НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ХАРЧОВИХ ТЕХНОЛОГІЙ

Кафедра автоматизації та компютерних технологій систем управління

**КУРСОВА РОБОТА**

“Програмна інженерія в системах управління”

на тему: «\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_»

Студента\_\_\_1\_\_\_курсу\_\_1\_\_\_групи

Напряму підготовки\_\_6.05020\_\_\_\_\_\_\_\_\_\_

спеціальності\_«Автоматизація та комп’ютерно-

інтегровані технології»\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Керівник: доцент АКТСУ

кандидат технічних наук \_\_\_\_\_\_\_\_\_ Пупена О.М.

(підпис)

Національна шкала\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_ Оцінка: ECTS\_\_\_

Члени комісії

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Пупена О.М.

(підпис) (прізвище та ініціали )

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Міркевич Р.М.

(підпис) ( прізвище та ініціали )

м. Київ – 2021 рік

Зміст

Зміст

[Вступ 3](#_Toc168562054)

[Огляд існуючих рішень 3](#_Toc168562055)

[Промислове застосування 3](#_Toc168562056)

[Комплектуючі для збирання маніпулятора 4](#_Toc168562057)

[Arduino Uno: Загальна інформація 4](#_Toc168562058)

[SG90 Servo: Загальна інформація 5](#_Toc168562059)

[Модуль подвійного джойстика для Arduino (PS2 JoyStick): Загальна інформація 6](#_Toc168562060)

[1. Перевірка компонентів: 7](#_Toc168562061)

[2. Підготовка плати Arduino Uno: 8](#_Toc168562062)

[3. Підключення сервоприводів: 8](#_Toc168562063)

[4. Монтаж каркасу: 8](#_Toc168562064)

[5. Підключення модуля джойстика: 8](#_Toc168562065)

[6. Перевірка живлення: 8](#_Toc168562066)

[7. Завантаження програмного забезпечення: 8](#_Toc168562067)

[8. Перевірка роботи джойстиків 10](#_Toc168562068)

[Робота з Node-Red 11](#_Toc168562069)

[Реалізація керування роботом через Node-Red 11](#_Toc168562070)

[Основний код (3 на 4) 15](#_Toc168562071)

[Основний код (4 на 4) 21](#_Toc168562072)

[Основний 29](#_Toc168562073)

[Корисні посилання 35](#_Toc168562074)

[Посилання на GitHub 35](#_Toc168562075)

## Вступ

Метою даного проекту є створення механічної руки-маніпулятора з 4 сервоприводами, яка програмується на базі платформи Arduino UNO. Основна ідея полягає в тому, щоб запрограмувати маніпулятор на виконання заздалегідь визначених рухів, базуючись на координатах, збережених у пам'яті. При запуску коду маніпулятор буде виконувати заплановані рухи автоматично.

## Огляд існуючих рішень

У сучасному світі існує безліч подібних проектів, доступних у відкритому доступі, які використовують Arduino та інші мікроконтролери для керування механічними маніпуляторами. Наприклад, багато ентузіастів та інженерів створюють роботизовані руки, які можуть виконувати різні завдання, від простого захвату об'єктів до складних маніпуляцій. Найбільш поширені проекти включають:

**3D-друковані роботизовані руки**: Ці проекти зазвичай використовують 3D-принтери для створення компонентів маніпулятора та керують ними за допомогою Arduino. Вони часто оснащені кількома сервоприводами для забезпечення різних ступенів свободи.

**Проекти з відкритим кодом**: На таких платформах, як GitHub, можна знайти багато проектів з детальними інструкціями та кодом для створення роботизованих маніпуляторів. Наприклад, Open Source Robotic Arm Project пропонує різні варіанти конструкції та програмного забезпечення для створення роботів з різними можливостями.

## Промислове застосування

Цей проект є мініатюрною версією промислових роботизованих маніпуляторів, які використовуються на виробництвах для виконання різних автоматизованих завдань, таких як збирання, пакування, зварювання та інші. Хоча наш маніпулятор менш потужний і має обмежені можливості порівняно з великими промисловими роботами, він демонструє основні принципи роботи таких систем.

Використання подібних роботизованих маніпуляторів у промисловості дозволяє підвищити ефективність виробництва, забезпечити точність виконання операцій та зменшити витрати на ручну працю. Таким чином, даний проект представляє собою інтеграцію апаратного та програмного забезпечення для створення універсальної та зручної у використанні механічної руки-маніпулятора.

Компоненти

### **Комплектуючі для збирання маніпулятора**

Для збирання маніпулятора необхідні такі компоненти:

**Серворухи**:

* 4 сервероприводи **Micro Servo SG90** (180 градусів).

**Контролер**:

* Плата **Arduino Uno**.

**Модуль управління**:

* Модуль подвійного джойстика для Arduino (**PS2 JoyStick**).

**Каркас**:

* Каркас з акрилу (відсутній у списку компонентів, але необхідний для побудови маніпулятора).

Всі компоненти були замовлені у наборі з магазину Prom. Посилання на набір: [Механічний робот-маніпулятор](https://elektro-kramnitsya.com.ua/ua/p2088223298-mehanicheskij-robot-manipulyator.html).

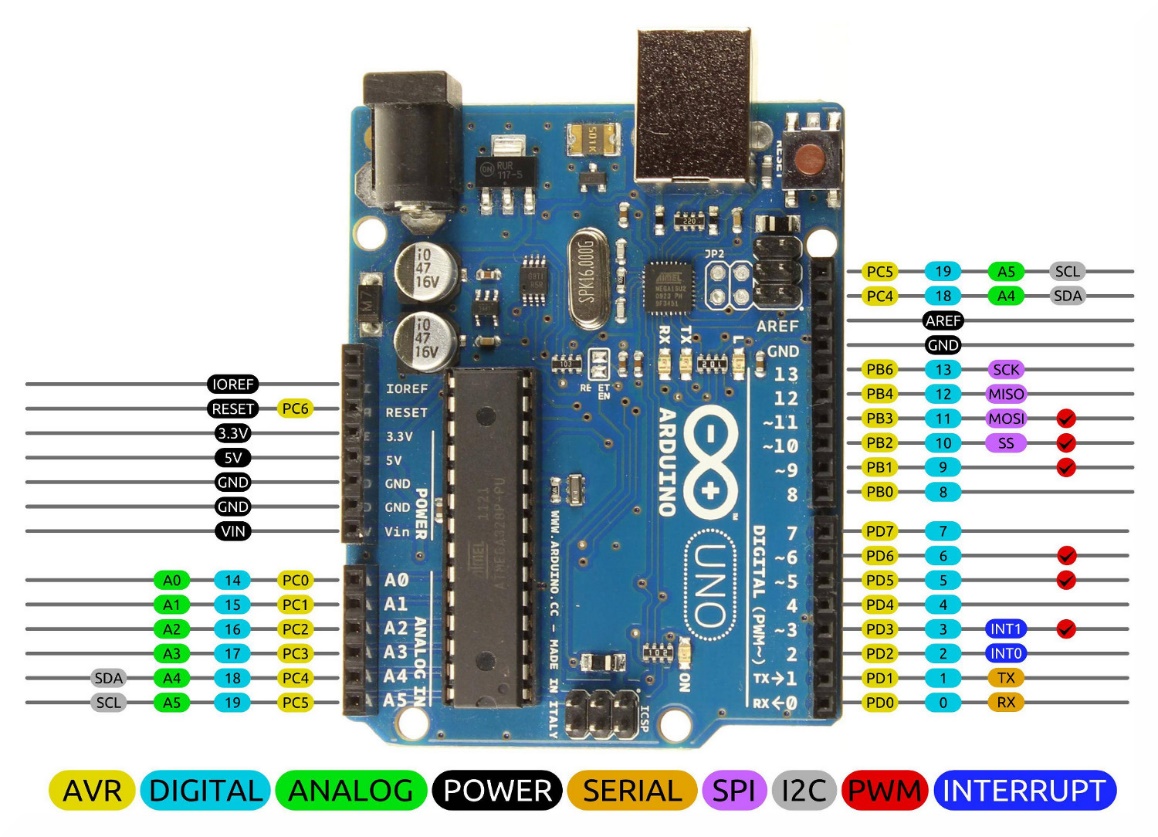
### **Arduino Uno: Загальна інформація**

