Івано-Франківський національний університет нафти і газу

Інститут інженерної механіки

Кафедра комп'ютеризованого машинобудування

Корбеляк Роман Вікторович

(прізвище, ім'я, по батькові)

УДК 32.816

(індекс)

БАКАЛАВРСЬКА РОБОТА

Оптимізація конструкції мобільного робота за допомогою еволюційних алгоритмів

(назва роботи)

Інженерія мехатронних сиситем

(назва освітньої програми)

131- Прикладна механіка

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня Корбеляк Р.В.

(підпис, ініціали та прізвище здобувача)

Науковий керівник Копей В.Б. доктор техн. наук, професор

(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

Професор Панчук В.Г
(посада) (підпис) (дата) (ініціали та прізвище)

Рецензент

Професор …..
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ
2023

Івано-Франківський національний технічний університет нафти і газу

(повне найменування закладу вищої освіти)

Інститут _Інженерної мфеханіки_____

Кафедра _Комп'ютеризованого машинобудування_____

Освітній рівень_Бакалавр_____

Спеціальність _131- Прикладна механіка_____

(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри _____

_____

«____» _____20__ року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Корбеляку Роману Вікторовичу

(прізвище, ім'я, по батькові)

1. Тема роботи Оптимізація конструкції мобільного робота за допомогою еволюційних алгоритмів

керівник роботи ___Копей В.Б. доктор техн. наук, професор_____,

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від "___"_____ 20__ року №_____

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи: література з питань проектування мобільних роботів, маніпуляторів, механізмів захоплення.

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

Огляд перспективних конструкцій мобільних роботів. Огляд методів оптимізації і еволюційних алгоритмів оптимізації. Області застосувань цих алгоритмів. Розроблення імітаційної моделі подолання роботом довільної перешкоди. Розроблення програми для оптимізації конструкції робота. Оптимізація конструкції за допомогою еволюційних алгоритмів. Розроблення параметричної моделі робота у SOLIDWORKS. Розроблення принципової електричної схеми і керуючої програми робота з можливістю його автономної роботи і керування через UART.

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

Алгоритм оптимізації, результати оптимізації, тривимірні моделі деталей робота, складальне креслення, принципова електрична схема (5 аркушів формату А1)_____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Копей В.Б. доктор техн. наук, професор		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Термін виконання етапів роботи	Примітка
1	Видача завдання		
2	Збір матеріалу, підготовка оглядової частини		
3	Програма для симуляції і оптимізації робота		
4	Проектування конструкції мобільного робота		
5	Розробка програми для керування роботом		
6	Оформлення пояснювальної записки і графічної частини		
7	Відгук керівника дипломного проєкту		
8	Рецензування і перевірка на плагіат		
9	Захист дипломної роботи		

Студент _____ Корбеляк Р.В
(підпис) (прізвище та ініціали)

Керівник роботи _____ Копей В.Б.
(підпис) (прізвище та ініціали)

Анотація

Бакалаврська робота на здобуття освітньо-кваліфікаційного рівня бакалавра на тему: «Оптимізація конструкції мобільного робота за допомогою еволюційних алгоритмів».

Бакалаврська робота складається з пояснювальної записки на __ аркушах формату А4 з додатками, містить __ рисунок та __ таблиць. Графічна частина проєкту містить 5 креслень формату А1.

Виконано аналіз конструкцій мобільних роботів та методів оптимізації їхньої конструкції. На основі Python-пакетів Nodebox for OpenGL і Pymunk розроблено програму для імітаційного моделювання та оптимізації конструкції робота, який повинен якнайшвидше подолати довільну перешкоду. За допомогою алгоритму диференціальної еволюції з пакету SciPy знайдено оптимальні параметри мобільного робота. Розроблено відповідну параметричну модель у SOLIDWORKS, принципову електричну схему та керуючу програму на основі Arduino.

Ключові слова: моделювання, оптимізація, мобільний робот, тривимірне моделювання, Python, Arduino.

Abstract

Bachelor's thesis for obtaining a bachelor's educational and qualification level on the topic: "Design optimization of a mobile robot using evolutionary algorithms".

The bachelor thesis consists of an explanatory note on __ sheets of A4 format with attachments, contains __ figures and __ tables. The graphic part of the project contains 5 drawings in A1 format.

