS;
}

// Читання значень з обох джойстиків
void readJoysticks()
{
  rightX = analogRead(RIGHT_X);
  rightY = analogRead(RIGHT_Y);
  rightKey = digitalRead(RIGHT_KEY);

  leftX = analogRead(LEFT_X);
  leftY = analogRead(LEFT_Y);
  leftKey = digitalRead(LEFT_KEY);
}

// Ініціалізувати захист double-click двома кнопками
void initBothKeysControl()
{
  lastBothKeysRawPressed = areBothKeysPressed();
  stableBothKeysPressed = lastBothKeysRawPressed;
  bothKeysRawChangedTime = millis();
  waitingSecondBothKeysClick = false;
  firstBothKeysClickTime = 0;
}

// Перевірити одночасне натискання двох кнопок джойстиків
bool areBothKeysPressed()
{
  return rightKey == LOW && leftKey == LOW;
}

// Прочитати всі доступні символи з термінала / Bluetooth
void readSerialInput()
{
  while (Serial.available() > 0)
  {
    char ch = (char)Serial.read();
    Parser.feed(ch);
  }
}
```

```
// Обробити запит програвання, який виставив парсер команд
void handleParserPlaybackRequest()
{
  if (Parser.isStateRequested())
  {
    printCurrentState();
    Parser.clearPlaybackRequest();
    return;
  }

  if (Parser.isStopRequested())
  {
    Player.stop();
    stopImmediateFrame();
    Parser.clearPlaybackRequest();
    return;
  }

  if (Parser.isImmediateRequested())
  {
    Player.stop();
    startImmediateFrame(Parser.getImmediateFrame());
    Parser.clearPlaybackRequest();
    return;
  }

  if (Parser.isOnceRequested())
  {
    stopImmediateFrame();
    Player.stop();
    Player.startOnce();
    Parser.clearPlaybackRequest();
    return;
  }

  if (Parser.isLoopRequested())
  {
    stopImmediateFrame();
    Player.stop();
    Player.startLoop();
    Parser.clearPlaybackRequest();
    return;
  }
}

// Надрукувати поточні кути осей у форматі команд програми
void printCurrentState()
{
  int rotate = Arm.Rotate.getPos();
  int claw = Arm.Claw.getPos();
```

```
  int elevation = Arm.Elevation.getPos();
  int stretch = Arm.Stretch.getPos();

  printAxisState('r', rotate);
  printAxisState('c', claw);
  printAxisState('e', elevation);
  printAxisState('s', stretch);

  Serial.print(F("IMMEDIATE "));
  printOneLineStateFrame(rotate, claw, elevation, stretch);
}

// Надрукувати одну вісь у форматі <axis>:<angle>;
void printAxisState(char axis, int angle)
{
  Serial.print(axis);
  Serial.print(F(":"));
  Serial.print(angle);
  Serial.println(F(";"));
}

// Надрукувати поточний стан одним кадром в один рядок
void printOneLineStateFrame(int rotate, int claw, int elevation, int
stretch)
{
  Serial.print(F("r:"));
  Serial.print(rotate);
  Serial.print(F("; c:"));
  Serial.print(claw);
  Serial.print(F("; e:"));
  Serial.print(elevation);
  Serial.print(F("; s:"));
  Serial.print(stretch);
  Serial.println(F("; delay:0;"));
}

// Запустити негайне виконання одного кадру
void startImmediateFrame(const RoboActionFrame& frame)
{
  ImmediateFrame = frame;
  immediateActive = true;
  immediateWaitingDelay = false;
  immediateDelayStart = 0;

  Arm.stop();
  Serial.println(F("immediate started"));
}

// Зупинити негайне виконання одного кадру
void stopImmediateFrame()
{
```

```
  if (immediateActive)
  {
    Serial.println(F("immediate stopped"));
  }

  immediateActive = false;
  immediateWaitingDelay = false;
  Arm.stop();
}

// Оновити негайне виконання одного кадру
void updateImmediateFrame()
{
  if (!immediateActive)
  {
    return;
  }

  if (immediateWaitingDelay)
  {
    if (millis() - immediateDelayStart < ImmediateFrame.getDelayMs())
    {
      return;
    }

    immediateActive = false;
    immediateWaitingDelay = false;
    Serial.println(F("immediate done"));
    return;
  }

  bool frameDone = executeActionFrame(ImmediateFrame);

  if (frameDone)
  {
    immediateWaitingDelay = true;
    immediateDelayStart = millis();
  }
}

// Виконати один кадр для команди IMMEDIATE
bool executeActionFrame(const RoboActionFrame& frame)
{
  bool allDone = true;

  if (frame.Rotate.isEnabled())
  {
    allDone           =           Arm.Rotate.moveTo(frame.Rotate.angle,
frame.Rotate.slowness) && allDone;
  }
```

```
  if (frame.Elevation.isEnabled())
  {
    allDone        =        Arm.Elevation.moveTo(frame.Elevation.angle,
frame.Elevation.slowness) && allDone;
  }

  if (frame.Stretch.isEnabled())
  {
    allDone          =          Arm.Stretch.moveTo(frame.Stretch.angle,
frame.Stretch.slowness) && allDone;
  }

  if (frame.Claw.isEnabled())
  {
    allDone = Arm.Claw.moveTo(frame.Claw.angle, frame.Claw.slowness) &&
allDone;
  }

  return allDone;
}

// Обробити double-click двома кнопками джойстиків
void handleBothKeysPlaybackControl()
{
  bool rawBothKeysPressed = areBothKeysPressed();
  unsigned long now = millis();

  if (rawBothKeysPressed != lastBothKeysRawPressed)
  {
    lastBothKeysRawPressed = rawBothKeysPressed;
    bothKeysRawChangedTime = now;
  }

  if (rawBothKeysPressed != stableBothKeysPressed)
  {
    if (now - bothKeysRawChangedTime >= BOTH_KEYS_DEBOUNCE_MS)
    {
      stableBothKeysPressed = rawBothKeysPressed;

      // Клік рахується тільки коли ОБИДВІ кнопки стабільно натиснуті.
      // Одне натискання одного джойстика тут не повинно нічого робити.
      if (stableBothKeysPressed)
      {
        handleBothKeysClick(now);
      }
    }
  }

  if (waitingSecondBothKeysClick)
  {
    if (now - firstBothKeysClickTime > BOTH_KEYS_DOUBLE_CLICK_MS)
```

```
    {
      waitingSecondBothKeysClick = false;
      handleBothKeysSingleClick();
    }
  }
}

// Обробити один підтверджений клік двома кнопками
void handleBothKeysClick(unsigned long now)
{
  if (Player.isActive())
  {
    waitingSecondBothKeysClick = false;
    Player.stop();
    Serial.println(F("playback stopped by joystick"));
    return;
  }

  if (immediateActive)
  {
    waitingSecondBothKeysClick = false;
    stopImmediateFrame();
    return;
  }

  if (!waitingSecondBothKeysClick)
  {
    waitingSecondBothKeysClick = true;
    firstBothKeysClickTime = now;
    Serial.println(F("both keys click: waiting 700 ms for LOOP,
otherwise ONCE"));
    return;
  }

  waitingSecondBothKeysClick = false;
  handleBothKeysDoubleClick();
}

// Виконати дію після одиночного кліку двома кнопками
void handleBothKeysSingleClick()
{
  if (Player.isActive())
  {
    Player.stop();
    Serial.println(F("playback stopped by joystick"));
    return;
  }

  if (immediateActive)
  {
    stopImmediateFrame();
```

```
    return;
  }

  startJoystickPlaybackOnce();
}

// Виконати дію після double-click двома кнопками
void handleBothKeysDoubleClick()
{
  if (Player.isActive())
  {
    Player.stop();
    Serial.println(F("playback stopped by joystick double-click"));
    return;
  }

  if (immediateActive)
  {
    stopImmediateFrame();
    return;
  }

  startJoystickPlaybackLoop();
}

// Запустити одноразове програвання програми по одному кліку двома кнопками
void startJoystickPlaybackOnce()
{
  if (!prepareProgramForJoystickPlayback())
  {
    Serial.println(F("cannot start: no valid program"));
    return;
  }

  Player.startOnce();

  if (Player.isActive())
  {
    Serial.println(F("play once started by joystick"));
  }
  else
  {
    Serial.println(F("cannot start: program is empty"));
  }
}

// Запустити циклічне програвання програми по double-click двома кнопками
void startJoystickPlaybackLoop()
{
```

```
  if (!prepareProgramForJoystickPlayback())
  {
    Serial.println(F("cannot start: no valid program"));
    return;
  }

  Player.startLoop();

  if (Player.isActive())
  {
    Serial.println(F("play loop started by joystick double-click"));
  }
  else
  {
    Serial.println(F("cannot start: program is empty"));
  }
}

// Підготувати програму для запуску з джойстика
bool prepareProgramForJoystickPlayback()
{
  if (ProgramBuffer.sequenceCount > 0)
  {
    Serial.print(F("using loaded program \""));
    Serial.print(ProgramManager.getProgramName());
    Serial.println(F("\""));
    return true;
  }

  RoboProgramManager::LoadStatus eepromStatus = ProgramManager.load();

  if (eepromStatus == RoboProgramManager::LOAD_OK)
  {
    Serial.print(F("EEPROM program loaded \""));
    Serial.print(ProgramManager.getProgramName());
    Serial.println(F("\""));
    return true;
  }

  Serial.print(F("EEPROM load failed: "));
  printLoadStatus(eepromStatus);

  RoboProgramManager::LoadStatus            builtinStatus            =
ProgramManager.loadBuiltin(FALLBACK_BUILTIN_PROGRAM);

  if (builtinStatus == RoboProgramManager::LOAD_OK)
  {
    Serial.print(F("fallback builtin loaded \""));
    Serial.print(ProgramManager.getProgramName());
    Serial.println(F("\""));
    return true;
```

```
  }

  Serial.print(F("fallback builtin failed: "));
  printLoadStatus(builtinStatus);

  return false;
}

// Надрукувати короткий статус помилки завантаження
void printLoadStatus(RoboProgramManager::LoadStatus status)
{
  if (status == RoboProgramManager::LOAD_OK)
  {
    Serial.println(F("ok"));
    return;
  }

  if (status == RoboProgramManager::LOAD_BAD_DESCRIPTOR)
  {
    Serial.println(F("bad descriptor"));
    return;
  }

  if (status == RoboProgramManager::LOAD_MAJOR_MISMATCH)
  {
    Serial.print(F("major mismatch stored="));
    Serial.print(ProgramManager.getLastMajorVersion());
    Serial.print(F(", current="));
    Serial.println(ProgramManager.getCurrentMajorVersion());
    return;
  }

  if (status == RoboProgramManager::LOAD_BAD_NAME)
  {
    Serial.println(F("bad name"));
    return;
  }

  if (status == RoboProgramManager::LOAD_BAD_PROGRAM)
  {
    Serial.println(F("bad program"));
    return;
  }

  if (status == RoboProgramManager::LOAD_BAD_INDEX)
  {
    Serial.println(F("bad index"));
    return;
  }

  Serial.println(F("unknown error"));
```

```
}

// Отримати затримку руху за відхиленням від центру
// 0 означає, що джойстик перебуває у мертвій зоні
int getSpeedByDeviation(int value, int center)
{
  int deviation = abs(value - center);

  if (deviation < JOYSTICK_DEADZONE)
  {
    return 0;
  }

  if (deviation < JOYSTICK_MEDIUM_ZONE)
  {
    return SPEED_SLOW;
  }

  if (deviation < JOYSTICK_FAST_ZONE)
  {
    return SPEED_MEDIUM;
  }

  return SPEED_FAST;
}

// Керування сервоприводом за координатою джойстика
void controlServoByAxis(ArmServo& servoUnit, int value, int center, bool
inverted)
{
  int speed = getSpeedByDeviation(value, center);

  if (speed == 0)
  {
    return;
  }

  if (value < center)
  {
    if (inverted)
    {
      servoUnit.requestIncrease(speed);
    }
    else
    {
      servoUnit.requestDecrease(speed);
    }
  }

  if (value > center)
  {
```

```
    if (inverted)
    {
      servoUnit.requestDecrease(speed);
    }
    else
    {
      servoUnit.requestIncrease(speed);
    }
  }
}

// Керування захватом
void controlClaw()
{
  controlServoByAxis(Arm.Claw, leftX, leftXCenter, false);
}