**Характеристики:**

* **Мікроконтролер:** ATmega328
* **Робоча напруга:** 5В
* **Напруга живлення (рекомендовано):** 7-12В
* **Напруга живлення (межова):** 6-20В
* **Цифрові входи-виходи:** 14 (з них 6 можуть використовуватись в якості ШІМ-виходів)
* **Аналогові входи:** 6
* **Максимальний ток одного виходу:** 40 мА
* **Максимальний вихідний ток виходу 3.3V:** 50 мА
* **Flash-пам’ять:** 32 КБ (ATmega328)
* **SRAM:** 2 КБ (ATmega328)
* **EEPROM:** 1 КБ (ATmega328)
* **Тактова частота:** 16 МГц

**Входи і виходи:**

* **VIN:** За необхідності, через цей контакт подається зовнішнє живлення.
* **5V:** На цей контакт подається напруга 5В від стабілізатора на платі.
* **3V3:** На цей контакт подається з плати 3.3В з внутрішнього стабілізатора.
* **IOREF:** Надає платам розширення інформацію про робочу напругу мікроконтролера Ардуіно.
* **Digital I/O:** 14 цифрових контактів, кожен з яких може працювати в якості входу або виходу.
* **Analog I/O:** 6 аналогових входів (A0 - A5).
* **AREF:** Опорна напруга для аналогових входів.
* **Reset:** Формування низького рівня (LOW) призведе до перезавантаження мікроконтролера.



Arduino Uno також має можливість виконувати додаткові функції через певні контакти, такі як послідовний інтерфейс, зовнішні переривання, аналоговий ШІМ, SPI та TWI.

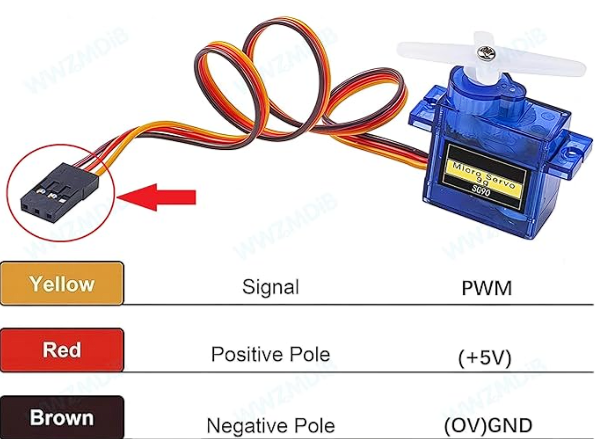
### **SG90 Servo: Загальна інформація**

**Характеристики:**

* **Швидкість без навантаження:** 0.12 сек/60 град. при живленні 4.8В
* **Крутний момент:** 2 кг/см
* **Температурний діапазон:** -30°C до +60°C
* **Ширина мертвої зони:** 4 мікросекунди
* **Робоча напруга живлення:** 3.5-5 В
* **Споживаний струм у русі:** 50-80 мА
* **Споживаний струм утриманні:** 5-10 мА
* **Кут повороту:** 180 градусів
* **Розміри:** 3.3 см x 3 см x 1.3 см
* **Вага:** 15 гр

**Розпинування для підключення сервоприводу:**

* Коричневий (**-**)
* Червоний (**+**)
* Помаранчевий (управління)



Сервопривод SG90 є компактним і має широкий діапазон робочих температур, що робить його ідеальним варіантом для багатьох проектів з рухомими частинами.

### Модуль подвійного джойстика для Arduino (PS2 JoyStick): Загальна інформація

**Характеристики:**

* **Тип модуля:** Подвійний джойстик
* **Напруга живлення:** 5 В
* **Вихідні сигнали:** Аналогові та цифрові
* **Інтерфейс:** Arduino Uno

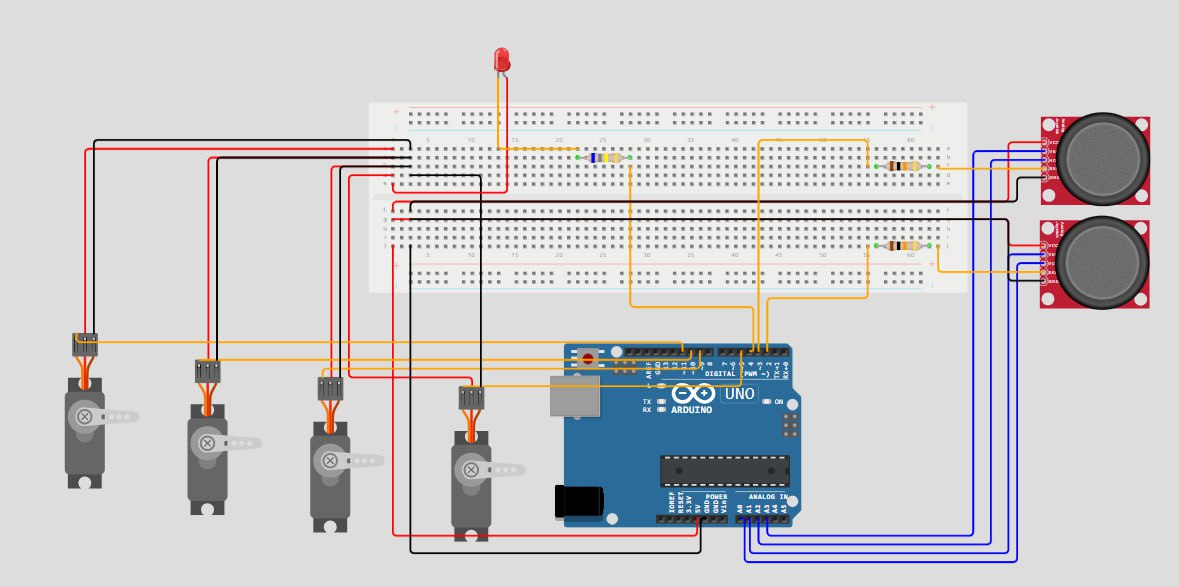
**Опис:** Модуль подвійного джойстика для Arduino (PS2 JoyStick) дозволяє зручно керувати рухом маніпулятора або іншими пристроями. Він має два аналогових виходи для осей X і Y, а також один цифровий вихід для кнопки натискання.

**Функції:**

* **Аналогові виходи:** Два потенціометри надають змінні сигнали для осей X і Y, що дозволяє точно контролювати рухи в двох вимірах.
* **Цифровий вихід:** Кнопка натискання розташована під джойстиком і може використовуватися для додаткових функцій, наприклад, для підтвердження вибору або запуску певних дій.

**Підключення:**

* **VCC:** Живлення модуля (5 В)
* **GND:** Загальний провід
* **VRx:** Аналоговий вихід для осі X
* **VRy:** Аналоговий вихід для осі Y
* **SW:** Цифровий вихід кнопки натискання



(схема підключення з використанням одинарних джойстиків керування)

**Особливості:** Модуль повторює всі піни плати Arduino Uno, що дозволяє просто накласти його поверх плати Arduino Uno. Це забезпечує зручне і швидке підключення без додаткових дротів.