The analysis of the designs of mobile robots and methods of optimizing their design was performed. Based on Python packages Nodebox for OpenGL and Pymunk, a program was developed for simulating and optimizing the design of a robot that must overcome an arbitrary obstacle as quickly as possible. Optimal parameters of the mobile robot were found using the differential evolution algorithm from the SciPy package. A corresponding parametric model in SOLIDWORKS, a schematic electrical diagram and an Arduino-based control program were developed.

Keywords: modeling, optimization, mobile robot, three-dimensional modeling, Python, Arduino.

ЗМІСТ

ВСТУП

Сучасне машинобудування є високотехнологічним, воно включає в себе промислові роботи та робототехнічні системи, які повсюдно інтегруються в різні галузі. Характерною рисою сучасної науково-технічної революції є широке використання роботів у побуті, у сфері виробництва та наукових дослідженнях. Роботи є універсальними пристроями для відтворення рухових та інтелектуальних функцій людини. Одним із важливих класів їх є мобільні роботи.

Практично метою створення роботів є передача їм тих видів діяльності, які для людини трудомісткі, важкі, монотонні, шкідливі для здоров'я і життя. Це насамперед:

- допоміжні виробничі операції (завантаження та вивантаження установок, верстатів, автоматів);

- основні виробничі операції (зварювання, фарбування, різання, збирання тощо);

- роботи у так званих екстремальних умовах (під водою, у космосі, у радіоактивних та отруйних середовищах).

Роботи застосовуються для комплексної автоматизації виробництва, зростання продуктивність праці, поліпшення якості продукції. Від традиційних засобів автоматизації промислові роботи відрізняються універсальністю, можливістю їхньої швидкої переналагодження, що дозволяє створювати на базі універсального обладнання роботизовані технологічні комплекси, гнучкі автоматизовані виробництва. В результаті розвитку робототехніки людство отримує можливість вирішувати принципово нові наукові та виробничі завдання.

На відміну від стаціонарних, більшість мобільних роботів відрізняються розмірами, можливістю пересування. Щоб мобільний робот працював продуктивно та коректно мобільному роботу необхідна надійна конструкція, електронні компоненти, що довго працюють, і точна система управління.

Метою роботи є оптимізація конструкції мобільного робота за допомогою еволюційних алгоритмів. Для досягнення мети потрібно розв’язати наступні задачі:

- виконати аналіз перспективних конструкцій мобільних роботів;

- виконати огляд методів оптимізації та обґрунтувати застосування еволюційних алгоритмів;

- розробити імітаційну модель для оптимізації конструкції мобільного робота і провести саму оптимізацію;

- спроектувати конструкцію мобільного робота зі знайденими оптимальними параметрами;

- розробити принципову електричну схему і керуючу програму робота.

1 ОГЛЯД КОНСТРУКЦІЙ МОБІЛЬНИХ РОБОТІВ

1.1 Колісна платформа

Багато мобільних роботів, особливо ті, що призначені для переміщення по плоскій поверхні, мають колісну конструкцію. Вони можуть мати два, три, чотири або навіть більше коліс, які дозволяють їм рухатися вперед, назад, вліво та вправо. Така конструкція дозволяє роботам швидко переміщатися та маневрувати.

Колісна платформа є одним з найпоширеніших типів конструкцій для мобільних роботів. Вона використовує колеса для руху та маневрування роботом. Розглянемо детальніше складові колісної конструкції:

- Типи коліс: колісні платформи можуть мати різні типи коліс в залежності від вимог і призначення робота. Існують звичайні колеса з пневматичними або гумовими шинами, які забезпечують гладкий рух по рівній поверхні. Також використовуються спеціалізовані колеса, такі як гусеничні колеса для кращої прохідності або м'які колеса для амортизації ударів.

- Кількість коліс: мобільні роботи з колісною платформою можуть мати різну кількість коліс. Найпоширеніші варіанти - двоколісні і чотириколісні платформи, але також можуть використовуватись більш складні конфігурації з більшою кількістю коліс для забезпечення стабільності та маневреності.

- Рушійна система: для приведення коліс у рух використовуються різні рушійні системи. Найпоширеніші варіанти - електричні мотори або гідравлічні приводи. Електричні мотори зазвичай живляться від акумуляторів і забезпечують електричну енергію для руху коліс. Гідравлічні приводи використовують рідину під високим тиском для переміщення коліс.