// Керування поворотом основи
void controlRotate()
{
  controlServoByAxis(Arm.Rotate, rightX, rightXCenter, false);
}

// Керування підйомом
void controlElevation()
{
  controlServoByAxis(Arm.Elevation, rightY, rightYCenter, false);
}

// Керування витягуванням
void controlStretch()
{
  controlServoByAxis(Arm.Stretch, leftY, leftYCenter, true);
}
```

ДОДАТОК Б – Модуль ArmServo.h

```
#ifndef ARM_SERVO_H
#define ARM_SERVO_H

#include <Arduino.h>
#include <Servo.h>

// Крок зміни кута за замовчуванням
const int DEFAULT_STEP = 1;

// Напрямки руху
const int DIR_STOP = 0;
const int DIR_DECREASE = -1;
const int DIR_INCREASE = 1;

// Клас для зберігання поточного кута та допустимих меж
class Angles
{
private:
  int lowest;
  int highest;
  int pos;

public:
  // startPos одразу стає поточною позицією
  Angles(int startPos, int minAngle, int maxAngle)
    :  lowest(minAngle),  highest(maxAngle),  pos(constrain(startPos,
minAngle, maxAngle))
  {
  }

  // Зменшити кут з обмеженням знизу
  bool decrease(int angle = DEFAULT_STEP)
  {
    int newPos = max(lowest, pos - angle);

    if (newPos == pos)
    {
      return false;
    }

    pos = newPos;
    return true;
  }

  // Збільшити кут з обмеженням зверху
```

```
  bool increase(int angle = DEFAULT_STEP)
  {
    int newPos = min(highest, pos + angle);

    if (newPos == pos)
    {
      return false;
    }

    pos = newPos;
    return true;
  }

  // Зробити один крок до цільового кута
  bool moveToward(int targetAngle)
  {
    targetAngle = constrain(targetAngle, lowest, highest);

    if (pos < targetAngle)
    {
      return increase();
    }

    if (pos > targetAngle)
    {
      return decrease();
    }

    return false;
  }

  // Перевірити, чи досягнуто цільового кута
  bool isAt(int targetAngle)
  {
    targetAngle = constrain(targetAngle, lowest, highest);
    return pos == targetAngle;
  }

  // Отримати поточний кут
  int getPos()
  {
    return pos;
  }
};

// Клас для плавного керування одним сервоприводом роботизованої руки
class ArmServo
{
private:
  Servo servo;
  Angles angles;
```

```
  int pin;
  int direction;
  int stepDelay;

  unsigned long lastMoveTime;

public:
  // servoPin – пін сервопривода
  // startPos – стартова позиція
  // minAngle – мінімальний допустимий кут
  // maxAngle – максимальний допустимий кут
  ArmServo(int servoPin, int startPos, int minAngle, int maxAngle)
    : angles(startPos, minAngle, maxAngle)
  {
    pin = servoPin;
    direction = DIR_STOP;
    stepDelay = 20;
    lastMoveTime = 0;
  }

  // Підключити сервопривод і виставити стартову позицію
  void begin()
  {
    servo.attach(pin);
    servo.write(angles.getPos());
  }

  // Зупинити поточну ручну команду
  void stop()
  {
    direction = DIR_STOP;
  }

  // Запросити збільшення кута
  void requestIncrease(int delayMs)
  {
    direction = DIR_INCREASE;
    stepDelay = delayMs;
  }

  // Запросити зменшення кута
  void requestDecrease(int delayMs)
  {
    direction = DIR_DECREASE;
    stepDelay = delayMs;
  }

  // Оновити сервопривод без блокувального delay()
  void update()
  {
```

```
  if (direction == DIR_STOP)
  {
    return;
  }

  unsigned long now = millis();

  if (now - lastMoveTime < stepDelay)
  {
    return;
  }

  lastMoveTime = now;

  if (direction == DIR_INCREASE)
  {
    if (angles.increase())
    {
      servo.write(angles.getPos());
    }
  }

  if (direction == DIR_DECREASE)
  {
    if (angles.decrease())
    {
      servo.write(angles.getPos());
    }
  }

  direction = DIR_STOP;
}

// Рухати сервопривод до конкретного кута
// Повертає true, якщо цільовий кут уже досягнуто
bool moveTo(int targetAngle, int delayMs)
{
  direction = DIR_STOP;

  if (angles.isAt(targetAngle))
  {
    return true;
  }

  unsigned long now = millis();

  if (now - lastMoveTime < delayMs)
  {
    return false;
  }
```

```
    lastMoveTime = now;

    if (angles.moveToward(targetAngle))
    {
      servo.write(angles.getPos());
    }

    return angles.isAt(targetAngle);
  }

  // Отримати поточний кут сервопривода
  int getPos()
  {
    return angles.getPos();
  }
};

// Клас усієї роботизованої руки
class RoboArm
{
public:
  ArmServo& Claw;
  ArmServo& Rotate;
  ArmServo& Elevation;
  ArmServo& Stretch;

  // Приймає вже створені та налаштовані сервоприводи
  RoboArm(
    ArmServo& clawServo,
    ArmServo& rotateServo,
    ArmServo& elevationServo,
    ArmServo& stretchServo
  )
    : Claw(clawServo),
      Rotate(rotateServo),
      Elevation(elevationServo),
      Stretch(stretchServo)
  {
  }

  // Ініціалізувати всі сервоприводи
  void begin()
  {
    Rotate.begin();
    delay(1000);

    Stretch.begin();
    Elevation.begin();
    Claw.begin();

    delay(1500);
```

```
  }

  // Зупинити всі поточні команди
  void stop()
  {
    Claw.stop();
    Rotate.stop();
    Elevation.stop();
    Stretch.stop();
  }

  // Оновити всі сервоприводи
  void update()
  {
    Claw.update();
    Rotate.update();
    Elevation.update();
    Stretch.update();
  }
};

#endif
```

ДОДАТОК В – Модуль Robo.h

```
#ifndef ROBO_H
#define ROBO_H

#include <Arduino.h>
#include <EEPROM.h>
#include <avr/pgmspace.h>

class RoboByteReader {
public:
  virtual byte readByte(int& address) = 0;
};

class RoboEepromReader : public RoboByteReader {
public:
  byte readByte(int& address) {
    byte value = EEPROM.read(address);
    address++;

    return value;
  }
};

class RoboProgmemReader : public RoboByteReader {
public:
  const byte* Data;

  RoboProgmemReader(const byte* data) {
    Data = data;
  }

  byte readByte(int& address) {
    byte value = pgm_read_byte_near(Data + address);
    address++;

    return value;
  }
};

class RoboAxisDestination {
public:
  static const byte NO_OPERATION_ANGLE = 255;

  static const byte DEFAULT_SLOWNESS = 10;
  static const byte MIN_SLOWNESS = 1;
  static const byte MAX_SLOWNESS = 30;
```

```
static const byte FLAG_NONE = 0b00;
static const byte FLAG_ANGLE = 0b10;
static const byte FLAG_SLOWNESS = 0b01;

byte angle;
byte slowness;

RoboAxisDestination() {
  noOperation();
}

void noOperation() {
  angle = NO_OPERATION_ANGLE;
  slowness = DEFAULT_SLOWNESS;
}

bool isEnabled() const {
  return angle != NO_OPERATION_ANGLE;
}

void setMove(byte targetAngle, byte targetSlowness) {
  angle = constrain(targetAngle, 0, 180);
  slowness = constrain(targetSlowness, MIN_SLOWNESS, MAX_SLOWNESS);
}

byte getFlags() const {
  if (!isEnabled()) {
    return FLAG_NONE;
  }

  if (slowness == DEFAULT_SLOWNESS) {
    return FLAG_ANGLE;
  }

  return FLAG_ANGLE | FLAG_SLOWNESS;
}

void compileFlags(byte axisFlags) {
  axisFlags = axisFlags & 0b11;

  bool hasAngle = (axisFlags & FLAG_ANGLE) != 0;
  bool hasSlowness = (axisFlags & FLAG_SLOWNESS) != 0;

  if (!hasAngle) {
    noOperation();
    return;
  }

  if (!hasSlowness) {
    slowness = DEFAULT_SLOWNESS;
```

```
      return;
    }

    slowness = constrain(slowness, MIN_SLOWNESS, MAX_SLOWNESS);
  }

  void writeToEeprom(int& address, byte axisFlags) const {
    axisFlags = axisFlags & 0b11;

    bool hasAngle = (axisFlags & FLAG_ANGLE) != 0;
    bool hasSlowness = (axisFlags & FLAG_SLOWNESS) != 0;

    if (hasAngle) {
      EEPROM.update(address, angle);
      address++;
    }

    if (hasSlowness) {
      EEPROM.update(address, slowness);
      address++;
    }
  }

  void readFrom(RoboByteReader& reader, int& address, byte axisFlags) {
    axisFlags = axisFlags & 0b11;

    bool hasAngle = (axisFlags & FLAG_ANGLE) != 0;
    bool hasSlowness = (axisFlags & FLAG_SLOWNESS) != 0;

    if (!hasAngle) {
      noOperation();
      return;
    }

    angle = reader.readByte(address);

    if (hasSlowness) {
      slowness = reader.readByte(address);
    } else {
      slowness = DEFAULT_SLOWNESS;
    }

    slowness = constrain(slowness, MIN_SLOWNESS, MAX_SLOWNESS);
  }

  void readFromEeprom(int& address, byte axisFlags) {
    RoboEepromReader reader;
    readFrom(reader, address, axisFlags);
  }
};
```

```
class RoboActionFrame {
public:
  // Біти прапорців:
  // 7 6 5 4 3 2 1 0
  // S s E e C c R r
  //
  // Старший біт пари: є angle
  // Молодший біт пари: є slowness
  //
  // S/s – Stretch
  // E/e – Elevation
  // C/c – Claw
  // R/r – Rotate

  static const byte STRETCH_SHIFT = 6;
  static const byte ELEVATION_SHIFT = 4;
  static const byte CLAW_SHIFT = 2;
  static const byte ROTATE_SHIFT = 0;

  static const byte AXIS_FLAGS_MASK = 0b11;

  byte flags;
  byte delay10;

  RoboAxisDestination Stretch;
  RoboAxisDestination Elevation;
  RoboAxisDestination Claw;
  RoboAxisDestination Rotate;

  RoboActionFrame() {
    clear();
  }

  void clear() {
    flags = 0;
    delay10 = 0;

    Stretch.noOperation();
    Elevation.noOperation();
    Claw.noOperation();
    Rotate.noOperation();
  }

  bool hasAnyAxis() const {
    return Stretch.isEnabled()
      || Elevation.isEnabled()
      || Claw.isEnabled()
      || Rotate.isEnabled();
  }

  void setStretch(byte angle, byte slowness) {
```

```
  Stretch.setMove(angle, slowness);
}

void setElevation(byte angle, byte slowness) {
  Elevation.setMove(angle, slowness);
}

void setClaw(byte angle, byte slowness) {
  Claw.setMove(angle, slowness);
}

void setRotate(byte angle, byte slowness) {
  Rotate.setMove(angle, slowness);
}

byte getAxisFlags(byte shift) const {
  return (flags >> shift) & AXIS_FLAGS_MASK;
}

void buildFlags() {
  flags = 0;

  flags |= (Stretch.getFlags() << STRETCH_SHIFT);
  flags |= (Elevation.getFlags() << ELEVATION_SHIFT);
  flags |= (Claw.getFlags() << CLAW_SHIFT);
  flags |= (Rotate.getFlags() << ROTATE_SHIFT);
}

void compile() {
  Stretch.compileFlags(getAxisFlags(STRETCH_SHIFT));
  Elevation.compileFlags(getAxisFlags(ELEVATION_SHIFT));
  Claw.compileFlags(getAxisFlags(CLAW_SHIFT));
  Rotate.compileFlags(getAxisFlags(ROTATE_SHIFT));
}

void writeToEeprom(int& address) {
  buildFlags();

  EEPROM.update(address, flags);
  address++;

  Stretch.writeToEeprom(address, getAxisFlags(STRETCH_SHIFT));
  Elevation.writeToEeprom(address, getAxisFlags(ELEVATION_SHIFT));
  Claw.writeToEeprom(address, getAxisFlags(CLAW_SHIFT));
  Rotate.writeToEeprom(address, getAxisFlags(ROTATE_SHIFT));