**Підключення для сервоприводів:** Модуль має 6 трипінових з'єднань для сервоприводів, що підключаються до наступних пінів:

* **D11 (1)**
* **D10 (2)**
* **D9 (3)**
* **D5 (4)**
* **D7**
* **D6**

Перевірка та підготовка до роботи

Перед початком роботи з маніпулятором необхідно виконати наступні кроки:

### 1. Перевірка компонентів:

* Переконайтеся, що всі компоненти з комплекту є наявними і не мають пошкоджень.
* Компоненти включають: 4 сервоприводи SG90, плата Arduino Uno, модуль подвійного джойстика (PS2 JoyStick), каркас з акрилу та необхідні дроти для підключення.

### 2. Підготовка плати Arduino Uno:

* Встановіть плату Arduino Uno на робочу поверхню.
* Перевірте підключення всіх необхідних компонентів до плати Arduino.

### 3. Підключення сервоприводів:

* Підключіть сервоприводи до відповідних пінів на модулі джойстика:
  + D11 (1)
  + D10 (2)
  + D9 (3)
  + D5 (4)
  + D7
  + D6
* Переконайтеся, що дроти підключені правильно: коричневий (GND), червоний (5V), помаранчевий (управління(S)).

### 4. Монтаж каркасу:

* Зберіть каркас з акрилу, дотримуючись інструкцій.
* Закріпіть сервоприводи на каркасі відповідно до схеми.

### 5. Підключення модуля джойстика:

* Встановіть модуль подвійного джойстика на плату Arduino Uno.
* Переконайтеся, що модуль правильно накладений на плату Arduino та підключений до відповідних пінів.

### 6. Перевірка живлення:

* Переконайтеся, що джерело живлення відповідає вимогам (5В для Arduino, 3.5-5 В для сервоприводів).
* Підключіть плату Arduino до комп'ютера через USB або до зовнішнього джерела живлення (7-12В).

### 7. Завантаження програмного забезпечення:

* Відкрийте Arduino IDE на вашому комп'ютері.
* Скопіюйте та вставте наступний код:

#include <Servo.h> // Підключення бібліотеки Servo для управління сервомоторами

Servo servo\_x;  // створення об'єкту servo для управління сервомотором x  
Servo servo\_y;  // створення об'єкту servo для управління сервомотором y  
Servo servo\_z;  // створення об'єкту servo для управління сервомотором z  
Servo servo\_grip;  // створення об'єкту servo для управління сервомотором захвату

int x\_pos;    // змінна для зберігання позиції сервомотора x  
int y\_pos;    // змінна для зберігання позиції сервомотора y  
int z\_pos;    // змінна для зберігання позиції сервомотора z  
int grip\_pos;    // змінна для зберігання позиції сервомотора захвату  
int direction = 1;  // змінна для зберігання напрямку руху (не використовується в даному коді)

void setup() {  
 // налаштування початкових параметрів, виконується один раз при запуску:  
 Serial.begin(9600);  // ініціалізація серійного з'єднання зі швидкістю 9600 біт/с  
 servo\_x.attach(11);  // підключення сервомотора x до піна 11  
 servo\_y.attach(10);  // підключення сервомотора y до піна 10  
 servo\_z.attach(9);  // підключення сервомотора z до піна 9  
 servo\_grip.attach(5);  // підключення сервомотора захвату до піна 5  
   
 x\_pos = 90;  // встановлення початкової позиції сервомотора x на 90 градусів  
 y\_pos = 90;  // встановлення початкової позиції сервомотора y на 90 градусів  
 z\_pos = 175;  // встановлення початкової позиції сервомотора z на 175 градусів  
 grip\_pos = 0;  // встановлення початкової позиції сервомотора захвату на 0 градусів

 servo\_x.write(x\_pos);  // рух сервомотора x до позиції x\_pos  
 servo\_y.write(y\_pos);  // рух сервомотора y до позиції y\_pos  
 servo\_z.write(z\_pos);  // рух сервомотора z до позиції z\_pos  
 servo\_grip.write(grip\_pos);  // рух сервомотора захвату до позиції grip\_pos  
}

void loop() {  
 // основний код, який виконується постійно:  
 // delay(100);  // затримка (закоментовано)  
 String recvString = Serial.readStringUntil("\n");  // зчитування рядка з серійного порту до символу нової строки  
 recvString.trim();  // видалення пробілів на початку та в кінці рядка  
 if (recvString.length() > 5) {  // перевірка, чи довжина рядка більше 5 символів  
   int i1 = recvString.indexOf(',');  // пошук першої коми  
   int i2 = recvString.indexOf(',', i1 + 1);  // пошук другої коми  
   int i3 = recvString.indexOf(',', i2 + 1);  // пошук третьої коми

   String v1 = recvString.substring(0, i1);  // отримання підрядка до першої коми  
   String v2 = recvString.substring(i1 + 1, i2);  // отримання підрядка між першою та другою комами  
   String v3 = recvString.substring(i2 + 1, i3);  // отримання підрядка між другою та третьою комами  
   String v4 = recvString.substring(i3 + 1);  // отримання підрядка після третьої коми  
   x\_pos = v1.toInt();  // перетворення підрядка v1 на ціле число та збереження в x\_pos  
   y\_pos = v2.toInt();  // перетворення підрядка v2 на ціле число та збереження в y\_pos  
   z\_pos = v3.toInt();  // перетворення підрядка v3 на ціле число та збереження в z\_pos  
   grip\_pos = v4.toInt();  // перетворення підрядка v4 на ціле число та збереження в grip\_pos

   Serial.println(x\_pos);  // виведення значення x\_pos в серійний порт  
   Serial.print(" ");  // виведення пробілу в серійний порт  
   Serial.println(y\_pos);  // виведення значення y\_pos в серійний порт  
   Serial.print(z\_pos);  // виведення значення z\_pos в серійний порт  
   Serial.print(" ");  // виведення пробілу в серійний порт  
   Serial.println(grip\_pos);  // виведення значення grip\_pos в серійний порт  
 }  
 Serial.println(x\_pos);  // виведення значення x\_pos в серійний порт  
 servo\_x.write(x\_pos);  // рух сервомотора x до позиції x\_pos  
 servo\_y.write(y\_pos);  // рух сервомотора y до позиції y\_pos  
 servo\_z.write(z\_pos);  // рух сервомотора z до позиції z\_pos  
 servo\_grip.write(grip\_pos);  // рух сервомотора захвату до позиції grip\_pos  
}

* Завантажте код на плату Arduino Uno.
* Щоб перевірити роботу сервоприводів, змінюйте значення в наступних рядках коду:

x\_pos = 90;

y\_pos = 90;

z\_pos = 175;

grip\_pos = 0;

Числа вказують на градуси повороту сервоприводів:

* **x\_pos:** сервопривод в основі
* **y\_pos:** сервопривод в плечі
* **z\_pos:** сервопривід в локті маніпулятора
* **grip\_pos:** сервопривод, відповідальний за стискання самого захвату

## 8. Перевірка роботи джойстиків

* Відкрийте Arduino IDE на вашому комп'ютері.
* Скопіюйте та вставте наступний код:

const int xPin = A0; // Визначення піну для осі X  
const int yPin = A1; // Визначення піну для осі Y  
const int dw = 2; // Визначення піну для кнопки

void setup() {  
 Serial.begin(9600); // Ініціалізація серійного зв'язку зі швидкістю 9600 бод  
 pinMode(dw, INPUT\_PULLUP); // Налаштування піну для кнопки як вхідного з підтягуванням до VCC  
}

void loop() {  
 int valueX = analogRead(xPin); // Зчитування значення з осі X  
 int valueY = analogRead(yPin); // Зчитування значення з осі Y  
 Serial.print("X: "); // Виведення тексту "X: " у серійний порт  
 Serial.print(valueX); // Виведення значення осі X у серійний порт  
 Serial.print("\t Y: "); // Виведення тексту " Y: " у серійний порт  
 Serial.print(valueY); // Виведення значення осі Y у серійний порт  
 Serial.print("\t Button "); // Виведення тексту " Button " у серійний порт  
 Serial.println(digitalRead(dw)); // Зчитування та виведення стану кнопки у серійний порт  
}

Цей код призначений для перевірки роботи джойстиків. Він зчитує аналогові значення з осей X та Y, а також стан кнопки, і виводить їх у серійний порт для подальшого аналізу. Інформація про положення джойстиків та натискання кнопки буде відображатися в Серіал Моніторі.\

* Запустіть тестовий скетч, щоб переконатися в правильності роботи сервоприводів та джойстика.
* Перевірте, чи правильно реагує маніпулятор на команди з джойстика.