- Керування: колісні платформи можуть мати різні системи керування. Вони можуть бути керовані за допомогою джойстика, радіокерування або програмного забезпечення, що дозволяє автономне пересування.

Прикладом колісної платформи є автокари для перевезення вантажів в складських приміщеннях. За часту складські приміщення використовуються з максимальною вигодою, тому проміжки між стелажами є мінімальними, саме тому

конструкція та розміри автокарів мають бути компактними, але забезпечувати максимальну маневреність.

Сучасні автокари мають класичну будову з поворотним механізмом направляючих коліс, що знаходяться на задній осі. Така конструкція забезпечує малий радіус розвороту, але для максимальної маневреності потрібно переміщатися в ліво та право без розвороту самої платформи, така задача не підсильна класичній чотирьох колісній базі з однією поворотною віссю. Для розв’язання цієї задачі потрібно використати інший підхід до конструкції колеса та автокара в цілому, а саме використати колеса Ілона. Автокари сьогодні використовують електричну тягу, так як використовуються в закритих приміщеннях, та мають повний привід, тому задача з використанням колеса Ілона є перспективною.

Колесо Ілона або шведське колесо — колесо з рівномірно розподіленими по ободу роликами, через які відбувається взаємодія колеса із поверхнею переміщення. Таке колесо дає змогу транспортному засобу шляхом незалежного приведення у рух своїх коліс забезпечити контрольоване переміщення у будь-якому напрямку. Загальний вигляд розробленої омніколісної платформи зображено на рисунку 1.1 [1]

Багато направлені колеса вже багато років використовуються в робототехніці, в промисловості та в логістиці. Основним джерелом омніколіс є компанії які виробляють їх для все направлених конвеєрних систем. Все направлені колеса популярні для роботів. Все направлений робот може рухатись по прямій від точки А до точки В, обертаючись уздовж лінії для того, щоб прибути з правильною орієнтацією. Тому дані платформи добре себе зарекомендували себе в виробництві, як рухома основа для маніпуляторів та для орієнтації заготовок для їхньої подальшої обробки. Для удосконалення конструкції та для розширення можливостей, платформа може бути обладнана GPS навігаторам та гіроскопом. Такі датчики збільшать точність переміщення та дозволять задавати маршрут руху дистанційно.

Також дана платформа потребує системи яка здатна контролювала оточуючу ситуацію. Така система повинна контролювати наближення платформи до перешкоди або людини та перешкоджати зіткненню. Такі удосконалення дозволять використовувати омніколісні платформи у будь-якому виробництві та збільшити продуктивність переміщення вантажів та виробництва загалом. [1]