  EEPROM.update(address, delay10);
  address++;
}

void readFrom(RoboByteReader& reader, int& address) {
```

```
    clear();

    flags = reader.readByte(address);

    Stretch.readFrom(reader, address, getAxisFlags(STRETCH_SHIFT));
    Elevation.readFrom(reader, address, getAxisFlags(ELEVATION_SHIFT));
    Claw.readFrom(reader, address, getAxisFlags(CLAW_SHIFT));
    Rotate.readFrom(reader, address, getAxisFlags(ROTATE_SHIFT));

    delay10 = reader.readByte(address);

    compile();
  }

  void readFromEeprom(int& address) {
    RoboEepromReader reader;
    readFrom(reader, address);
  }

  void readFromProgmem(const byte* data, int& address) {
    RoboProgmemReader reader(data);
    readFrom(reader, address);
  }

  unsigned int getDelayMs() const {
    return (unsigned int)delay10 * 100;
  }
};

class RoboSequence {
public:
  static const byte DEFAULT_REPEATS = 1;

  RoboActionFrame* Frames;

  byte frameCount;
  byte repeats;

  RoboSequence() {
    clear();
  }

  void clear() {
    Frames = 0;
    frameCount = 0;
    repeats = DEFAULT_REPEATS;
  }

  void start(RoboActionFrame* frames, byte repeatCount) {
    Frames = frames;
    frameCount = 0;
```

```
  setRepeats(repeatCount);
}

void setRepeats(byte repeatCount) {
  if (repeatCount == 0) {
    repeats = DEFAULT_REPEATS;
    return;
  }

  repeats = repeatCount;
}

bool hasFrameBuffer() const {
  return Frames != 0;
}

bool isEmpty() const {
  return Frames == 0 || frameCount == 0;
}

RoboActionFrame* getFrame(byte index) {
  if (Frames == 0) {
    return 0;
  }

  if (index >= frameCount) {
    return 0;
  }

  return &Frames[index];
}

const RoboActionFrame* getFrame(byte index) const {
  if (Frames == 0) {
    return 0;
  }

  if (index >= frameCount) {
    return 0;
  }

  return &Frames[index];
}

bool addFrame(const RoboActionFrame& frame) {
  if (Frames == 0) {
    return false;
  }

  Frames[frameCount] = frame;
  frameCount++;
```

```
    return true;
  }

  void writeToEeprom(int& address) {
    EEPROM.update(address, repeats);
    address++;

    EEPROM.update(address, frameCount);
    address++;

    for (byte i = 0; i < frameCount; i++) {
      Frames[i].writeToEeprom(address);
    }
  }

  bool readFrom(RoboByteReader& reader, int& address, byte
maxFrameCount) {
    int startAddress = address;

    if (Frames == 0) {
      return false;
    }

    byte storedRepeats = reader.readByte(address);
    byte storedFrameCount = reader.readByte(address);

    if (storedFrameCount > maxFrameCount) {
      address = startAddress;
      clear();
      return false;
    }

    setRepeats(storedRepeats);
    frameCount = 0;

    for (byte i = 0; i < storedFrameCount; i++) {
      Frames[i].readFrom(reader, address);
      frameCount++;
    }

    return true;
  }

  bool readFromEeprom(int& address, byte maxFrameCount) {
    RoboEepromReader reader;
    return readFrom(reader, address, maxFrameCount);
  }

  bool readFromProgmem(const byte* data, int& address, byte
maxFrameCount) {
```

```
    RoboProgmemReader reader(data);
    return readFrom(reader, address, maxFrameCount);
  }
};

class RoboProgramBuffer {
public:
  static const byte MAX_SEQUENCES = 25;
  static const byte MAX_FRAMES = 40;

  RoboSequence Sequences[MAX_SEQUENCES];
  RoboActionFrame FrameBuffer[MAX_FRAMES];

  byte sequenceCount;
  byte frameCount;

  bool sequenceOpen;

private:
  byte currentSequenceIndex;

public:
  RoboProgramBuffer() {
    clear();
  }

  void clear() {
    sequenceCount = 0;
    frameCount = 0;

    sequenceOpen = false;
    currentSequenceIndex = 0;

    for (byte i = 0; i < MAX_SEQUENCES; i++) {
      Sequences[i].clear();
    }

    for (byte i = 0; i < MAX_FRAMES; i++) {
      FrameBuffer[i].clear();
    }
  }

  bool canAddSequence() const {
    return sequenceCount < MAX_SEQUENCES;
  }

  bool canAddFrame() const {
    return frameCount < MAX_FRAMES;
  }

  byte getFreeSequenceCount() const {
```

```
    return MAX_SEQUENCES - sequenceCount;
  }

  byte getFreeFrameCount() const {
    return MAX_FRAMES - frameCount;
  }

  bool isFullBySequences() const {
    return sequenceCount >= MAX_SEQUENCES;
  }

  bool isFullByFrames() const {
    return frameCount >= MAX_FRAMES;
  }

  RoboSequence* startSequence(byte repeats) {
    if (sequenceOpen) {
      return 0;
    }

    if (!canAddSequence()) {
      return 0;
    }

    RoboSequence* sequence = &Sequences[sequenceCount];

    sequence->start(&FrameBuffer[frameCount], repeats);

    currentSequenceIndex = sequenceCount;
    sequenceCount++;
    sequenceOpen = true;

    return sequence;
  }

  bool      addFrameToSequence(RoboSequence*      sequence,      const
RoboActionFrame& frame) {
    if (sequence == 0) {
      return false;
    }

    if (!canAddFrame()) {
      return false;
    }

    if (!sequence->hasFrameBuffer()) {
      return false;
    }

    sequence->addFrame(frame);
    frameCount++;
```

```
  return true;
}

bool addFrame(const RoboActionFrame& frame) {
  if (!sequenceOpen) {
    return false;
  }

  RoboSequence* sequence = getSequence(currentSequenceIndex);

  return addFrameToSequence(sequence, frame);
}

bool finishSequence() {
  if (!sequenceOpen) {
    return false;
  }

  RoboSequence* sequence = getSequence(currentSequenceIndex);

  if (sequence == 0) {
    sequenceOpen = false;
    return false;
  }

  if (sequence->isEmpty()) {
    sequenceOpen = false;
    return false;
  }

  sequenceOpen = false;
  return true;
}

RoboSequence* getSequence(byte index) {
  if (index >= sequenceCount) {
    return 0;
  }

  return &Sequences[index];
}

const RoboSequence* getSequence(byte index) const {
  if (index >= sequenceCount) {
    return 0;
  }

  return &Sequences[index];
}
```

```
  void printStats(Stream& out) const {
    out.println(F("OK UPLOAD"));

    out.print(F("Sequences: "));
    out.println(sequenceCount);

    out.print(F("Total frames: "));
    out.println(frameCount);

    for (byte i = 0; i < sequenceCount; i++) {
      out.print(F("Sequence "));
      out.print(i + 1);
      out.print(F(": repeat="));
      out.print(Sequences[i].repeats);
      out.print(F(", frames="));
      out.println(Sequences[i].frameCount);
    }
  }

  void writeToEeprom(int& address) {
    EEPROM.update(address, sequenceCount);
    address++;

    for (byte i = 0; i < sequenceCount; i++) {
      Sequences[i].writeToEeprom(address);
    }
  }

  bool readFrom(RoboByteReader& reader, int& address) {
    clear();

    byte storedSequenceCount = reader.readByte(address);

    if (storedSequenceCount > MAX_SEQUENCES) {
      clear();
      return false;
    }

    for (byte i = 0; i < storedSequenceCount; i++) {
      if (frameCount >= MAX_FRAMES) {
        clear();
        return false;
      }

      RoboSequence* sequence = &Sequences[sequenceCount];

      sequence->start(&FrameBuffer[frameCount],
RoboSequence::DEFAULT_REPEATS);

      byte framesLeft = MAX_FRAMES - frameCount;
```

```
      bool ok = sequence->readFrom(reader, address, framesLeft);

      if (!ok) {
        clear();
        return false;
      }

      frameCount += sequence->frameCount;
      sequenceCount++;
    }

    sequenceOpen = false;
    currentSequenceIndex = 0;

    return true;
  }

  bool readFromEeprom(int& address) {
    RoboEepromReader reader;
    return readFrom(reader, address);
  }

  bool readFromProgmem(const byte* data, int& address) {
    RoboProgmemReader reader(data);
    return readFrom(reader, address);
  }
};

#endif
```

ДОДАТОК Г – Модуль BluetoothUploadParser.h

```
#ifndef BLUETOOTH_UPLOAD_PARSER_H
#define BLUETOOTH_UPLOAD_PARSER_H

#include <Arduino.h>
#include <avr/pgmspace.h>
#include <string.h>
#include "Robo.h"
#include "RoboProgramManager.h"

class BluetoothUploadParser {
private:
  enum ParserState {
    WAIT_START,
    UPLOADING,
    IMMEDIATE_INPUT
  };

  enum PlaybackRequest {
    PLAY_NONE,
    PLAY_ONCE,
    PLAY_LOOP,
    PLAY_STOP,
    PLAY_STATE,
    PLAY_IMMEDIATE
  };

  static const byte TOKEN_SIZE = 72;

  RoboProgramBuffer &memory;
  RoboProgramManager &programManager;
  Stream &out;

  ParserState state;
  PlaybackRequest playbackRequest;

  bool errorFlag;

  char token[TOKEN_SIZE];
  byte tokenLen;

  unsigned int lineNumber;

  bool inComment;
  bool slashPending;
```

```
  RoboActionFrame pendingFrame;
  RoboActionFrame immediateFrame;

public:
  BluetoothUploadParser(
    RoboProgramBuffer &targetMemory,
    RoboProgramManager &targetProgramManager,
    Stream &output
  )
    : memory(targetMemory),
      programManager(targetProgramManager),
      out(output)
  {
    reset();
  }

  void reset() {
    state = WAIT_START;
    playbackRequest = PLAY_NONE;
    errorFlag = false;

    tokenLen = 0;
    token[0] = '\0';

    lineNumber = 1;
    inComment = false;
    slashPending = false;

    pendingFrame.clear();
    immediateFrame.clear();
  }

  bool isUploading() const {
    return state == UPLOADING;
  }

  bool hasError() const {
    return errorFlag;
  }

  void clearError() {
    errorFlag = false;
  }

  bool isOnceRequested() const {
    return playbackRequest == PLAY_ONCE;
  }

  bool isLoopRequested() const {
    return playbackRequest == PLAY_LOOP;
  }
```

```
bool isStopRequested() const {
  return playbackRequest == PLAY_STOP;
}

bool isStateRequested() const {
  return playbackRequest == PLAY_STATE;
}

bool isImmediateRequested() const {
  return playbackRequest == PLAY_IMMEDIATE;
}

const RoboActionFrame& getImmediateFrame() const {
  return immediateFrame;
}

void clearPlaybackRequest() {
  playbackRequest = PLAY_NONE;
}

void printHelp() {
  printAvailableInstructions();
}

void feed(char ch) {
  if (ch == '\r') {
    return;
  }

  if (inComment) {
    if (ch == '\n') {
      inComment = false;

      if (state == IMMEDIATE_INPUT) {
        processCurrentToken(true);
        clearToken();
      }

      lineNumber++;
    }

    return;
  }

  if (slashPending) {
    slashPending = false;

    if (ch == '/') {
      if (tokenLen > 0 && token[tokenLen - 1] == '/') {
        tokenLen--;
```