## Робота з Node-Red

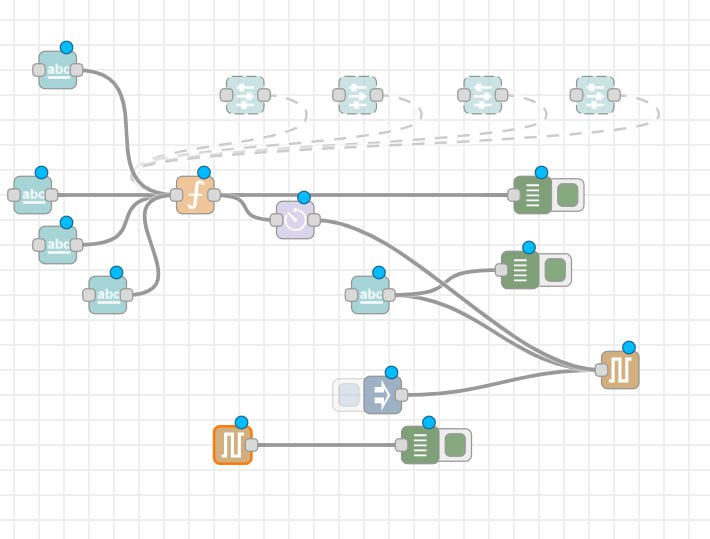
Node-Red є потужним інструментом для візуального програмування, який дозволяє легко створювати потоки даних та автоматизацію завдань. Нижче наведені приклади роботи з ним і реалізація керування роботом через нього.

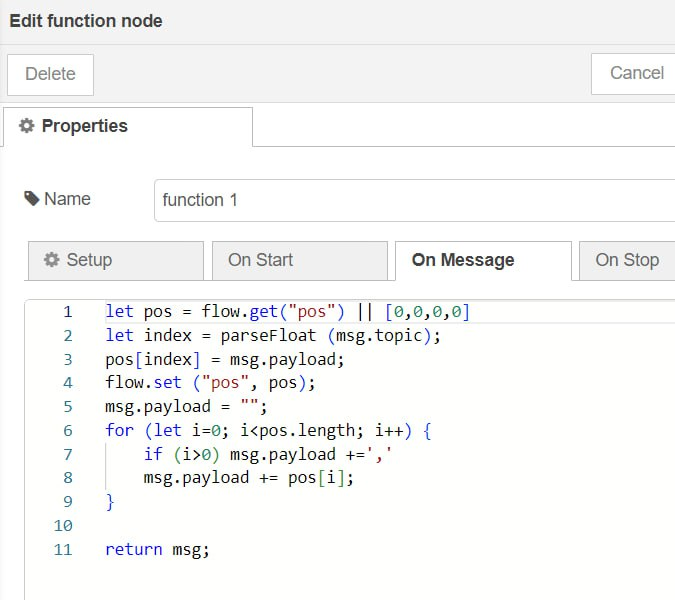
### Реалізація керування роботом через Node-Red

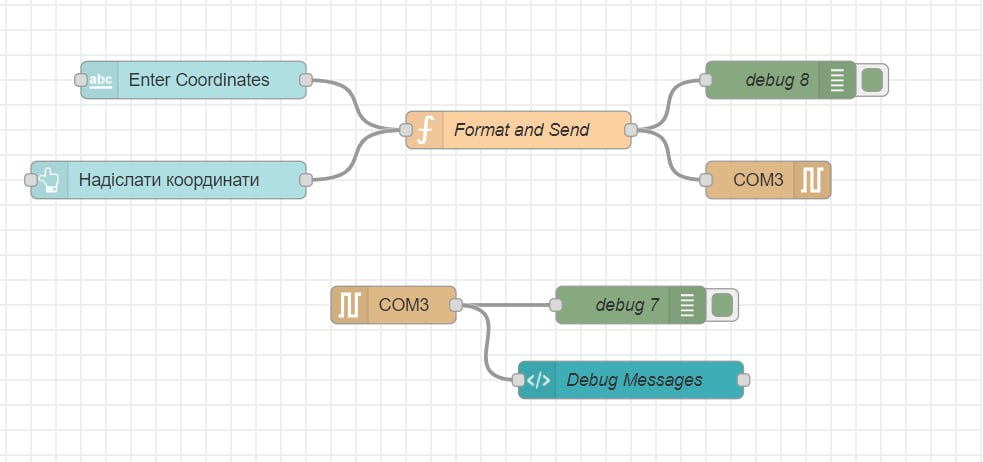
1. **Встановлення Node-Red**: Скачайте та встановіть Node-Red на ваш комп'ютер.
2. **Підключення Arduino до Node-Red**: Використовуйте Node-Red для серійного зв'язку з Arduino.
3. **Створення потоку даних**: Налаштуйте Node-Red для прийому команд і відправки їх на Arduino.
4. **Створення інтерфейсу керування**: Використовуйте Node-Red Dashboard для створення графічного інтерфейсу керування роботом.

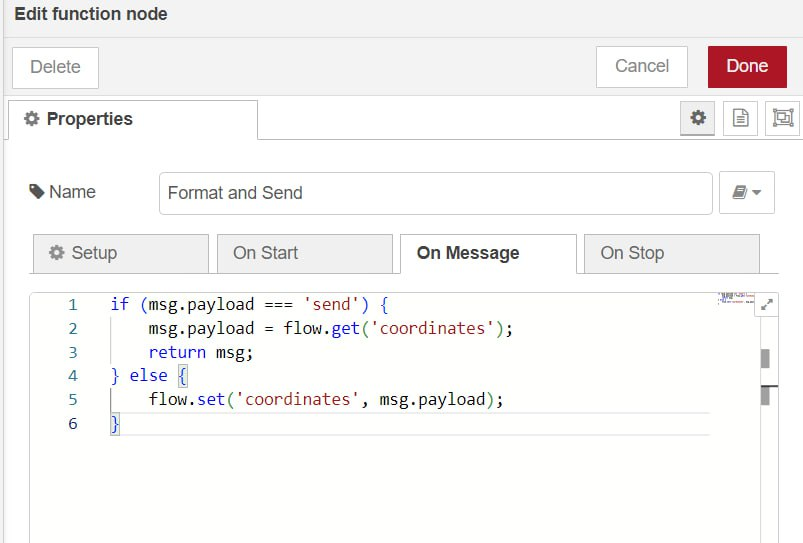
Node-Red дозволяє легко реалізувати керування роботом через веб-інтерфейс або інші сервіси.