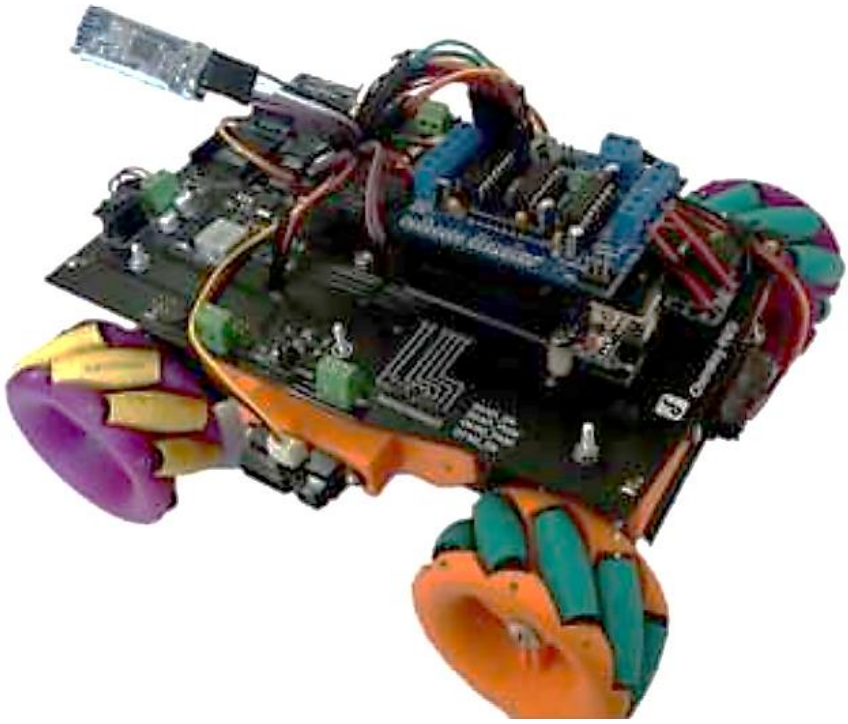


Рисунок 1.1 - Загальний вигляд омніколісної платформи

1.2 Гусенична платформа

Деякі мобільні роботи, які призначені для руху по нерівним або складним поверхнях, можуть мати гусеничну конструкцію. Гусениці забезпечують кращу стабільність та зчеплення з поверхнею, дозволяючи роботу пройти через перешкоди, такі як нерівності, камені або піски. Мобільні роботи на гусеничній платформі використовуються в різних галузях і мають широкий спектр застосувань. Гусенична платформа надає переваги у теренних умовах, де колісні транспортні засоби можуть бути обмежені.

Приклади застосування мобільних роботів на гусеничній платформі:

- Експлуатація у важкодоступних місцях: Гусеничні роботи можуть використовуватися для досягнення місць, куди важко або неможливо дістатися колісними транспортними засобами. Наприклад, вони можуть бути використані для ремонту трубопроводів або ліній електропередач, обстеження складних ландшафтів або проведення рятувальних операцій у важкодоступних місцях.

- Військове застосування: Гусеничні роботи широко використовуються військовими для розвідки, патрулювання, доставки вантажів та бойової підтримки. Вони можуть працювати в різних умовах, включаючи гірські райони, болота або снігові покриви (рисунок 1.2).

Рисунок 1.2 – Застосування гусеничної платформи на прикладі танка Leopard 2

- Гірничодобувна промисловість: Гусеничні роботи використовуються у гірничодобувній промисловості для транспортування матеріалів і обладнання, видалення відсічних порід та виконання робіт у складних умовах шахт або кар'єрів.

- Сільське господарство: У сільському господарстві гусеничні роботи можуть використовуватися для засівання полів, збирання врожаю, обробки землі або доставки сільськогосподарської техніки на важкодоступні ділянки (рисунок 1.3).

Рисунок 1.3 – Застосування гусеничної платформи у сільськогосподарських потребах (John Deere 9RX)

- Будівництво та ремонт інфраструктури: Гусеничні роботи можуть бути використані для будівництва та ремонту доріг, мостів, залізничних шляхів та інших інженерних споруд. Вони забезпечують стабільність і маневреність у важких умовах будівництва.

Гусенична платформа є одним з типів ходових систем, що використовуються в мобільних роботах. Вона складається з низки з'єднаних гусениць, які прокладені навколо рухомого об'єкта, такого як робот або транспортний засіб. Гусениці зазвичай складаються з металевих пластин, з'єднаних гумовими або металевими ланками.

Гусенична платформа складається з таких основних елементів (рисунок 1.4): основною складовою даного типу платформ є гусениці: Гусениці - це набір металевих чи гумових пластин, з'єднаних гумовими або металевими ланками. Вони утворюють "ланцюг" або "змійку", яка обгортається навколо колісної пари або ролика на рухомому об'єкті. Гусениці забезпечують контакт з поверхнею та передають рух об'єкту.

Колісні пари або ролики, які розташовані вздовж гусениць. Вони допомагають підтримувати та керувати гусеницями, дозволяючи рухатися вперед, назад та повороти. Колісні пари або ролики можуть бути приводними або відновлювальними, залежно від конструкції платформи.

Ходова система гусеничної платформи включає механізм для керування рухом гусениць. Вона може включати гідроприводи, електроприводи або механічні системи, які забезпечують рух та маневреність платформи. Рама є структурою, на якій розташована гусенична платформа та інші компоненти робота. Вона забезпечує жорсткість та міцність всієї конструкції, що дозволяє платформі переносити вантажі та працювати в умовах, вимагають стійкості. Приводний механізм забезпечує рух гусениць. Він може включати двигуни, гідравлічні або електричні системи, що перетворюють енергію в рух гусениць.