```
          token[tokenLen] = '\0';
        }

        inComment = true;
        return;
      }
    }

    if (ch == '/') {
      appendChar(ch);
      slashPending = true;
      return;
    }

    if (ch == ';') {
      processCurrentToken(false);
      clearToken();
      return;
    }

    if (ch == '\n') {
      processCurrentToken(true);
      clearToken();
      lineNumber++;
      return;
    }

    appendChar(ch);
  }

private:
  void appendChar(char ch) {
    if (tokenLen >= TOKEN_SIZE - 1) {
      failNearChar(F("token is too long"), ch);
      return;
    }

    token[tokenLen] = ch;
    tokenLen++;
    token[tokenLen] = '\0';
  }

  void clearToken() {
    tokenLen = 0;
    token[0] = '\0';
  }

  void resetInputState() {
    clearToken();

    state = WAIT_START;
```

```
  inComment = false;
  slashPending = false;

  pendingFrame.clear();
}

void processCurrentToken(bool lineEnd) {
  trimToken();

  if (tokenLen == 0) {
    if (state == IMMEDIATE_INPUT && lineEnd) {
      finishImmediateWithoutDelay();
    }

    return;
  }

  if (state == WAIT_START) {
    parseRootCommand(lineEnd);
    return;
  }

  if (state == UPLOADING) {
    if (equalsToken_P(PSTR("STOP"))) {
      abortUploadAndRequestStop();
      return;
    }

    if (equalsToken_P(PSTR("END UPLOAD"))) {
      finishUpload();
      return;
    }

    parseInstruction();
    return;
  }

  if (state == IMMEDIATE_INPUT) {
    parseImmediateInstruction(lineEnd);
    return;
  }
}

void requestStopBeforeProgramChange() {
  playbackRequest = PLAY_STOP;
}

void parseRootCommand(bool lineEnd) {
  if (equalsToken_P(PSTR("START UPLOAD"))) {
    requestStopBeforeProgramChange();
    memory.clear();
```

```
    pendingFrame.clear();

    lineNumber = 1;
    state = UPLOADING;
    errorFlag = false;

    out.println(F("UPLOAD STARTED"));
    return;
  }

  if (equalsToken_P(PSTR("INFO"))) {
    printInfo();
    return;
  }

  if (equalsToken_P(PSTR("PRINT"))) {
    printProgram();
    return;
  }

  if (equalsToken_P(PSTR("CLEAR"))) {
    requestStopBeforeProgramChange();
    clearProgram();
    return;
  }

  if (equalsToken_P(PSTR("BUILTIN"))) {
    printBuiltinPrograms();
    return;
  }

  if (isDefloadCommand()) {
    requestStopBeforeProgramChange();
    loadBuiltinProgram();
    return;
  }

  if (isSaveCommand()) {
    requestStopBeforeProgramChange();
    saveProgram();
    return;
  }

  if (equalsToken_P(PSTR("LOAD"))) {
    requestStopBeforeProgramChange();
    loadProgram();
    return;
  }

  if (equalsToken_P(PSTR("ONCE"))) {
    requestPlayOnce();
```

```
      return;
    }

    if (equalsToken_P(PSTR("LOOP"))) {
      requestPlayLoop();
      return;
    }

    if (equalsToken_P(PSTR("STOP"))) {
      requestStopPlayback();
      return;
    }

    if (equalsToken_P(PSTR("STATE"))) {
      requestState();
      return;
    }

    if (isImmediateCommand()) {
      startImmediateCommand(lineEnd);
      return;
    }

    unknownInstruction();
  }

  bool isCommandWithOptionalArgument_P(PGM_P command, byte
commandLength) const {
    if (strncmp_P(token, command, commandLength) != 0) {
      return false;
    }

    return token[commandLength] == '\0'
      || token[commandLength] == ' '
      || token[commandLength] == '\t';
  }

  bool isSaveCommand() const {
    return isCommandWithOptionalArgument_P(PSTR("SAVE"), 4);
  }

  bool isDefloadCommand() const {
    return isCommandWithOptionalArgument_P(PSTR("DEFLOAD"), 7);
  }

  bool isImmediateCommand() const {
    return isCommandWithOptionalArgument_P(PSTR("IMMEDIATE"), 9);
  }

  void startImmediateCommand(bool lineEnd) {
    pendingFrame.clear();
```

```
  state = IMMEDIATE_INPUT;
  errorFlag = false;

  const char *p = token + 9;
  skipSpaces(p);

  if (*p != '\0') {
    moveTextToToken(p);
    parseImmediateInstruction(lineEnd);
    return;
  }

  if (lineEnd) {
    failImmediate(F("immediate frame is empty"));
  }
}

void finishUpload() {
  if (memory.sequenceOpen) {
    fail(F("END UPLOAD before endrepeat"));
    return;
  }

  unsigned long uploadTick = millis() / 100;

  programManager.setUploadedName(uploadTick);

  out.print(F("Uploaded."));
  out.println(uploadTick);

  memory.printStats(out);

  state = WAIT_START;
  pendingFrame.clear();
}

void abortUploadAndRequestStop() {
  memory.clear();
  pendingFrame.clear();

  state = WAIT_START;
  playbackRequest = PLAY_STOP;
  inComment = false;
  slashPending = false;

  out.println(F("upload stopped"));
  out.println(F("stop playback requested"));
}

void printInfo() {
  if (memory.sequenceCount == 0) {
```

```
      out.println(F("No program loaded"));
      return;
    }

    out.print(F("Program: "));
    out.println(programManager.getProgramName());

    memory.printStats(out);
  }

  void printProgram() {
    if (memory.sequenceCount == 0) {
      out.println(F("No program loaded"));
      return;
    }

    out.println(F("START UPLOAD"));

    for (byte i = 0; i < memory.sequenceCount; i++) {
      const RoboSequence* sequence = memory.getSequence(i);

      if (sequence == 0) {
        continue;
      }

      out.print(F("repeat:"));
      out.print(sequence->repeats);
      out.println(F(";"));

      for (byte j = 0; j < sequence->frameCount; j++) {
        const RoboActionFrame* frame = sequence->getFrame(j);

        if (frame != 0) {
          printFrameOneLine(*frame, true);
        }
      }

      out.println(F("endrepeat;"));
    }

    out.println(F("END UPLOAD"));
  }

  void    printFrameOneLine(const    RoboActionFrame&    frame,    bool
includeDelay) {
    bool printed = false;

    printed = printAxisIfEnabled('r', frame.Rotate, printed);
    printed = printAxisIfEnabled('c', frame.Claw, printed);
    printed = printAxisIfEnabled('e', frame.Elevation, printed);
    printed = printAxisIfEnabled('s', frame.Stretch, printed);
```

```
    if (includeDelay) {
      if (printed) {
        out.print(F(" "));
      }

      out.print(F("delay:"));
      printDelayTenths(frame.delay10);
      out.print(F(";"));
    }

    out.println();
  }

  bool   printAxisIfEnabled(char   axis,   const   RoboAxisDestination&
destination, bool printedBefore) {
    if (!destination.isEnabled()) {
      return printedBefore;
    }

    if (printedBefore) {
      out.print(F(" "));
    }

    out.print(axis);
    out.print(F(":"));
    out.print(destination.angle);

    if (destination.slowness != RoboAxisDestination::DEFAULT_SLOWNESS)
{
      out.print(F("/"));
      out.print(destination.slowness);
    }

    out.print(F(";"));

    return true;
  }

  void printDelayTenths(byte delay10) {
    byte whole = delay10 / 10;
    byte tenth = delay10 % 10;

    out.print(whole);

    if (tenth > 0) {
      out.print(F("."));
      out.print(tenth);
    }
  }
```

```
  void clearProgram() {
    memory.clear();
    programManager.clearProgramName();
    pendingFrame.clear();

    state = WAIT_START;
    inComment = false;
    slashPending = false;
    errorFlag = false;

    out.println(F("program cleared"));
  }

  void printBuiltinPrograms() {
    programManager.printBuiltinList(out);
  }

  void loadBuiltinProgram() {
    const char *p = token + 7;

    skipSpaces(p);

    int index = parseUnsigned(p);

    skipSpaces(p);

    if (index < 1 || *p != '\0') {
      out.println(F("cannot defload: bad index"));
      return;
    }

    RoboProgramManager::LoadStatus status =
programManager.loadBuiltin((byte)index);

    if (status == RoboProgramManager::LOAD_OK) {
      out.print(F("builtin loaded \""));
      out.print(programManager.getProgramName());
      out.println(F("\""));

      memory.printStats(out);
      return;
    }

    printLoadError(status, true);
  }

  void saveProgram() {
    const char *nameSource = token + 4;

    RoboProgramManager::SaveStatus status =
programManager.save(nameSource);
```

```
    if (status == RoboProgramManager::SAVE_OK) {
      out.print(F("program saved \""));
      out.print(programManager.getProgramName());
      out.println(F("\""));
      return;
    }

    if (status == RoboProgramManager::SAVE_EMPTY_PROGRAM) {
      out.println(F("cannot save: no program loaded"));
      return;
    }

    if (status == RoboProgramManager::SAVE_NO_SPACE) {
      out.println(F("cannot save: not enough EEPROM space"));
      return;
    }
  }

  void loadProgram() {
    RoboProgramManager::LoadStatus status = programManager.load();

    if (status == RoboProgramManager::LOAD_OK) {
      out.print(F("program loaded \""));
      out.print(programManager.getProgramName());
      out.println(F("\""));

      memory.printStats(out);
      return;
    }

    printLoadError(status, false);
  }

  void   printLoadError(RoboProgramManager::LoadStatus   status,   bool
isBuiltin) {
    if (status == RoboProgramManager::LOAD_BAD_INDEX) {
      out.println(F("cannot defload: index out of range"));
      return;
    }

    if (status == RoboProgramManager::LOAD_BAD_DESCRIPTOR) {
      if (isBuiltin) {
        out.println(F("cannot load builtin: invalid descriptor"));
      } else {
        out.println(F("cannot load: invalid EEPROM descriptor"));
        programManager.printEepromDescriptor(out);
      }

      return;
    }
```

```
    if (status == RoboProgramManager::LOAD_MAJOR_MISMATCH) {
      if (isBuiltin) {
        out.print(F("cannot  load  builtin:  major  version  mismatch.
Stored="));
      } else {
        out.print(F("cannot load: major version mismatch. Stored="));
      }

      out.print(programManager.getLastMajorVersion());
      out.print(F(", current="));
      out.println(programManager.getCurrentMajorVersion());
      return;
    }

    if (status == RoboProgramManager::LOAD_BAD_NAME) {
      if (isBuiltin) {
        out.println(F("cannot load builtin: invalid program name"));
      } else {
        out.println(F("cannot load: invalid program name"));
      }

      return;
    }

    if (status == RoboProgramManager::LOAD_BAD_PROGRAM) {
      if (isBuiltin) {
        out.println(F("cannot load builtin: program data error"));
      } else {
        out.println(F("cannot load: program data error"));
      }

      return;
    }
  }

  void requestPlayOnce() {
    if (memory.sequenceCount == 0) {
      out.println(F("No program loaded"));
      return;
    }

    playbackRequest = PLAY_ONCE;
    out.println(F("play once requested"));
  }

  void requestPlayLoop() {
    if (memory.sequenceCount == 0) {
      out.println(F("No program loaded"));
      return;
    }
```

```
    playbackRequest = PLAY_LOOP;
    out.println(F("loop playback requested"));
  }

  void requestStopPlayback() {
    playbackRequest = PLAY_STOP;
    out.println(F("stop playback requested"));
  }

  void requestState() {
    playbackRequest = PLAY_STATE;
  }

  void requestImmediate() {
    immediateFrame = pendingFrame;
    playbackRequest = PLAY_IMMEDIATE;
    pendingFrame.clear();
    state = WAIT_START;

    out.println(F("immediate requested"));
  }

  void parseInstruction() {
    if (startsWith_P(PSTR("repeat:"))) {
      parseRepeat();
      return;
    }

    if (equalsToken_P(PSTR("endrepeat"))) {
      parseEndRepeat();
      return;
    }

    if (startsWith_P(PSTR("delay:"))) {
      parseDelay();
      return;
    }

    if (token[0] == 'r' || token[0] == 'c' || token[0] == 'e' || token[0]
== 's') {
      parseAxis();
      return;
    }

    fail(F("unknown instruction"));
  }

  void parseImmediateInstruction(bool lineEnd) {
    if                (startsWith_P(PSTR("repeat:"))              ||
equalsToken_P(PSTR("endrepeat"))) {
```