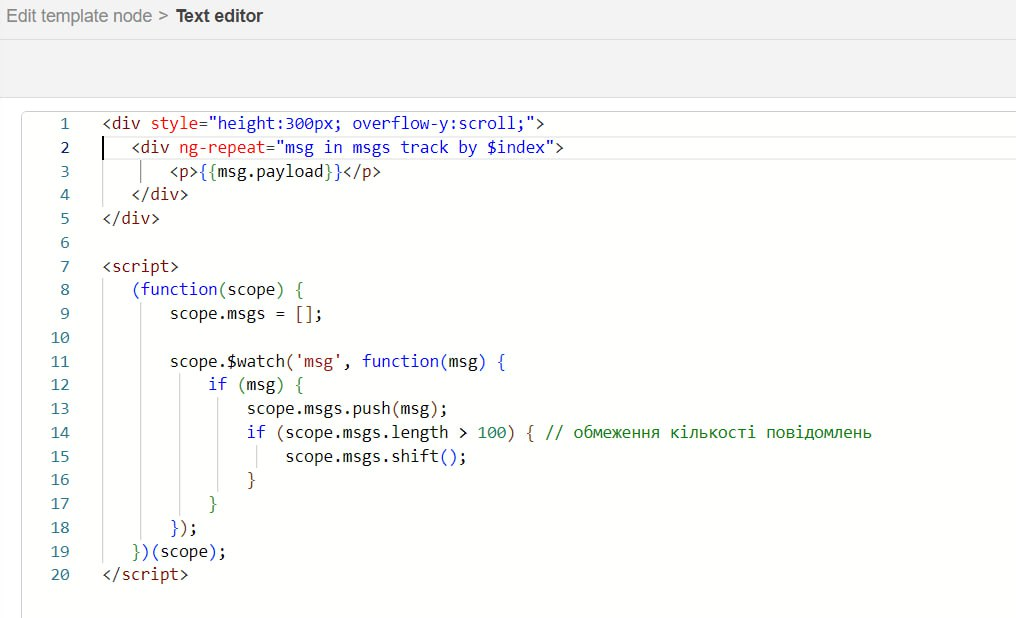
Нижче наведені скріншоти з нашої роботи з Node-Red, що демонструють створення потоку даних та інтерфейсу керування роботом.











## Основний код (3 на 4)

Цей код визначає змінні для сервоприводів, позицій та інтервалів між операціями. Ініціалізує серійний зв'язок і підключає сервоприводи до відповідних пінів. Також встановлює початкові позиції сервоприводів. Основні функції коду:

* **Ініціалізація**: Визначення змінних для сервоприводів, позицій та інтервалів. Ініціалізація серійного зв'язку та підключення сервоприводів до пінів.
* **Початкові позиції**: Встановлення початкових позицій сервоприводів.
* **Перевірка часу**: Перевірка поточного часу і виклик відповідних функцій через задані інтервали.
* **Зміна позицій**: Зміна позицій сервоприводів на основі поточного кроку та оновлення лічильника кроків.
* **Зчитування позицій**: Зчитування поточних позицій сервоприводів та виведення їх у серійний порт.
* **Обробка JSON**: Перевірка наявності нових JSON даних у серійному порту, десеріалізація їх та оновлення масиву

**Код 3 на 4:**

#include <Servo.h> // Підключення бібліотеки для керування сервоприводами

#include <ArduinoJson.h> // Підключення бібліотеки для роботи з JSON

Servo servo\_x; // Оголошення об'єкту для керування сервоприводом по осі X

Servo servo\_y; // Оголошення об'єкту для керування сервоприводом по осі Y

Servo servo\_z; // Оголошення об'єкту для керування сервоприводом по осі Z

Servo servo\_grip; // Оголошення об'єкту для керування сервоприводом захвату

int step = 0; // Лічильник кроків для зміни позицій сервоприводів

const int stepscount = 3; // Кількість кроків у послідовності

int x\_pos; // Змінна для зберігання позиції сервопривода по осі X

int y\_pos; // Змінна для зберігання позиції сервопривода по осі Y

int z\_pos; // Змінна для зберігання позиції сервопривода по осі Z

int grip\_pos; // Змінна для зберігання позиції сервопривода захвату

unsigned long previousMillisStep = 0; // Час останнього виконання функції getSetpoint

unsigned long previousMillisRead = 0; // Час останнього виконання функції ReadPosition

unsigned long currentMillis; // Поточний час

unsigned long previousMillisCheckJson = 0; // Час останнього виконання функції checkForNewJson

const long intervalStep = 3000; // Інтервал між викликами getSetpoint (в мілісекундах)

const long intervalRead = 3000; // Інтервал між викликами ReadPosition (в мілісекундах)

const long intervalCheckJson = 3000; // Інтервал між викликами checkForNewJson (в мілісекундах)

int arrayStep[3][4] = {

{111, 122, 155, 77}, // Початкові позиції для кожного кроку

{140, 120, 150, 70}, // Початкові позиції для кожного кроку

{123, 134, 156, 66}, // Початкові позиції для кожного кроку

};

void setup() {

Serial.begin(9600); // Ініціалізація серійного зв'язку зі швидкістю 9600 бод

servo\_x.attach(11); // Приєднання сервопривода осі X до 11 піна

servo\_y.attach(10); // Приєднання сервопривода осі Y до 10 піна

servo\_z.attach(9); // Приєднання сервопривода осі Z до 9 піна

servo\_grip.attach(5); // Приєднання сервопривода захвату до 5 піна

x\_pos = arrayStep[0][0]; // Ініціалізація позиції X з першого кроку

y\_pos = arrayStep[0][1]; // Ініціалізація позиції Y з першого кроку

z\_pos = arrayStep[0][2]; // Ініціалізація позиції Z з першого кроку

grip\_pos = arrayStep[0][3]; // Ініціалізація позиції захвату з першого кроку

servo\_x.write(x\_pos); // Переміщення сервопривода осі X в початкову позицію

servo\_y.write(y\_pos); // Переміщення сервопривода осі Y в початкову позицію

servo\_z.write(z\_pos); // Переміщення сервопривода осі Z в початкову позицію

servo\_grip.write(grip\_pos); // Переміщення сервопривода захвату в початкову позицію

getSetpoint(); // Виклик функції для ініціалізації позицій сервоприводів

}

void loop() {

currentMillis = millis(); // Отримання поточного часу

if (currentMillis - previousMillisStep >= intervalStep) { // Перевірка, чи пройшов інтервал для getSetpoint

previousMillisStep = currentMillis; // Оновлення часу останнього виклику getSetpoint

getSetpoint(); // Виклик функції для зміни позицій сервоприводів

}

if (currentMillis - previousMillisRead >= intervalRead) { // Перевірка, чи пройшов інтервал для ReadPosition

previousMillisRead = currentMillis; // Оновлення часу останнього виклику ReadPosition

ReadPosition(); // Виклик функції для зчитування поточних позицій сервоприводів

}

if (currentMillis - previousMillisCheckJson >= intervalCheckJson) { // Перевірка, чи пройшов інтервал для checkForNewJson

previousMillisCheckJson = currentMillis; // Оновлення часу останнього виклику checkForNewJson

checkForNewJson(); // Виклик функції для перевірки нових даних JSON

}

servo\_x.write(x\_pos); // Переміщення сервопривода осі X в поточну позицію

servo\_y.write(y\_pos); // Переміщення сервопривода осі Y в поточну позицію

servo\_z.write(z\_pos); // Переміщення сервопривода осі Z в поточну позицію

servo\_grip.write(grip\_pos); // Переміщення сервопривода захвату в поточну позицію