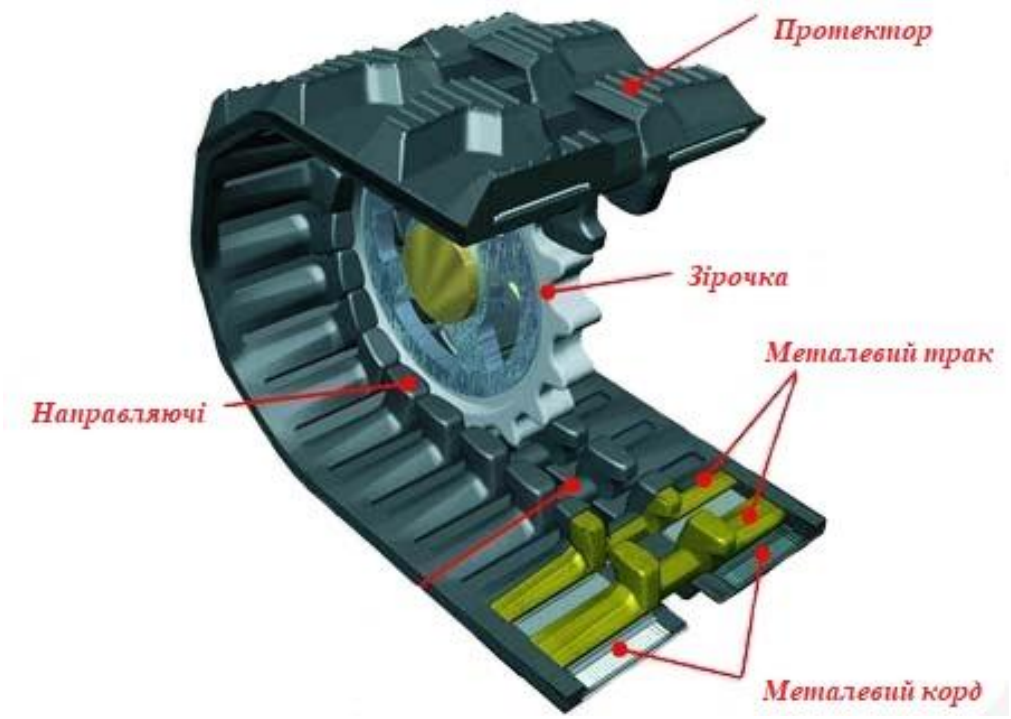


Рисунок 1.4 – Будова гусениці

Основні переваги гусеничної платформи:

- Прохідність: Гусенична платформа забезпечує високу прохідність у різних умовах, включаючи нерівний, м'який або складний рельєф. Гусениці розподіляють вагу робота по більшій площі, допомагаючи уникнути провалів, підскакування або застрягання.

- Стабільність: Гусенична платформа забезпечує високу стабільність робота, особливо на нерівних або крутих поверхнях. Це дозволяє роботу ефективно працювати навіть у ситуаціях, коли колісна платформа може бути нестабільною або нездатною до руху.

- Вантажопідйомність: Гусенична платформа здатна витримувати велику вагу завдяки рівномірному розподілу тиску по грунту. Це робить її ідеальною для перевезення важких вантажів або обладнання.

- Маневреність: Хоча гусенична платформа може бути менш маневрена, ніж колісна, вона все ж здатна повертати на місці або керувати вузькими проходами. Деякі гусеничні роботи також оснащені системами повороту гусениць, що дозволяють їм здійснювати повороти з більшою легкістю.

- Амортизація: Гусенична платформа може забезпечувати кращу амортизацію і знижувати вібрацію при русі по нерівній поверхні. Це допомагає зберегти цілісність робота та зменшити вплив вібрацій на вантаж або обладнання.

Гусенична платформа має свої обмеження, зокрема, повільнішу швидкість руху порівняно з колісною платформою та вищий рівень шуму. Однак, вона є дуже ефективним вибором для роботів, що працюють у важких умовах, де потрібна велика прохідність та стабільність.

1.3 Крокуючі роботи

Крокуючі роботи. Особливий розділ робототехніки складають крокуючі системи пересування та засновані на них транспортні машини. Вони є предметом робототехніки тому, що механічні ноги – педипулятори (від латинського слова pes, pedis – нога) – найбільш близькі до іншого основного об'єкту робототехніки – маніпуляторів. Однак значення і потенціал у галузі застосування крокуючих систем машин виходять за межі робототехніки.