```
      failImmediate(F("repeat is not allowed in immediate mode"));
      return;
    }

    if (startsWith_P(PSTR("delay:"))) {
      parseImmediateDelay();
      return;
    }

    if (token[0] == 'r' || token[0] == 'c' || token[0] == 'e' || token[0]
== 's') {
      if (!parseAxisTokenInto(pendingFrame, false)) {
        return;
      }

      if (lineEnd) {
        finishImmediateWithoutDelay();
      }

      return;
    }

    failImmediate(F("unknown immediate instruction"));
  }

  void finishImmediateWithoutDelay() {
    if (state != IMMEDIATE_INPUT) {
      return;
    }

    if (!pendingFrame.hasAnyAxis()) {
      failImmediate(F("immediate frame is empty"));
      return;
    }

    pendingFrame.delay10 = 0;
    requestImmediate();
  }

  void parseImmediateDelay() {
    if (!pendingFrame.hasAnyAxis()) {
      failImmediate(F("delay without axis commands"));
      return;
    }

    const char *p = token + 6;
    skipSpaces(p);

    int delay10 = parseDelayTenths(p);

    skipSpaces(p);
```

```
  if (*p != '\0') {
    failImmediate(F("bad delay syntax"));
    return;
  }

  if (delay10 < 0 || delay10 > 255) {
    failImmediate(F("delay must be 0..25.5 seconds"));
    return;
  }

  pendingFrame.delay10 = (byte)delay10;
  requestImmediate();
}

void parseRepeat() {
  if (memory.sequenceOpen) {
    fail(F("nested repeat is not allowed"));
    return;
  }

  const char *p = token + 7;
  skipSpaces(p);

  int repeatCount = parseUnsigned(p);

  skipSpaces(p);

  if (*p != '\0') {
    fail(F("bad repeat syntax"));
    return;
  }

  if (repeatCount < 1 || repeatCount > 255) {
    fail(F("repeat must be 1..255"));
    return;
  }

  if (!memory.startSequence((byte)repeatCount)) {
    fail(F("too many sequences"));
    return;
  }

  pendingFrame.clear();
}

void parseEndRepeat() {
  if (!memory.sequenceOpen) {
    fail(F("endrepeat without repeat"));
    return;
  }
```

```
  if (pendingFrame.hasAnyAxis()) {
    fail(F("frame has no delay before endrepeat"));
    return;
  }

  if (!memory.finishSequence()) {
    fail(F("empty sequence"));
    return;
  }
}

void parseDelay() {
  if (!memory.sequenceOpen) {
    fail(F("delay outside repeat"));
    return;
  }

  if (!pendingFrame.hasAnyAxis()) {
    fail(F("delay without axis commands"));
    return;
  }

  const char *p = token + 6;
  skipSpaces(p);

  int delay10 = parseDelayTenths(p);

  skipSpaces(p);

  if (*p != '\0') {
    fail(F("bad delay syntax"));
    return;
  }

  if (delay10 < 0 || delay10 > 255) {
    fail(F("delay must be 0..25.5 seconds"));
    return;
  }

  pendingFrame.delay10 = (byte)delay10;

  if (!memory.addFrame(pendingFrame)) {
    fail(F("too many frames"));
    return;
  }

  pendingFrame.clear();
}

void parseAxis() {
```

```
  if (!memory.sequenceOpen) {
    fail(F("axis command outside repeat"));
    return;
  }

  parseAxisTokenInto(pendingFrame, true);
}

bool parseAxisTokenInto(RoboActionFrame& frame, bool uploadMode) {
  char axis = token[0];
  const char *p = token + 1;

  skipSpaces(p);

  if (*p != ':') {
    failForMode(F("bad axis syntax"), uploadMode);
    return false;
  }

  p++;
  skipSpaces(p);

  int angle = parseUnsigned(p);

  if (angle < 0 || angle > 180) {
    failForMode(F("angle must be 0..180"), uploadMode);
    return false;
  }

  byte slowness = RoboAxisDestination::DEFAULT_SLOWNESS;

  skipSpaces(p);

  if (*p == '/') {
    p++;
    skipSpaces(p);

    int parsedSlowness = parseUnsigned(p);

    if (
      parsedSlowness < RoboAxisDestination::MIN_SLOWNESS
      || parsedSlowness > RoboAxisDestination::MAX_SLOWNESS
    ) {
      failForMode(F("slowness is out of range"), uploadMode);
      return false;
    }

    slowness = (byte)parsedSlowness;
  }

  skipSpaces(p);
```

```
    if (*p != '\0') {
      failForMode(F("bad axis tail"), uploadMode);
      return false;
    }

    if (axis == 'r') {
      frame.setRotate((byte)angle, slowness);
      return true;
    }

    if (axis == 'c') {
      frame.setClaw((byte)angle, slowness);
      return true;
    }

    if (axis == 'e') {
      frame.setElevation((byte)angle, slowness);
      return true;
    }

    if (axis == 's') {
      frame.setStretch((byte)angle, slowness);
      return true;
    }

    failForMode(F("unknown axis"), uploadMode);
    return false;
  }

  void failForMode(const __FlashStringHelper *message, bool uploadMode)
{
    if (uploadMode) {
      fail(message);
    } else {
      failImmediate(message);
    }
  }

  int parseUnsigned(const char *&p) {
    if (*p < '0' || *p > '9') {
      return -1;
    }

    int value = 0;

    while (*p >= '0' && *p <= '9') {
      value = value * 10 + (*p - '0');

      if (value > 999) {
        return -1;
```

```
    }

    p++;
  }

  return value;
}

int parseDelayTenths(const char *&p) {
  int whole = parseUnsigned(p);

  if (whole < 0) {
    return -1;
  }

  int tenth = 0;

  if (*p == '.') {
    p++;

    if (*p < '0' || *p > '9') {
      return -1;
    }

    tenth = *p - '0';
    p++;

    if (*p >= '0' && *p <= '9') {
      return -1;
    }
  }

  return whole * 10 + tenth;
}

void skipSpaces(const char *&p) {
  while (*p == ' ' || *p == '\t') {
    p++;
  }
}

void trimToken() {
  byte start = 0;

  while (start < tokenLen && isSpaceChar(token[start])) {
    start++;
  }

  byte end = tokenLen;

  while (end > start && isSpaceChar(token[end - 1])) {
```

```
    end--;
  }

  byte newLen = end - start;

  if (start > 0) {
    for (byte i = 0; i < newLen; i++) {
      token[i] = token[start + i];
    }
  }

  token[newLen] = '\0';
  tokenLen = newLen;
}

void moveTextToToken(const char *source) {
  byte index = 0;

  while (source[index] != '\0' && index < TOKEN_SIZE - 1) {
    token[index] = source[index];
    index++;
  }

  token[index] = '\0';
  tokenLen = index;
  trimToken();
}

bool isSpaceChar(char ch) const {
  return ch == ' ' || ch == '\t' || ch == '\n' || ch == '\r';
}

bool equalsToken_P(PGM_P text) const {
  byte i = 0;

  while (token[i] != '\0') {
    char expected = (char)pgm_read_byte(text + i);

    if (expected == '\0') {
      return false;
    }

    if (token[i] != expected) {
      return false;
    }

    i++;
  }

  return pgm_read_byte(text + i) == '\0';
}
```

```
  bool startsWith_P(PGM_P text) const {
    byte i = 0;

    while (true) {
      char expected = (char)pgm_read_byte(text + i);

      if (expected == '\0') {
        return true;
      }

      if (token[i] != expected) {
        return false;
      }

      i++;
    }
  }

  void unknownInstruction() {
    out.print(F("unknown instruction \""));
    out.print(token);
    out.println(F("\""));

    printAvailableInstructions();

    resetInputState();
  }

  void printAvailableInstructions() {
    out.println(F("Available instructions:"));

    out.println(F("1. START UPLOAD"));
    out.println(F("   START UPLOAD"));
    out.println(F("   <CODE>"));
    out.println(F("   END UPLOAD"));

    out.println(F("2. INFO        - show loaded program statistics"));
    out.println(F("3. PRINT      - print loaded program as START UPLOAD
block"));
    out.println(F("4. CLEAR       - clear loaded program"));
    out.println(F("5. SAVE <name> - save loaded program to EEPROM"));
    out.println(F("6. LOAD        - load program from EEPROM"));
    out.println(F("7. BUILTIN     - show builtin program list"));
    out.println(F("8. DEFLOAD xx  - load builtin program by number"));
    out.println(F("9. ONCE        - set flag to run program once"));
    out.println(F("10. LOOP       - set flag to run program in loop"));
    out.println(F("11. STOP       - set flag to stop playback or abort
upload"));
    out.println(F("12. STATE      - print current axis angles"));
```

```
    out.println(F("13. IMMEDIATE <frame> - execute one frame without
repeat"));
  }

  void fail(const __FlashStringHelper *message) {
    errorFlag = true;

    out.print(F("ERROR line "));
    out.print(lineNumber);
    out.print(F(": "));
    out.print(message);

    if (tokenLen > 0) {
      out.print(F(" near \""));
      out.print(token);
      out.print(F("\""));
    }

    out.println();

    if (state == UPLOADING) {
      memory.clear();
    }

    resetInputState();
  }

  void failImmediate(const __FlashStringHelper *message) {
    errorFlag = true;

    out.print(F("ERROR line "));
    out.print(lineNumber);
    out.print(F(": "));
    out.print(message);

    if (tokenLen > 0) {
      out.print(F(" near \""));
      out.print(token);
      out.print(F("\""));
    }

    out.println();

    pendingFrame.clear();
    resetInputState();
  }

  void failNearChar(const __FlashStringHelper *message, char ch) {
    errorFlag = true;

    out.print(F("ERROR line "));
```

```
    out.print(lineNumber);
    out.print(F(": "));
    out.print(message);
    out.print(F(" near char \""));
    out.print(ch);
    out.println(F("\""));

    if (state == UPLOADING) {
      memory.clear();
    }

    resetInputState();
  }
};

#endif
```

ДОДАТОК Д – Модуль Playback.h

```
ываыва
#ifndef PLAYBACK_H
#define PLAYBACK_H

#include <Arduino.h>
#include "Robo.h"
#include "ArmServo.h"

class Playback
{
private:
  enum PlaybackMode
  {
    MODE_ONCE,
    MODE_LOOP
  };

  RoboProgramBuffer* program;
  RoboArm* arm;

  const RoboSequence* currentSequence;

  byte sequenceIndex;
  byte frameIndex;
  byte repeatIndex;

  bool active;
  bool waitingDelay;
  bool finishPending;
  PlaybackMode mode;

  unsigned long delayStart;

public:
  Playback()
  {
    program = 0;
    arm = 0;

    currentSequence = 0;

    sequenceIndex = 0;
    frameIndex = 0;
    repeatIndex = 0;
```

```
  active = false;
  waitingDelay = false;
  finishPending = false;
  mode = MODE_ONCE;

  delayStart = 0;
}

// Підключити програму та руку
void begin(RoboProgramBuffer& programBuffer, RoboArm& roboArm)
{
  program = &programBuffer;
  arm = &roboArm;
}

// Перевірити, чи зараз іде програвання
bool isActive()
{
  return active;
}

// Почати одноразове програвання всієї програми
void startOnce()
{
  mode = MODE_ONCE;
  startFromBeginning();
}

// Почати циклічне програвання всієї програми
void startLoop()
{
  mode = MODE_LOOP;
  startFromBeginning();
}

// Сумісність зі старою назвою методу
void start()
{
  startOnce();
}

// Зупинити програвання
void stop()
{
  active = false;
  waitingDelay = false;
  finishPending = false;
  currentSequence = 0;

  if (arm != 0)
  {
```

```
      arm->stop();
    }
  }

  // Оновити програвання програми
  void update()
  {
    if (!active)
    {
      return;
    }

    if (finishPending)
    {
      stop();
      return;
    }

    if (currentSequence == 0)
    {
      stop();
      return;
    }

    const      RoboActionFrame*      frame      =      currentSequence-
>getFrame(frameIndex);

    if (frame == 0)
    {
      finishCurrentRepeat();
      return;
    }

    if (waitingDelay)
    {
      updateDelay(frame);
      return;
    }

    bool frameDone = executeFrame(frame);

    if (frameDone)
    {
      waitingDelay = true;
      delayStart = millis();
    }
  }

private:
  // Почати програвання з першої секвенції
  void startFromBeginning()
```

```
  {
    if (program == 0 || arm == 0)
    {
      return;
    }

    if (program->sequenceCount == 0)
    {
      return;
    }

    sequenceIndex = 0;
    currentSequence = program->getSequence(sequenceIndex);

    if (currentSequence == 0)
    {
      return;
    }

    if (currentSequence->isEmpty())
    {
      return;
    }

    arm->stop();

    frameIndex = 0;
    repeatIndex = 0;
    waitingDelay = false;
    finishPending = false;
    delayStart = 0;
    active = true;
  }