}

void getSetpoint() {

x\_pos = arrayStep[step][0]; // Зміна позиції X на основі поточного кроку

y\_pos = arrayStep[step][1]; // Зміна позиції Y на основі поточного кроку

z\_pos = arrayStep[step][2]; // Зміна позиції Z на основі поточного кроку

grip\_pos = arrayStep[step][3]; // Зміна позиції захвату на основі поточного кроку

step++; // Збільшення лічильника кроків

if (step >= stepscount) { // Перевірка, чи досягнуто кінця послідовності кроків

step = 0; // Скидання лічильника кроків, якщо досягнуто кінця

}

}

void ReadPosition() {

int x = servo\_x.read(); // Зчитування поточної позиції сервопривода осі X

int y = servo\_y.read(); // Зчитування поточної позиції сервопривода осі Y

int z = servo\_z.read(); // Зчитування поточної позиції сервопривода осі Z

int grip = servo\_grip.read(); // Зчитування поточної позиції сервопривода захвату

Serial.print(x); // Виведення позиції X в серійний порт

Serial.print(" "); // Виведення пробілу

Serial.print(y); // Виведення позиції Y в серійний порт

Serial.print(" "); // Виведення пробілу

Serial.print(z); // Виведення позиції Z в серійний порт

Serial.print(" "); // Виведення пробілу

Serial.println(grip); // Виведення позиції захвату в серійний порт та перехід на новий рядок

}

void checkForNewJson() {

StaticJsonDocument<200> doc; // Оголошення документа JSON з фіксованим розміром 200 байт

if (Serial.available() > 0) { // Перевірка, чи є доступні дані в серійному порті

String input = Serial.readString(); // Зчитування даних з серійного порту в рядок

input.trim(); // Видалення будь-яких початкових або кінцевих пробілів

Serial.print("Received input: "); // Виведення отриманих даних в серійний порт

Serial.println(input); // Виведення отриманих даних в серійний порт з переходом на новий рядок

if (input.charAt(input.length() - 1) != ']') { // Перевірка, чи рядок закінчується символом ']', якщо ні, то додати його

input += "]"; // Додавання символу ']' до кінця рядка

}

DeserializationError error = deserializeJson(doc, input); // Спроба десеріалізації JSON з рядка

if (error) { // Перевірка на помилку десеріалізації

Serial.print(F("deserializeJson() failed: ")); // Виведення повідомлення про помилку

Serial.println(error.c\_str()); // Виведення опису помилки

switch (error.code()) { // Перевірка коду помилки

case DeserializationError::InvalidInput: // Якщо помилка через некоректний вхідний рядок

Serial.print(F("Invalid input!")); // Виведення повідомлення про некоректний вхідний рядок

break;

case DeserializationError::NoMemory: // Якщо помилка через недостатню пам'ять

Serial.print(F("Not enough memory")); // Виведення повідомлення про недостатню пам'ять

break;

}

return; // Вихід з функції у разі помилки

}

JsonArray jsonArray = doc.as<JsonArray>(); // Перетворення десеріалізованого документа в JSON масив

if (jsonArray.size() != 3) { // Перевірка розміру зовнішнього масиву

Serial.println("Invalid JSON format"); // Виведення повідомлення про некоректний формат JSON

return; // Вихід з функції у разі некоректного формату

}

for (int i = 0; i < 3; i++) { // Цикл для оновлення значень масиву arrayStep з отриманого JSON

JsonArray innerArray = jsonArray[i]; // Отримання внутрішнього масиву з JSON

if (innerArray.size() != 4) { // Перевірка, чи внутрішній масив має 4 елементи

Serial.println("Invalid JSON format"); // Виведення повідомлення про некоректний формат JSON

return; // Вихід з функції у разі некоректного формату

}

for (int j = 0; j < 4; j++) { // Цикл для оновлення значень кожного елемента в масиві arrayStep

arrayStep[i][j] = innerArray[j]; // Присвоєння значення з JSON до масиву arrayStep

}

}

}

}

## Основний код (4 на 4)

Цей код використовує бібліотеку Servo.h для контролю сервоприводів і ArduinoJson.h для роботи з даними у форматі JSON. Основні функції коду:

* **Ініціалізація**: Ініціалізація серійного зв'язку та підключення сервоприводів до пінів. Встановлення початкових позицій сервоприводів.
* **Перевірка часу**: Постійно перевіряється поточний час і, якщо пройшов відповідний інтервал часу, викликаються відповідні функції.
* **Оновлення позицій**: Позиції сервоприводів оновлюються на основі поточних значень змінних. Після досягнення кінця масиву кроків, лічильник кроків скидається.
* **Зчитування позицій**: Зчитування поточних позицій сервоприводів і виведення їх у серійний порт.
* **Обробка JSON**: Зчитування даних з серійного порту, перевірка їх на наявність нового JSON. Якщо отриманий JSON правильний, оновлення масиву arrayStep новими значеннями позицій для кожного кроку.

**Код 4 на 4**:

#include <Servo.h> // Підключення бібліотеки для керування сервоприводами

#include <ArduinoJson.h> // Підключення бібліотеки для роботи з JSON

Servo servo\_x; // Оголошення об'єкту для керування сервоприводом по осі X

Servo servo\_y; // Оголошення об'єкту для керування сервоприводом по осі Y

Servo servo\_z; // Оголошення об'єкту для керування сервоприводом по осі Z

Servo servo\_grip; // Оголошення об'єкту для керування сервоприводом захвату

int step = 0; // Лічильник кроків для зміни позицій сервоприводів

const int stepscount = 4; // Кількість кроків у послідовності

int x\_pos; // Змінна для зберігання позиції сервопривода по осі X

int y\_pos; // Змінна для зберігання позиції сервопривода по осі Y

int z\_pos; // Змінна для зберігання позиції сервопривода по осі Z

int grip\_pos; // Змінна для зберігання позиції сервопривода захвату

unsigned long previousMillisStep = 0; // Час останнього виконання функції getSetpoint

unsigned long previousMillisRead = 0; // Час останнього виконання функції ReadPosition

unsigned long currentMillis; // Поточний час

unsigned long previousMillisCheckJson = 0; // Час останнього виконання функції checkForNewJson

const long intervalStep = 3000; // Інтервал між викликами getSetpoint (в мілісекундах)

const long intervalRead = 3000; // Інтервал між викликами ReadPosition (в мілісекундах)

const long intervalCheckJson = 3000; // Інтервал між викликами checkForNewJson (в мілісекундах)

int arrayStep[4][4] = {

{111, 122, 155, 77}, // Початкові позиції для кожного кроку

{140, 120, 150, 70}, // Початкові позиції для кожного кроку

{123, 134, 156, 66}, // Початкові позиції для кожного кроку

{90, 110, 133, 50}, // Початкові позиції для кожного кроку

};

void setup()

{

Serial.begin(9600); // Ініціалізація серійного зв'язку зі швидкістю 9600 бод

servo\_x.attach(11); // Приєднання сервопривода осі X до 11 піна

servo\_y.attach(10); // Приєднання сервопривода осі Y до 10 пін

servo\_z.attach(9); // Приєднання сервопривода осі Z до 9 піна

servo\_grip.attach(5); // Приєднання сервопривода захвату до 5 піна

x\_pos = arrayStep[0][0]; // Ініціалізація позиції X з першого кроку

y\_pos = arrayStep[0][1]; // Ініціалізація позиції Y з першого кроку

z\_pos = arrayStep[0][2]; // Ініціалізація позиції Z з першого кроку

grip\_pos = arrayStep[0][3]; // Ініціалізація позиції захвату з першого кроку

servo\_x.write(x\_pos); // Переміщення сервопривода осі X в початкову позицію

servo\_y.write(y\_pos); // Переміщення сервопривода осі Y в початкову позицію

servo\_z.write(z\_pos); // Переміщення сервопривода осі Z в початкову позицію

servo\_grip.write(grip\_pos); // Переміщення сервопривода захвату в початкову позицію

getSetpoint(); // Виклик функції для ініціалізації позицій сервоприводів

}

void loop()