Спосіб пересування за допомогою ніг (крокування, біг, стрибання), як відомо, є найбільш поширеними в живій природі. Однак у техніці він ще не отримав помітного застосування через складність управління. Такі роботи можуть мати дві

або більше ніг і використовувати їх для ходьби, бігу, лазіння або навіть сходження по сходах.

Розвиток робототехніки створив необхідну науково-технічну основу для реалізації цього принципово нового для техніки способу пересування та створення нового типу транспортних машин – крокуючих (рисунок 1.5).

Крокуючий спосіб представляє основний інтерес для руху по заздалегідь непідготовленою місцевістю із перешкодами. Традиційні колісні та гусеничні транспортні машини залишають за собою безперервну колію, витрачаючи на це значно більшу енергію, ніж у разі пересування кроками, коли взаємодія з ґрунтом відбувається лише у місцях упору стопи. Крім цього, крокуючий спосіб пересування має і більшу прохідністю на пересіченій місцевості до можливості пересуватися стрибками, долати перешкоди тощо. При крокуючому способі менше руйнується ґрунт, що, наприклад, важливо у природоохоронних зонах. При рух по досить гладким і підготовленим поверхням цей спосіб поступається колісному в економічності, швидкості пересування та простоті управління [2].

Рисунок 1.5 – Крокуючі роботи компанії Boston Dynamics

Існує багато типів конструкцій для мобільних роботів, включаючи колісні платформи, гусеничні платформи, ноги та дроноподібні роботи. Кожен з цих типів має свої переваги та використання залежно від конкретних умов та завдань.

Проте колісна платформа є однією з найуніверсальніших та широко використовуваних конструкцій. Вона дозволяє роботам швидко переміщатися по плоскій поверхні, маневрувати в різних напрямках і досягати високої швидкості. Колісна платформа також легко налаштовується для різних завдань та може бути використана в різних сферах, включаючи промисловість, медицину, дослідження та багато інших.

Отже, колісна платформа може бути розглянута як універсальна та широко використовувана конструкція для мобільних роботів, що демонструє її популярність і придатність у багатьох сферах діяльності. Саме тому дана платформа була обрана ,як база для реалізації теми бакалаврської роботи та вирішення поставлених ним завдань з оптимізації її конструкції мобільного робота.

2 ОГЛЯД АЛГОРИТМІВ ОПТИМІЗАЦІЇ

2.1 Оптимізація в математиці

Оптимізацією в математиці називають задачу знаходження екстремуму функції f(x) в певній області значень її аргументів x. Можуть шукати мінімум або максимум функції f(x). В техніці математичні методи оптимізації застосовують для пошуку оптимальних технічних рішень (наприклад оптимальної конструкції). Оптимізація буває [5]:

- локальна, в якій шукають локальний екстремум функції f(x);

- глобальна, в якій шукають глобальний екстремум.

Усі методи оптимізації за ознакою детермінованість-стохастичність можна поділити на:

- детерміновані, в яких не застосовують випадкові елементи;

- стохастичні, в яких застосовують випадкові елементи;

- комбіновані, в яких є і детерміновані елементи і стохастичні.

За порядком обчислюваної похідної усі методи оптимізації можна поділити на:

- прямі, в яких обчислюються значення функції f(x), але не її похідної;

- першого порядку, в яких обчислюється перша похідна. Ще їх називають градієнтними;

- другого порядку, в яких обчислюється друга похідна.

2.2 Алгоритми оптимізації

Існує кілька методів оптимізації функції, і вибір методу залежить від конкретних характеристик функції та задачі оптимізації. Приклади поширених методів оптимізації функцій:

Градієнтний спуск: Градієнтний спуск — це ітеративний алгоритм оптимізації, який має на меті знайти мінімум функції шляхом ітеративного

коригування параметрів на основі від’ємного градієнта (нахилу) функції в кожній точці. Він часто використовується в задачах, де функція є диференційованою, а її градієнт легко доступний.

Метод Ньютона: метод Ньютона — це ще один ітеративний алгоритм оптимізації, який використовує першу та другу похідні функції для знаходження її мінімуму. Він може збігатися швидше, ніж градієнтний спуск, але вимагає обчислення других похідних і може бути більш інтенсивним з точки зору обчислень.