  // Виконати один кадр програми
  bool executeFrame(const RoboActionFrame* frame)
  {
    bool allDone = true;

    if (frame->Rotate.isEnabled())
    {
      allDone    =    arm->Rotate.moveTo(frame->Rotate.angle,    frame-
>Rotate.slowness) && allDone;
    }

    if (frame->Elevation.isEnabled())
    {
      allDone  =  arm->Elevation.moveTo(frame->Elevation.angle,  frame-
>Elevation.slowness) && allDone;
    }
```

```
    if (frame->Stretch.isEnabled())
    {
      allDone   =   arm->Stretch.moveTo(frame->Stretch.angle,   frame-
>Stretch.slowness) && allDone;
    }

    if (frame->Claw.isEnabled())
    {
      allDone     =     arm->Claw.moveTo(frame->Claw.angle,     frame-
>Claw.slowness) && allDone;
    }

    return allDone;
  }

  // Обробити затримку після кадру
  void updateDelay(const RoboActionFrame* frame)
  {
    unsigned int delayMs = frame->getDelayMs();

    if (millis() - delayStart < delayMs)
    {
      return;
    }

    waitingDelay = false;
    frameIndex++;

    if (frameIndex >= currentSequence->frameCount)
    {
      finishCurrentRepeat();
    }
  }

  // Завершити поточний повтор секвенції
  void finishCurrentRepeat()
  {
    repeatIndex++;

    if (repeatIndex < currentSequence->repeats)
    {
      frameIndex = 0;
      waitingDelay = false;
      return;
    }

    startNextSequence();
  }

  // Перейти до наступної секвенції програми
  void startNextSequence()
```

```
  {
    sequenceIndex++;

    if (sequenceIndex >= program->sequenceCount)
    {
      if (mode == MODE_LOOP)
      {
        startFromBeginning();
        return;
      }

      finishPending = true;
      return;
    }

    currentSequence = program->getSequence(sequenceIndex);

    if (currentSequence == 0)
    {
      stop();
      return;
    }

    if (currentSequence->isEmpty())
    {
      startNextSequence();
      return;
    }

    frameIndex = 0;
    repeatIndex = 0;
    waitingDelay = false;
    delayStart = 0;
  }
};

#endif
ываыва
```

ДОДАТОК Е – модуль RoboProgramManager.h

```
#ifndef ROBO_PROGRAM_MANAGER_H
#define ROBO_PROGRAM_MANAGER_H

#include <Arduino.h>
#include <EEPROM.h>
#include <avr/pgmspace.h>
#include "Robo.h"
#include "RoboVersion.h"
#include "DefaultProg.h"

class RoboProgramManager {
public:
  static const int EEPROM_START_ADDRESS = 512;

  static const byte CURRENT_MAJOR = ROBO_VERSION_MAJOR;
  static const byte CURRENT_MINOR = ROBO_VERSION_MINOR;

  static const byte MAX_NAME_LENGTH = 25;
  static const byte DESCRIPTOR_LENGTH = ROBO_DESCRIPTOR_LENGTH;
  static const byte HEADER_LENGTH = DESCRIPTOR_LENGTH + 1 +
MAX_NAME_LENGTH;

  enum SaveStatus {
    SAVE_OK,
    SAVE_EMPTY_PROGRAM,
    SAVE_NO_SPACE
  };

  enum LoadStatus {
    LOAD_OK,
    LOAD_BAD_DESCRIPTOR,
    LOAD_MAJOR_MISMATCH,
    LOAD_BAD_NAME,
    LOAD_BAD_PROGRAM,
    LOAD_BAD_INDEX
  };

private:
  RoboProgramBuffer &program;

  char programName[MAX_NAME_LENGTH + 1];

  byte lastMajorVersion;
  byte lastMinorVersion;
```

```
public:
  RoboProgramManager(RoboProgramBuffer &targetProgram)
    : program(targetProgram)
  {
    clearProgramName();

    lastMajorVersion = 0;
    lastMinorVersion = 0;
  }

  const char* getProgramName() const {
    return programName;
  }

  byte getCurrentMajorVersion() const {
    return CURRENT_MAJOR;
  }

  byte getCurrentMinorVersion() const {
    return CURRENT_MINOR;
  }

  byte getLastMajorVersion() const {
    return lastMajorVersion;
  }

  byte getLastMinorVersion() const {
    return lastMinorVersion;
  }

  byte getBuiltinProgramCount() const {
    return BUILTIN_PROGRAM_COUNT;
  }

  void clearProgramName() {
    copyFromProgmem(programName, PSTR("Unnamed"));
  }

  void setUploadedName(unsigned long tick100) {
    copyFromProgmem(programName, PSTR("Uploaded."));
    appendUnsignedLong(programName, tick100);
  }

  void printEepromDescriptor(Stream &out) {
    out.print(F("EEPROM descriptor: \""));

    for (byte i = 0; i < DESCRIPTOR_LENGTH; i++) {
      int address = EEPROM_START_ADDRESS + i;

      if (address >= EEPROM.length()) {
```

```
      out.print(F("<out>"));
      break;
    }

    byte value = EEPROM.read(address);

    if (value >= 32 && value <= 126) {
      out.print((char)value);
    } else {
      out.print(F("\\x"));

      if (value < 16) {
        out.print('0');
      }

      out.print(value, HEX);
    }
  }

  out.println(F("\""));
}

SaveStatus save(const char *nameSource) {
  if (program.sequenceCount == 0) {
    return SAVE_EMPTY_PROGRAM;
  }

  char preparedName[MAX_NAME_LENGTH + 1];

  buildProgramName(nameSource, preparedName);

  int requiredSize = getRequiredEepromSize();

  if (EEPROM_START_ADDRESS + requiredSize > EEPROM.length()) {
    return SAVE_NO_SPACE;
  }

  int address = EEPROM_START_ADDRESS;

  writeHeaderToEeprom(address, preparedName);
  program.writeToEeprom(address);

  copyName(preparedName, programName);

  return SAVE_OK;
}

LoadStatus load() {
  int address = EEPROM_START_ADDRESS;

  LoadStatus headerStatus = readHeaderFromEeprom(address);
```

```
  if (headerStatus != LOAD_OK) {
    return headerStatus;
  }

  if (!program.readFromEeprom(address)) {
    return LOAD_BAD_PROGRAM;
  }

  return LOAD_OK;
}

void printBuiltinList(Stream &out) {
  out.println(F("Builtin programs:"));

  for (byte i = 0; i < BUILTIN_PROGRAM_COUNT; i++) {
    const byte *data = getBuiltinPointer(i);

    char name[MAX_NAME_LENGTH + 1];
    byte major = 0;
    byte minor = 0;

    bool ok = readBuiltinHeaderPreview(data, name, major, minor);

    out.print(F("DEFLOAD "));
    out.print(i + 1);
    out.print(F(" - "));

    if (!ok) {
      out.println(F("<bad descriptor>"));
      continue;
    }

    out.print(name);
    out.print(F(" ["));
    printTwoDigits(out, major);
    out.print(F("."));
    printTwoDigits(out, minor);
    out.println(F("]"));
  }
}

LoadStatus loadBuiltin(byte userIndex) {
  if (userIndex == 0 || userIndex > BUILTIN_PROGRAM_COUNT) {
    return LOAD_BAD_INDEX;
  }

  byte realIndex = userIndex - 1;
  const byte *data = getBuiltinPointer(realIndex);

  int address = 0;
```

```
    LoadStatus headerStatus = readHeaderFromProgmem(data, address);

    if (headerStatus != LOAD_OK) {
      return headerStatus;
    }

    if (!program.readFromProgmem(data, address)) {
      return LOAD_BAD_PROGRAM;
    }

    return LOAD_OK;
  }

private:
  const byte* getBuiltinPointer(byte index) const {
    return (const byte*)pgm_read_word(&BUILTIN_PROGRAMS[index]);
  }

  void copyFromProgmem(char *target, PGM_P source) {
    strncpy_P(target, source, MAX_NAME_LENGTH);
    target[MAX_NAME_LENGTH] = '\0';
  }

  void buildProgramName(const char *source, char *target) {
    const char *p = source;

    if (p != 0) {
      while (*p == ' ' || *p == '\t') {
        p++;
      }
    }

    if (p == 0 || *p == '\0') {
      buildDefaultName(target);
      return;
    }

    byte index = 0;

    while (*p != '\0' && index < MAX_NAME_LENGTH) {
      target[index] = *p;
      index++;
      p++;
    }

    target[index] = '\0';
  }

  void buildDefaultName(char *target) {
    copyFromProgmem(target, PSTR("Unnamed."));
```

```
  appendUnsignedLong(target, millis() / 100);
}

void appendUnsignedLong(char *target, unsigned long value) {
  char numberBuffer[11];

  ultoa(value, numberBuffer, 10);

  byte targetLength = getStringLength(target);
  byte numberIndex = 0;

  while (
    numberBuffer[numberIndex] != '\0'
    && targetLength < MAX_NAME_LENGTH
  ) {
    target[targetLength] = numberBuffer[numberIndex];
    targetLength++;
    numberIndex++;
  }

  target[targetLength] = '\0';
}

byte getStringLength(const char *text) const {
  byte length = 0;

  while (text[length] != '\0' && length < MAX_NAME_LENGTH) {
    length++;
  }

  return length;
}

void copyName(const char *source, char *target) {
  byte index = 0;

  while (source[index] != '\0' && index < MAX_NAME_LENGTH) {
    target[index] = source[index];
    index++;
  }

  target[index] = '\0';
}

int getRequiredEepromSize() const {
  int size = 0;

  size += HEADER_LENGTH;
  size += getProgramStorageSize();

  return size;
```

```
  }

  int getProgramStorageSize() const {
    int size = 0;

    // Кількість секвенцій.
    size += 1;

    for (byte sequenceIndex = 0; sequenceIndex < program.sequenceCount;
sequenceIndex++) {
      const             RoboSequence             *sequence             =
program.getSequence(sequenceIndex);

      if (sequence == 0) {
        continue;
      }

      // repeats + frameCount.
      size += 2;

      for (byte frameIndex = 0; frameIndex < sequence->frameCount;
frameIndex++) {
        const RoboActionFrame *frame = sequence->getFrame(frameIndex);

        if (frame != 0) {
          size += getFrameStorageSize(*frame);
        }
      }
    }

    return size;
  }

  int getFrameStorageSize(const RoboActionFrame &frame) const {
    int size = 0;

    // flags.
    size += 1;

    size += getAxisStorageSize(frame.Stretch);
    size += getAxisStorageSize(frame.Elevation);
    size += getAxisStorageSize(frame.Claw);
    size += getAxisStorageSize(frame.Rotate);

    // delay10.
    size += 1;

    return size;
  }

  byte getAxisStorageSize(const RoboAxisDestination &axis) const {
```

```
  byte axisFlags = axis.getFlags();
  byte size = 0;

  if ((axisFlags & RoboAxisDestination::FLAG_ANGLE) != 0) {
    size++;
  }

  if ((axisFlags & RoboAxisDestination::FLAG_SLOWNESS) != 0) {
    size++;
  }

  return size;
}

void writeHeaderToEeprom(int &address, const char *name) {
  writeDescriptorToEeprom(address);
  writeFixedNameToEeprom(address, name);
}

void writeDescriptorToEeprom(int &address) {
  EEPROM.update(address, ROBO_DESCRIPTOR_PREFIX_0);
  address++;

  EEPROM.update(address, ROBO_DESCRIPTOR_PREFIX_1);
  address++;

  EEPROM.update(address, ROBO_DESCRIPTOR_PREFIX_2);
  address++;

  EEPROM.update(address, ROBO_DESCRIPTOR_PREFIX_3);
  address++;

  EEPROM.update(address, ROBO_DESCRIPTOR_PREFIX_4);
  address++;

  EEPROM.update(address, ROBO_DESCRIPTOR_PREFIX_5);
  address++;

  EEPROM.update(address, ROBO_DESCRIPTOR_PREFIX_6);
  address++;

  EEPROM.update(address, ROBO_VERSION_MAJOR_HI);
  address++;

  EEPROM.update(address, ROBO_VERSION_MAJOR_LOW);
  address++;

  EEPROM.update(address, '.');
  address++;

  EEPROM.update(address, ROBO_VERSION_MINOR_HI);
```

```
  address++;

  EEPROM.update(address, ROBO_VERSION_MINOR_LOW);
  address++;
}

void writeFixedNameToEeprom(int &address, const char *name) {
  byte length = getStringLength(name);