{

currentMillis = millis(); // Отримання поточного часу

if (currentMillis - previousMillisStep >= intervalStep) // Перевірка, чи пройшов інтервал для getSetpoint

{

previousMillisStep = currentMillis; // Оновлення часу останнього виклику getSetpoint

getSetpoint(); // Виклик функції для зміни позицій сервоприводів

}

if (currentMillis - previousMillisRead >= intervalRead) // Перевірка, чи пройшов інтервал для ReadPosition

{

previousMillisRead = currentMillis; // Оновлення часу останнього виклику ReadPosition

ReadPosition(); // Виклик функції для зчитування поточних позицій сервоприводів

}

if (currentMillis - previousMillisCheckJson >= intervalCheckJson) // Перевірка, чи пройшов інтервал для checkForNewJson

{

previousMillisCheckJson = currentMillis; // Оновлення часу останнього виклику checkForNewJson

checkForNewJson(); // Виклик функції для перевірки нових даних JSON

}

servo\_x.write(x\_pos); // Переміщення сервопривода осі X в початкову позицію

servo\_y.write(y\_pos); // Переміщення сервопривода осі Y в початкову позицію

servo\_z.write(z\_pos); // Переміщення сервопривода осі Z в початкову позицію

servo\_grip.write(grip\_pos); // Переміщення сервопривода захвату в початкову позицію

}

void getSetpoint()

{

x\_pos = arrayStep[step][0]; // Зміна позиції X на основі поточного кроку

y\_pos = arrayStep[step][1]; // Зміна позиції Y на основі поточного кроку

z\_pos = arrayStep[step][2]; // Зміна позиції Z на основі поточного кроку

grip\_pos = arrayStep[step][3]; // Зміна позиції захвату на основі поточного кроку

step++; // Збільшення лічильника кроків

if (step >= stepscount) // Перевірка, чи досягнуто кінця послідовності кроків

{

step = 0; // Скидання лічильника кроків, якщо досягнуто кінця

}

}

void ReadPosition()

{

int x = servo\_x.read(); // Зчитування поточної позиції сервопривода осі X

int y = servo\_y.read(); // Зчитування поточної позиції сервопривода осі Y

int z = servo\_z.read(); // Зчитування поточної позиції сервопривода осі Z

int grip = servo\_grip.read(); // Зчитування поточної позиції сервопривода захвату

Serial.print(x); // Виведення позиції X в серійний порт

Serial.print(" "); // Виведення пробілу

Serial.print(y); // Виведення позиції Y в серійний порт

Serial.print(" "); // Виведення пробілу

Serial.print(z); // Виведення позиції Z в серійний порт

Serial.print(" "); // Виведення пробілу

Serial.println(grip); // Виведення позиції захвату в серійний порт та перехід на новий рядок

}

void checkForNewJson()

{

StaticJsonDocument<200> doc; // Оголошення документа JSON з фіксованим розміром 200 байт

if (Serial.available() > 0) // Перевірка, чи є доступні дані в серійному порті

{

String input = Serial.readString(); // Зчитування даних з серійного порту в рядок

input.trim(); // Видалення будь-яких початкових або кінцевих пробілів

Serial.print("Received input: "); // Виведення отриманих даних в серійний порт

Serial.println(input); // Виведення отриманих даних в серійний порт з переходом на новий рядок

if (input.charAt(input.length() - 1) != ']') // Перевірка, чи рядок закінчується символом ']', якщо ні, то додати його

{

input += "]"; // Додавання символу ']' до кінця рядка

}

DeserializationError error = deserializeJson(doc, input); // Спроба десеріалізації JSON з рядка

if (error) // Перевірка на помилку десеріалізації

{

Serial.print(F("deserializeJson() failed: ")); // Виведення повідомлення про помилку

Serial.println(error.c\_str()); // Виведення опису помилки

switch (error.code()) // Перевірка коду помилки

{

case DeserializationError::InvalidInput: // Якщо помилка через некоректний вхідний рядок

Serial.print(F("Invalid input!")); // Виведення повідомлення про некоректний вхідний рядок

break;

case DeserializationError::NoMemory: // Якщо помилка через недостатню пам'ять

Serial.print(F("Not enough memory")); // Виведення повідомлення про недостатню пам'ять

break;

}

return; // Вихід з функції у разі помилки

}

for (int i = 0; i < 4; i++) // Цикл для оновлення значень масиву arrayStep з отриманого JSON

{

JsonArray innerArray = doc[i]; // Отримання внутрішнього масиву з JSON

if (innerArray.size() != 4) // Перевірка, чи внутрішній масив має 4 елементи

{

Serial.println("Invalid JSON format"); // Виведення повідомлення про некоректний формат JSON

return; // Вихід з функції у разі некоректного формату

}

for (int j = 0; j < 4; j++) // Цикл для оновлення значень кожного елемента в масиві arrayStep

{

arrayStep[i][j] = innerArray[j]; // Присвоєння значення з JSON до масиву arrayStep

}

}

}

}

## Основний

Цей код також обробляє JSON-дані, що надходять через серійний порт для зміни позицій сервоприводів. Основні функції коду:

* **Перевірка часу**: Постійно перевіряє поточний час і викликає відповідні функції через задані інтервали.
* **Оновлення позицій**: Постійно оновлює позиції сервоприводів на основі поточного кроку з масиву. Збільшує лічильник кроків і скидає його після досягнення кінця послідовності.
* **Зчитування позицій**: Зчитує поточні позиції всіх сервоприводів і виводить їх у серійний порт.
* **Обробка JSON**: Зчитує вхідні дані з серійного порту і обробляє їх як JSON. Оновлює масив новими позиціями з отриманого JSON. Перевіряє наявність помилок при десеріалізації JSON і виводить відповідні повідомлення.

#include <Servo.h> // Підключення бібліотеки для керування сервоприводами

#include <ArduinoJson.h> // Підключення бібліотеки для роботи з JSON

Servo servo\_x; // Оголошення об'єкту для керування сервоприводом по осі X

Servo servo\_y; // Оголошення об'єкту для керування сервоприводом по осі Y

Servo servo\_z; // Оголошення об'єкту для керування сервоприводом по осі Z

Servo servo\_grip; // Оголошення об'єкту для керування сервоприводом захвату

int step = 0; // Лічильник кроків для зміни позицій сервоприводів

const int stepscount = 3; // Кількість кроків у послідовності

int x\_pos; // Змінна для зберігання позиції сервопривода по осі X

int y\_pos; // Змінна для зберігання позиції сервопривода по осі Y

int z\_pos; // Змінна для зберігання позиції сервопривода по осі Z

int grip\_pos; // Змінна для зберігання позиції сервопривода захвату

String input; // Змінна для зберігання вхідного JSON рядка

unsigned long previousMillisStep = 0; // Час останнього виконання функції getSetpoint

unsigned long previousMillisRead = 0; // Час останнього виконання функції ReadPosition

unsigned long currentMillis; // Поточний час

unsigned long previousMillisCheckJson = 0; // Час останнього виконання функції checkForNewJson

const long intervalStep = 3000; // Інтервал між викликами getSetpoint (в мілісекундах)

const long intervalRead = 3000; // Інтервал між викликами ReadPosition (в мілісекундах)

const long intervalCheckJson = 3000; // Інтервал між викликами checkForNewJson (в мілісекундах)

int arrayStep[10][4] = { // Масив для зберігання позицій сервоприводів для кожного кроку