Спряжений градієнт: Спряжений градієнт — це ітеративний метод оптимізації, який підходить для широкомасштабних задач оптимізації без обмежень. Він поєднує в собі переваги градієнтного спуску та методу Ньютона, уникаючи необхідності обчислення других похідних, водночас досягаючи швидшої збіжності.

Методи квазіньютона: методи квазіньютона апроксимують матрицю Гессе (другі похідні) функції за допомогою перших похідних. Вони є обчислювально ефективною альтернативою методу Ньютона і можуть досить добре сходитися в багатьох випадках.

Генетичні алгоритми. Генетичні алгоритми — це алгоритми оптимізації, натхненні процесом природного відбору. Вони використовують такі методи, як мутація, схрещування та відбір, щоб розвивати популяцію потенційних рішень протягом кількох поколінь з метою пошуку оптимального рішення.

Імітація відпалу: імітація відпалу — це алгоритм стохастичної оптимізації, який імітує процес відпалу в металургії. Він починається з початкового рішення та ітеративно досліджує простір рішень, дозволяючи «погані» рухи з певною ймовірністю. З часом ймовірність прийняття гірших рішень зменшується, що призводить до конвергенції до кращого рішення.

Оптимізація роєм частинок: оптимізація роєм частинок (PSO) — це метод оптимізації на основі популяції, коли група частинок переміщується в просторі пошуку, регулюючи свої позиції на основі їхньої найкращої відомої позиції та

найкращої позиції, знайденої будь-якою частинкою в групі. Він особливо ефективний для безперервних і багатовимірних задач оптимізації.

Еволюційні стратегії: еволюційні стратегії — це алгоритми оптимізації, які використовують популяцію потенційних рішень і застосовують принципи еволюційної біології, такі як мутація, рекомбінація та відбір. Вони часто використовуються в задачах, де простір пошуку великий, а інформація про градієнти недоступна.

Байєсова оптимізація: байєсовська оптимізація — це послідовний метод оптимізації на основі моделі, який використовує імовірнісні моделі для моделювання цільової функції та ітеративно вибирає нові точки для вибірки на основі функції збору даних. Він ефективний для задач оптимізації чорної скриньки, де оцінка цільової функції є дорогою [3].

Вибір методу залежить від різних факторів, таких як характеристики проблеми, доступність інформації про градієнт, розмірність проблеми та доступні обчислювальні ресурси.

2.3 Суть еволюційних алгоритмів оптимізації

Суть алгоритмів генетичної та еволюційної оптимізації полягає в тому, що вони базуються на принципах природного відбору та еволюції для вирішення задач оптимізації. Ці алгоритми імітують процес природної еволюції, коли найбільш пристосовані особини мають більше шансів вижити та розмножуватися, що з часом призводить до створення кращих рішень.

У генетичних та еволюційних алгоритмах оптимізації популяція кандидатів на рішення розвивається протягом кількох поколінь для пошуку оптимального рішення даної проблеми. Ось розбивка їхньої суті:

Представлення рішень: у цих алгоритмах рішення задачі оптимізації кодуються як індивідууми в сукупності. Представлення може змінюватися залежно від проблеми, і воно може бути простим, як двійковий рядок, або більш складною структурою. Люди представляють потенційні рішення в просторі пошуку.

Оцінка придатності: кожна особа в популяції оцінюється за допомогою функції придатності, яка кількісно визначає, наскільки добре вона виконує задачу

оптимізації. Функція придатності оцінює якість рішення, закодованого індивідом, на основі об’єктивних критеріїв проблеми. Особи з вищими показниками фітнесу вважаються кращими рішеннями.

Відбір (добір). Процес відбору базується на принципі «виживає найпристосованіший». Особи з вищими показниками пристосованості мають більшу ймовірність бути відібраними для відтворення, імітуючи ідею, що кращі рішення з більшою ймовірністю сприятимуть наступному поколінню. Для вибору особин для розмноження можна використовувати різні методи відбору, такі як турнірний відбір або вибір колеса рулетки.