  EEPROM.update(address, length);
  address++;

  for (byte i = 0; i < MAX_NAME_LENGTH; i++) {
    if (i < length) {
      EEPROM.update(address, name[i]);
    } else {
      EEPROM.update(address, 0);
    }

    address++;
  }
}

LoadStatus readHeaderFromEeprom(int &address) {
  if (EEPROM.length() < EEPROM_START_ADDRESS + HEADER_LENGTH) {
    return LOAD_BAD_DESCRIPTOR;
  }

  if (!readDescriptorFromEeprom(address)) {
    return LOAD_BAD_DESCRIPTOR;
  }

  if (lastMajorVersion != CURRENT_MAJOR) {
    return LOAD_MAJOR_MISMATCH;
  }

  if (!readFixedNameFromEeprom(address, programName)) {
    return LOAD_BAD_NAME;
  }

  return LOAD_OK;
}

bool readDescriptorFromEeprom(int &address) {
  if (!readPrefixFromEeprom(address)) {
    return false;
  }

  int major = readTwoDigitsFromEeprom(address);

  if (major < 0) {
```

```
    return false;
  }

  if (EEPROM.read(address) != '.') {
    return false;
  }

  address++;

  int minor = readTwoDigitsFromEeprom(address);

  if (minor < 0) {
    return false;
  }

  lastMajorVersion = (byte)major;
  lastMinorVersion = (byte)minor;

  return true;
}

bool readPrefixFromEeprom(int &address) {
  if (address + 7 > EEPROM.length()) {
    return false;
  }

  if (EEPROM.read(address) != ROBO_DESCRIPTOR_PREFIX_0) {
    return false;
  }
  address++;

  if (EEPROM.read(address) != ROBO_DESCRIPTOR_PREFIX_1) {
    return false;
  }
  address++;

  if (EEPROM.read(address) != ROBO_DESCRIPTOR_PREFIX_2) {
    return false;
  }
  address++;

  if (EEPROM.read(address) != ROBO_DESCRIPTOR_PREFIX_3) {
    return false;
  }
  address++;

  if (EEPROM.read(address) != ROBO_DESCRIPTOR_PREFIX_4) {
    return false;
  }
  address++;
```

```
  if (EEPROM.read(address) != ROBO_DESCRIPTOR_PREFIX_5) {
    return false;
  }
  address++;

  if (EEPROM.read(address) != ROBO_DESCRIPTOR_PREFIX_6) {
    return false;
  }
  address++;

  return true;
}

int readTwoDigitsFromEeprom(int &address) {
  if (address + 1 >= EEPROM.length()) {
    return -1;
  }

  byte first = EEPROM.read(address);
  address++;

  byte second = EEPROM.read(address);
  address++;

  if (first < '0' || first > '9') {
    return -1;
  }

  if (second < '0' || second > '9') {
    return -1;
  }

  return (first - '0') * 10 + (second - '0');
}

bool readFixedNameFromEeprom(int &address, char *target) {
  if (address >= EEPROM.length()) {
    return false;
  }

  byte length = EEPROM.read(address);
  address++;

  if (length > MAX_NAME_LENGTH) {
    return false;
  }

  if (address + MAX_NAME_LENGTH > EEPROM.length()) {
    return false;
  }
```

```
  for (byte i = 0; i < MAX_NAME_LENGTH; i++) {
    byte value = EEPROM.read(address);
    address++;

    if (i < length) {
      target[i] = (char)value;
    }
  }

  target[length] = '\0';

  if (length == 0) {
    buildDefaultName(target);
  }

  return true;
}

LoadStatus readHeaderFromProgmem(const byte *data, int &address) {
  if (!readDescriptorFromProgmem(data, address)) {
    return LOAD_BAD_DESCRIPTOR;
  }

  if (lastMajorVersion != CURRENT_MAJOR) {
    return LOAD_MAJOR_MISMATCH;
  }

  if (!readFixedNameFromProgmem(data, address, programName)) {
    return LOAD_BAD_NAME;
  }

  return LOAD_OK;
}

bool readBuiltinHeaderPreview(
  const byte *data,
  char *name,
  byte &major,
  byte &minor
) {
  int address = 0;

  if (!readDescriptorFromProgmem(data, address)) {
    return false;
  }

  major = lastMajorVersion;
  minor = lastMinorVersion;

  if (!readFixedNameFromProgmem(data, address, name)) {
    return false;
```

```
    }

    return true;
  }

  bool readDescriptorFromProgmem(const byte *data, int &address) {
    if (!readPrefixFromProgmem(data, address)) {
      return false;
    }

    int major = readTwoDigitsFromProgmem(data, address);

    if (major < 0) {
      return false;
    }

    if (pgm_read_byte_near(data + address) != '.') {
      return false;
    }

    address++;

    int minor = readTwoDigitsFromProgmem(data, address);

    if (minor < 0) {
      return false;
    }

    lastMajorVersion = (byte)major;
    lastMinorVersion = (byte)minor;

    return true;
  }

  bool readPrefixFromProgmem(const byte *data, int &address) {
    if (pgm_read_byte_near(data + address) != ROBO_DESCRIPTOR_PREFIX_0)
{
      return false;
    }
    address++;

    if (pgm_read_byte_near(data + address) != ROBO_DESCRIPTOR_PREFIX_1)
{
      return false;
    }
    address++;

    if (pgm_read_byte_near(data + address) != ROBO_DESCRIPTOR_PREFIX_2)
{
      return false;
    }
```

```
    address++;

    if (pgm_read_byte_near(data + address) != ROBO_DESCRIPTOR_PREFIX_3)
{
      return false;
    }
    address++;

    if (pgm_read_byte_near(data + address) != ROBO_DESCRIPTOR_PREFIX_4)
{
      return false;
    }
    address++;

    if (pgm_read_byte_near(data + address) != ROBO_DESCRIPTOR_PREFIX_5)
{
      return false;
    }
    address++;

    if (pgm_read_byte_near(data + address) != ROBO_DESCRIPTOR_PREFIX_6)
{
      return false;
    }
    address++;

    return true;
  }

  int readTwoDigitsFromProgmem(const byte *data, int &address) {
    byte first = pgm_read_byte_near(data + address);
    address++;

    byte second = pgm_read_byte_near(data + address);
    address++;

    if (first < '0' || first > '9') {
      return -1;
    }

    if (second < '0' || second > '9') {
      return -1;
    }

    return (first - '0') * 10 + (second - '0');
  }

  bool readFixedNameFromProgmem(const byte *data, int &address, char
*target) {
    byte length = pgm_read_byte_near(data + address);
    address++;
```

```
    if (length > MAX_NAME_LENGTH) {
      return false;
    }

    for (byte i = 0; i < MAX_NAME_LENGTH; i++) {
      byte value = pgm_read_byte_near(data + address);
      address++;

      if (i < length) {
        target[i] = (char)value;
      }
    }

    target[length] = '\0';

    if (length == 0) {
      buildDefaultName(target);
    }

    return true;
  }

  void printTwoDigits(Stream &out, byte value) {
    out.print(value / 10);
    out.print(value % 10);
  }
};

#endif
```

ДОДАТОК Ж – мудуль RoboVersion.h

```
#ifndef ROBO_VERSION_H
#define ROBO_VERSION_H

// Цифри версії у вигляді символів для descriptor:
// RoboArm01.02
#define ROBO_VERSION_MAJOR_HI '0'
#define ROBO_VERSION_MAJOR_LOW '1'
#define ROBO_VERSION_MINOR_HI '0'
#define ROBO_VERSION_MINOR_LOW '2'

// Числові значення версії для перевірок у коді.
#define ROBO_VERSION_MAJOR \
  ((ROBO_VERSION_MAJOR_HI - '0') * 10 + (ROBO_VERSION_MAJOR_LOW - '0'))

#define ROBO_VERSION_MINOR \
  ((ROBO_VERSION_MINOR_HI - '0') * 10 + (ROBO_VERSION_MINOR_LOW - '0'))

// Початок descriptor.
#define ROBO_DESCRIPTOR_PREFIX_0 'R'
#define ROBO_DESCRIPTOR_PREFIX_1 'o'
#define ROBO_DESCRIPTOR_PREFIX_2 'b'
#define ROBO_DESCRIPTOR_PREFIX_3 'o'
#define ROBO_DESCRIPTOR_PREFIX_4 'A'
#define ROBO_DESCRIPTOR_PREFIX_5 'r'
#define ROBO_DESCRIPTOR_PREFIX_6 'm'

// Повний descriptor у вигляді байтів.
// Його можна вставляти прямо в PROGMEM-масив.
#define ROBO_DESCRIPTOR_BYTES \
  ROBO_DESCRIPTOR_PREFIX_0, \
  ROBO_DESCRIPTOR_PREFIX_1, \
  ROBO_DESCRIPTOR_PREFIX_2, \
  ROBO_DESCRIPTOR_PREFIX_3, \
  ROBO_DESCRIPTOR_PREFIX_4, \
  ROBO_DESCRIPTOR_PREFIX_5, \
  ROBO_DESCRIPTOR_PREFIX_6, \
  ROBO_VERSION_MAJOR_HI, \
  ROBO_VERSION_MAJOR_LOW, \
  '.', \
  ROBO_VERSION_MINOR_HI, \
  ROBO_VERSION_MINOR_LOW

#define ROBO_DESCRIPTOR_LENGTH 12

#endif
```

ДОДАТОК З – модуль DefaultProg.h

```
#ifndef DEFAULT_PROG_H
#define DEFAULT_PROG_H

#include <Arduino.h>
#include <avr/pgmspace.h>
#include "RoboVersion.h"

// Формат вбудованої програми:
//
// descriptor: 12 байт
// nameLength: 1 байт
// nameBuffer: 25 байт
// programData: поточний формат RoboProgramBuffer
//
// descriptor:
// RoboArmXX.YY
//
// programData:
// sequenceCount
// sequence: repeats, frameCount
// frame: flags, axis data..., delay10
//
// Порядок осей у кадрі:
// Stretch, Elevation, Claw, Rotate
//
// Біти flags:
// 7 6 5 4 3 2 1 0
// S s E e C c R r
//
// Велика літера = є angle.
// Мала літера = є slowness.
//
// delay10 зберігає затримку в десятих частках секунди:
// 5 = 0.5 секунди
// 0 = 0 секунд

const byte BUILTIN_PROGRAM_1[] PROGMEM = {
  ROBO_DESCRIPTOR_BYTES,

  // Ім'я: Default pickup
  14,
  'D', 'e', 'f', 'a', 'u', 'l', 't', ' ', 'p', 'i', 'c', 'k', 'u', 'p',
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

  // Програма
```

```
  6,              // Кількість секвенцій

  // Секвенція 1
  1,              // Кількість повторів
  1,              // Кількість кадрів

  0b10101111,     // Кадр 1: Stretch angle, Elevation angle, Claw angle
+ slowness, Rotate angle + slowness
  120,            // Stretch angle
  90,             // Elevation angle
  0,              // Claw angle
  1,              // Claw slowness
  0,              // Rotate angle
  20,             // Rotate slowness
  5,              // delay 5 * 100 мс = 0.5 секунди

  // Секвенція 2
  2,              // Кількість повторів
  2,              // Кількість кадрів

  0b00001100,     // Кадр 1: Claw angle + slowness
  90,             // Claw angle
  1,              // Claw slowness
  0,              // delay 0 * 100 мс

  0b00001100,     // Кадр 2: Claw angle + slowness
  0,              // Claw angle
  1,              // Claw slowness
  5,              // delay 5 * 100 мс = 0.5 секунди

  // Секвенція 3
  1,              // Кількість повторів
  1,              // Кількість кадрів

  0b10101111,     // Кадр 1: Stretch angle, Elevation angle, Claw angle
+ slowness, Rotate angle + slowness
  180,            // Stretch angle
  0,              // Elevation angle
  0,              // Claw angle
  1,              // Claw slowness
  90,             // Rotate angle
  20,             // Rotate slowness
  5,              // delay 5 * 100 мс = 0.5 секунди

  // Секвенція 4
  2,              // Кількість повторів
  2,              // Кількість кадрів

  0b00001100,     // Кадр 1: Claw angle + slowness
  90,             // Claw angle
  1,              // Claw slowness
```

```
  0,              // delay 0 * 100 мс

  0b00001100,     // Кадр 2: Claw angle + slowness
  0,              // Claw angle
  1,              // Claw slowness
  5,              // delay 5 * 100 мс = 0.5 секунди