{111, 122, 155, 77},

{140, 120, 150, 70},

{123, 134, 156, 66}, // кінцевий step, всі інші для виділення пам'яті

{123, 134, 156, 66},

{123, 134, 156, 66},

{123, 134, 156, 66},

{123, 134, 156, 66},

{123, 134, 156, 66},

{123, 134, 156, 66},

{123, 134, 156, 66},

};

void setup()

{

Serial.begin(9600); // Ініціалізація серійного зв'язку зі швидкістю 9600 бод

servo\_x.attach(11); // Приєднання сервопривода осі X до 11 піна

servo\_y.attach(10); // Приєднання сервопривода осі Y до 10 пін

servo\_z.attach(9); // Приєднання сервопривода осі Z до 9 піна

servo\_grip.attach(5); // Приєднання сервопривода захвату до 5 піна

x\_pos = arrayStep[0][0]; // Ініціалізація позиції X з першого кроку

y\_pos = arrayStep[0][1]; // Ініціалізація позиції Y з першого кроку

z\_pos = arrayStep[0][2]; // Ініціалізація позиції Z з першого кроку

grip\_pos = arrayStep[0][3]; // Ініціалізація позиції захвату з першого кроку

servo\_x.write(x\_pos); // Переміщення сервопривода осі X в початкову позицію

servo\_y.write(y\_pos); // Переміщення сервопривода осі Y в початкову позицію

servo\_z.write(z\_pos); // Переміщення сервопривода осі Z в початкову позицію

servo\_grip.write(grip\_pos); // Переміщення сервопривода захвату в початкову позицію

getSetpoint(); // Виклик функції для ініціалізації позицій сервоприводів

}

void loop()

{

currentMillis = millis(); // Отримання поточного часу

if (currentMillis - previousMillisStep >= intervalStep) // Перевірка, чи пройшов інтервал для getSetpoint

{

previousMillisStep = currentMillis; // Оновлення часу останнього виклику getSetpoint

getSetpoint(); // Виклик функції для зміни позицій сервоприводів

}

if (currentMillis - previousMillisRead >= intervalRead) // Перевірка, чи пройшов інтервал для ReadPosition

{

previousMillisRead = currentMillis; // Оновлення часу останнього виклику ReadPosition

ReadPosition(); // Виклик функції для зчитування поточних позицій сервоприводів

}

checkForNewJson(); // Виклик функції для перевірки нових даних JSON

servo\_x.write(x\_pos); // Переміщення сервопривода осі X в початкову позицію

servo\_y.write(y\_pos); // Переміщення сервопривода осі Y в початкову позицію

servo\_z.write(z\_pos); // Переміщення сервопривода осі Z в початкову позицію

servo\_grip.write(grip\_pos); // Переміщення сервопривода захвату в початкову позицію

}

void getSetpoint()

{

x\_pos = arrayStep[step][0]; // Зміна позиції X на основі поточного кроку

y\_pos = arrayStep[step][1]; // Зміна позиції Y на основі поточного кроку

z\_pos = arrayStep[step][2]; // Зміна позиції Z на основі поточного кроку

grip\_pos = arrayStep[step][3]; // Зміна позиції захвату на основі поточного кроку

step++; // Збільшення лічильника кроків

if (step >= stepscount) // Перевірка, чи досягнуто кінця послідовності кроків

{

step = 0; // Скидання лічильника кроків, якщо досягнуто кінця

}

}

void ReadPosition()

{

int x = servo\_x.read(); // Зчитування поточної позиції сервопривода осі X

int y = servo\_y.read(); // Зчитування поточної позиції сервопривода осі Y

int z = servo\_z.read(); // Зчитування поточної позиції сервопривода осі Z

int grip = servo\_grip.read(); // Зчитування поточної позиції сервопривода захвату

Serial.print(x); // Виведення позиції X в серійний порт

Serial.print(" "); // Виведення пробілу

Serial.print(y); // Виведення позиції Y в серійний порт

Serial.print(" "); // Виведення пробілу

Serial.print(z); // Виведення позиції Z в серійний порт

Serial.print(" "); // Виведення пробілу

Serial.println(grip); // Виведення позиції захвату в серійний порт та перехід на новий рядок

}

void checkForNewJson()

{

StaticJsonDocument<200> doc; // Оголошення документа JSON з фіксованим розміром 200 байт

while (Serial.available() > 0) { // Перевірка наявності даних у серійному порту

input += Serial.readString(); // Зчитування даних з серійного порту та додавання їх до рядка input

}

if (input.length()>30) // Перевірка довжини вхідного рядка для уникнення обробки неповних даних

{

Serial.print("JSON array: "); // Виведення повідомлення про отримання JSON

Serial.println(input); // Виведення отриманого JSON

input.trim(); // Видалення пробілів з початку та кінця рядка

if (input.charAt(input.length() - 1)!= ']') // Перевірка, чи останній символ є ']', щоб переконатися, що JSON завершений

{

input += "]"; // Додавання ']', якщо його немає

}

DeserializationError error = deserializeJson(doc, input); // Спроба десеріалізації JSON з рядка

if (error) // Перевірка на помилку десеріалізації

{

Serial.print(F("deserializeJson() failed: ")); // Виведення повідомлення про помилку

Serial.println(error.c\_str()); // Виведення опису помилки

switch (error.code()) // Перевірка коду помилки

{

case DeserializationError::InvalidInput: // Якщо помилка через некоректний вхідний рядок

Serial.print(F("Invalid input!")); // Виведення повідомлення про некоректний вхідний рядок

break;

case DeserializationError::NoMemory: // Якщо помилка через недостатню пам'ять

Serial.print(F("Not enough memory")); // Виведення повідомлення про недостатню пам'ять

break;

}

return; // Вихід з функції у разі помилки

}

JsonArray jsonArray = doc.as<JsonArray>(); // Перетворення десеріалізованого документа в JSON масив

stepscount = jsonArray.size(); // Оновлення кількості кроків на основі розміру JSON масиву

for (int i = 0;i < jsonArray.size(); i++) // Цикл для обробки кожного кроку з JSON масиву

{

JsonArray innerArray = jsonArray[i]; // Отримання внутрішнього масиву для поточного кроку

if (innerArray.size()!= 4) // Перевірка розміру внутрішнього масиву

{

Serial.println("Invalid JSON format"); // Виведення повідомлення про некоректний формат JSON

return; // Вихід з функції у разі некоректного формату JSON

}

for (int j = 0; j < 4; j++) // Цикл для збереження значень з внутрішнього масиву до arrayStep

{

arrayStep[i][j] = innerArray[j]; // Збереження значення в arrayStep

}

}

input = ""; // Очищення рядка input після обробки JSON

}

}

## Корисні посилання

Ось кілька корисних ресурсів, які можуть допомогти в роботі з Arduino та керуванням сервоприводами:

* [Офіційна документація Arduino](https://docs.arduino.cc/programming/)
* [Arduino Servo Library Reference](https://www.arduino.cc/reference/en/libraries/servo/)
* [ArduinoJson Library](https://arduinojson.org)
* [Wokwi: Arduino Simulator](https://wokwi.com/projects/new/arduino-uno)
* [Telegram Bot з Node-RED (українською)](https://pupenasan.github.io/NodeREDGuidUKR/bots/telegrambot.html)
* [Joystick Controller for MeArm Robot (Arduino Project Hub)](https://projecthub.arduino.cc/utilstudio/joystick-controller-for-mearm-robot-recording-coordinates-892be7)
* [MeArm Controlled with Arduino (Instructables)](https://www.instructables.com/MeArm-Controled-With-Arduino/)

## Посилання на GitHub

Це посилання на наш GitHub, де ви можете знайти весь код та матеріали цієї курсової роботи:

* [GitHub цього курсового проекту](https://github.com/AnastasiaDem/MeARM-Arduino-Uno)