  // Секвенція 5
  1,              // Кількість повторів
  1,              // Кількість кадрів

  0b10101111,     // Кадр 1: Stretch angle, Elevation angle, Claw angle
+ slowness, Rotate angle + slowness
  120,            // Stretch angle
  90,             // Elevation angle
  0,              // Claw angle
  1,              // Claw slowness
  180,            // Rotate angle
  5,              // Rotate slowness
  5,              // delay 5 * 100 мс = 0.5 секунди

  // Секвенція 6
  2,              // Кількість повторів
  2,              // Кількість кадрів

  0b00001100,     // Кадр 1: Claw angle + slowness
  90,             // Claw angle
  1,              // Claw slowness
  0,              // delay 0 * 100 мс

  0b00001100,     // Кадр 2: Claw angle + slowness
  0,              // Claw angle
  1,              // Claw slowness
  5               // delay 5 * 100 мс = 0.5 секунди
};

const byte BUILTIN_PROGRAM_2[] PROGMEM = {
  ROBO_DESCRIPTOR_BYTES,

  // Ім'я: Claw sweep
  10,
  'C', 'l', 'a', 'w', ' ', 's', 'w', 'e', 'e', 'p',
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

  // Програма:
  //
  // repeat:3;
  // c:30/1;
  // delay:0;
  // c:75/5;
  // delay:0;
```

```
  // c:120/10;
  // delay:0;
  // c:0/1;
  // delay:0;
  // endrepeat;

  1,              // Кількість секвенцій

  // Секвенція 1
  3,              // Кількість повторів
  4,              // Кількість кадрів

  0b00001100,     // Кадр 1: Claw angle + slowness
  30,             // Claw angle
  1,              // Claw slowness
  0,              // delay 0 * 100 мс

  0b00001100,     // Кадр 2: Claw angle + slowness
  75,             // Claw angle
  5,              // Claw slowness
  0,              // delay 0 * 100 мс

  0b00001100,     // Кадр 3: Claw angle + slowness
  120,            // Claw angle
  10,             // Claw slowness
  0,              // delay 0 * 100 мс

  0b00001100,     // Кадр 4: Claw angle + slowness
  0,              // Claw angle
  1,              // Claw slowness
  0               // delay 0 * 100 мс
};

const byte BUILTIN_PROGRAM_3[] PROGMEM = {
  ROBO_DESCRIPTOR_BYTES,

  // Ім'я: Rotate sweep
  12,
  'R', 'o', 't', 'a', 't', 'e', ' ', 's', 'w', 'e', 'e', 'p',
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

  // Програма:
  //
  // repeat:1;
  // r:0/20;
  // delay:0.5;
  // r:90/20;
  // delay:0.5;
  // r:180/20;
  // delay:0.5;
  // endrepeat;
```

```
  1,              // Кількість секвенцій

  // Секвенція 1
  1,              // Кількість повторів
  3,              // Кількість кадрів

  0b00000011,     // Кадр 1: Rotate angle + slowness
  0,              // Rotate angle
  20,             // Rotate slowness
  5,              // delay 5 * 100 мс = 0.5 секунди

  0b00000011,     // Кадр 2: Rotate angle + slowness
  90,             // Rotate angle
  20,             // Rotate slowness
  5,              // delay 5 * 100 мс = 0.5 секунди

  0b00000011,     // Кадр 3: Rotate angle + slowness
  180,            // Rotate angle
  20,             // Rotate slowness
  5               // delay 5 * 100 мс = 0.5 секунди
};

const byte BUILTIN_PROGRAM_COUNT = 3;

const byte* const BUILTIN_PROGRAMS[] PROGMEM = {
  BUILTIN_PROGRAM_1,
  BUILTIN_PROGRAM_2,
  BUILTIN_PROGRAM_3
};

#endif
```

Позиція	Найменування і технічна характеристика устаткування і матеріалів завод-виготівник (для імпортного устаткування – країна, фірма)	Тип, марка устаткування. Позначення документа і N опитового листа	Одиниця вимірювання		Код заводу-виготовника	Код устаткування матеріалу	Ціна одиниці тис.грн.	Кількість	Маса одиниці устатку-вання, кг
			найменування	код					
1	2	3	4	5	6	7	8	9	10
	Прилади і засоби автоматизації								
1	Позиційний сервопривод постійного струму.	MG90S	шт	796				4	
	Діапазон повороту до 180°. Напруга живлення 6 В. Струм 200 мА. Сигнал керування PWM.								
	Китай.								
2	Джойстик модуль для Ардуїно PS2.	KY-023	шт	796				2	
	Напруга живлення 5 В. Дві віссі і механічна кнопка.								
	Китай.								
3	Bluetooth модуль. Профіль Bluetooth serial port.	HC-05	шт	796				1	
	Робоча частота: 2,4 ГГц ISM. Напруга живлення 3.3 ... 5 В.								
	Китай.								
4	Плата спряження з драйвером електродвигунів.	KS0550	шт	796				1	
	10 стабілізованих каналів живлення для пристроїв. Вхідна напруга 7 ... 12 В.								
	Вихідний неперервний струм до 5 А на всі канали, до 1 А на кожен канал.								
	Китай.								
	Засоби обчислювальной техніки								
5	Мікроконтролерна платформа. Мікроконтролер: ATmega328P.	Arduino Uno	шт	796				1	
	Робоча напруга 5 В. Тактова частота 16 МГц.								
	Китай.								

Виконав	СОЛОВЙОВ		02.06	Розробка та реалізація робота маніпулятора з механічним захопленням з дистанційним керуванням Специфікація устаткування на засоби автоматизації
Перевірив	ЛОСІХІН			
Рецензент	ЧЕРНЕЦЬКИЙ			
Н.Контр.				
Затвердив	ЛЕВЧУК			

А26.14.АРТ.00.С

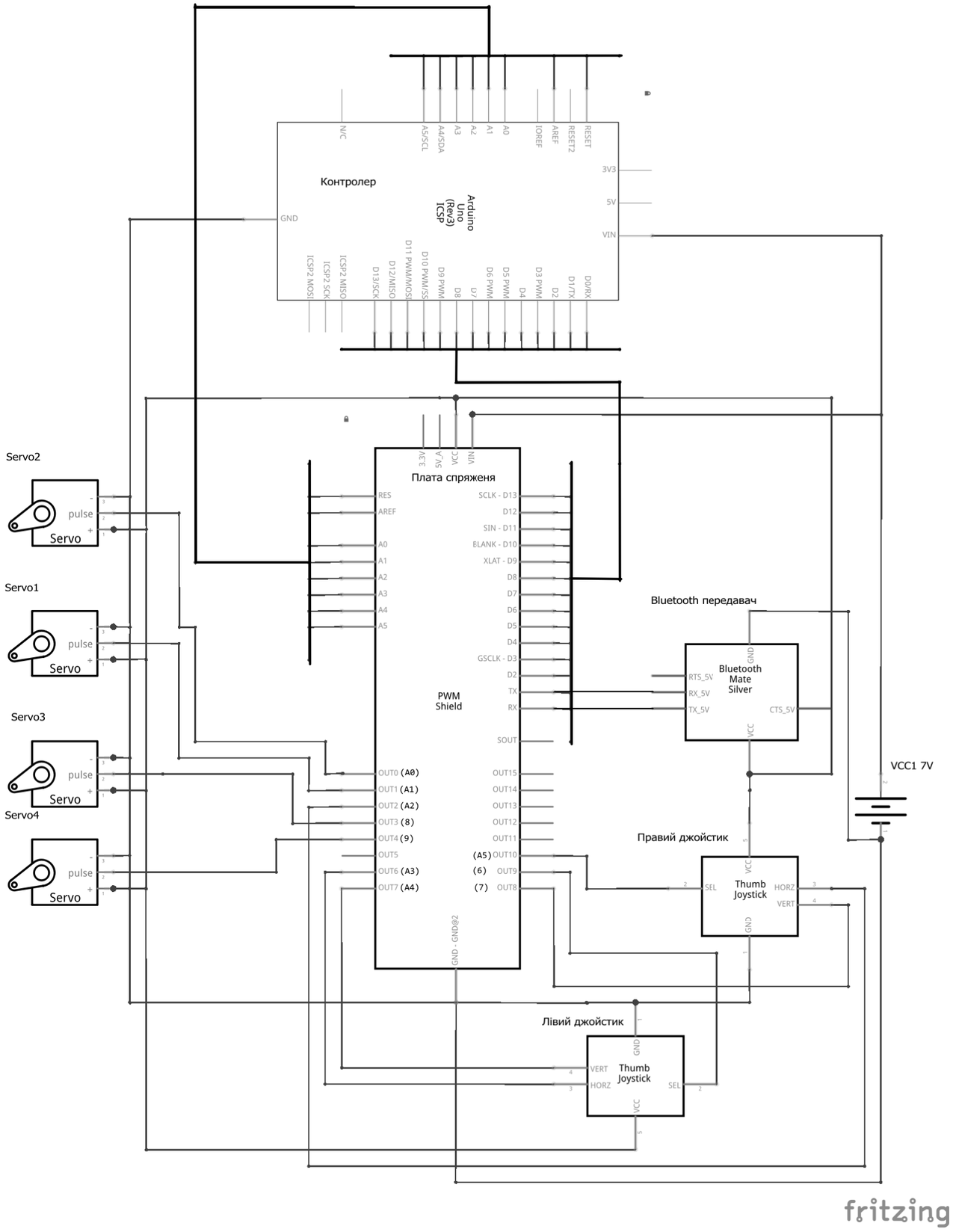
Стадія	Лист	Листів
ДП	1	2

УДУНТ, ННІ УДХТУ, кафедра КІТтаР, гр. 4-АВП-22

Позиція	Найменування і технічна характеристика устаткування і матеріалів завод-виготівник (для імпортного устаткування – країна, фірма)	Тип, марка устаткування. Позначення документа і N опитового листа	Одиниця вимірювання		Код заводу-виготовника	Код устаткування матеріалу	Ціна одиниці тис.грн.	Кількість	Маса одиниці устаткування, кг
			найменування	код					
1	2	3	4	5	6	7	8	9	10
	Електроапаратура								
6	Тримач акумуляторів.	АОС771	шт	796				1	
	Кількість і формфактор 2×18650. Послідовне з'єднання. Зі штекером DC 5,5×2,1 мм.								
	Китай.								
7	Кабель з'єднувальний DuPont.	DuPont 20 F-F	шт	796				1	
	Тип з'єднання F-F. Кількість у кабелі: 20 дротів. Довжина 200 мм.								
	Китай.								
8	Кабель з'єднувальний DuPont.	DuPont 10 M-F	шт	796				3	
	Тип з'єднання M-F. Довжина 100 мм.								
	Китай.								

ЗМІСТ ТОМУ

№	Найменування	Позначення	Кількість листів
1	Титульний лист		1
2	Завдання на дипломний проект		1
3	Відомість дипломного проекту	А26.14.АРТ.00.ВД	1
4	Пояснювальна записка	А26.14.АРТ.00.ПЗ	158
5	Специфікація устаткування на засоби автоматизації	А26.14.АРТ.00.СО1	2



				А26.14.АРТ.01.СБ		
				м. Дніпро, УДУНТ, ННІ УДХТУ		
Виконав	Соловйов			Розробка та реалізація роботи маніпулятора з механічним захопленням з дистанційним керуванням	Стадія	Аркуш	Аркушів
Перевірив	Лосихін				ДП	1	1
Рецензент	Чернецький						
НКонтр				Схема електрична принципова підключень компонентів	УДУНТ, ННІ УДХТУ, каф. КІТтаР, гр. 4-АВП-22		
Затвердив	Левчук						

Servo4
Servo3
Servo1
Servo2
18650
18650
Лівий джойстик
Правий джойстик

				А26.14.АРТ.02.СБ		
				м. Дніпро, УДУНТ, ННІ УДХТУ		
Виконав	Соловйов			Розробка та реалізація робота маніпулятора з механічним захопленням з дистанційним керуванням	Стадія	Аркуш	Аркушів
Перевірив	Лосіхін				ДП	1	1
Рецензент	Чернецький						
НКонтр				Схема електрична принципова з'єднань компонентів	УДУНТ, ННІ УДХТУ, каф. КІТтаР, гр. 4-АВП-22		
Затвердив	Левчук						

fritzing