Розмноження: розмноження передбачає створення нових особин, яких часто називають нащадками, з відібраних особин. Цей процес зазвичай включає такі генетичні оператори, як схрещування та мутація. Схрещування поєднує генетичний матеріал двох батьківських особин для створення нових рішень, імітуючи генетичну рекомбінацію. Мутація вносить невеликі випадкові зміни в потомство, сприяючи дослідженню простору пошуку.

Еволюція та повторення: нове потомство замінює деяких особин у поточній популяції, утворюючи наступне покоління. Процес відбору, відтворення та заміни повторюється протягом багатьох поколінь. Кожна ітерація представляє нову популяцію рішень, де найбільш підготовлені особи з попереднього покоління мають вищі шанси бути відібраними та передати свій генетичний матеріал наступному поколінню.

Конвергенція: протягом ітерацій популяція еволюціонує, і через механізм природного відбору та генетичних операторів відкриваються кращі рішення. Алгоритм спрямований на зближення до оптимального рішення, де придатність популяції більше не покращується суттєво або відповідає попередньо визначеному критерію завершення.

Алгоритми генетичної та еволюційної оптимізації особливо ефективні для проблем оптимізації, коли простір пошуку великий, ландшафт складної або зашумленої або коли інформація про градієнт недоступна. Вони можуть ефективно

досліджувати простір пошуку, виконувати глобальну оптимізацію та працювати з мультимодальними або нелінійними цільовими функціями[4].

Використовуючи принципи природного відбору та еволюції, ці алгоритми забезпечують потужну структуру оптимізації, застосовну до широкого кола проблемних областей, включаючи інженерне проектування, планування, аналіз даних і машинне навчання.

2.4 Область застосування еволюційних алгоритмів оптимізації

Алгоритми генетичної та еволюційної оптимізації добре підходять для різних сценаріїв і типів проблем. Ось деякі ситуації, коли генетичні та еволюційні алгоритми особливо корисні:

Недиференційовані функції або функції чорного ящика: Генетичні та еволюційні алгоритми можуть вирішувати проблеми оптимізації, де цільова функція є недиференційованою, розривною або не має відомої аналітичної форми. Ці алгоритми не вимагають явного знання похідних і можуть оптимізувати функції виключно на основі їхніх оцінок.

Великі та складні простори пошуку: коли простір пошуку великий, складний або містить дискретні змінні, генетичні та еволюційні алгоритми можуть ефективно досліджувати простір рішень. Вони здатні обробляти багатовимірні проблеми з великою кількістю змінних, що може бути складним для методів на основі градієнта.

Глобальна оптимізація. Генетичні й еволюційні алгоритми добре підходять для вирішення проблем глобальної оптимізації, де метою є пошук найкращого рішення в усьому просторі пошуку, а не локального оптимуму. Ці алгоритми мають можливість виконувати широкий пошук, уникаючи потрапляння в пастку локальних оптимумів.

Багатомодальна оптимізація: якщо проблема оптимізації має кілька піків або кілька дійсних рішень, генетичні та еволюційні алгоритми можуть досліджувати простір пошуку, щоб знайти кілька рішень. Популяційний характер цих алгоритмів дозволяє їм підтримувати різноманітність і досліджувати різні регіони одночасно.

Комбінаторна та дискретна оптимізація: Генетичні та еволюційні алгоритми можуть ефективно вирішувати задачі комбінаторної оптимізації, де рішення передбачають вибір підмножини або впорядкування послідовності дискретних елементів. Ці алгоритми здатні досліджувати та оцінювати різні комбінації або перестановки змінних.

Зашумлені або стохастичні цільові функції: Генетичні та еволюційні алгоритми можуть вирішувати проблеми оптимізації, де оцінки цільової функції зашумлені або схильні до випадкових флуктуацій. Завдяки ітераційній вибірці та зміні генеральної сукупності ці алгоритми можуть ефективно переходити до хороших рішень навіть за наявності шуму.

Компроміс дослідження та експлуатації: Генетичні та еволюційні алгоритми встановлюють баланс між дослідженням та використанням простору пошуку. Вони мають можливість досліджувати нові регіони на ранніх стадіях оптимізації, одночасно зближуючись до перспективних регіонів у міру просування оптимізації.

Оптимізація без аналітичного градієнта: Генетичні та еволюційні алгоритми не потребують аналітичних градієнтів цільової функції. Це робить їх придатними для проблем, де обчислення градієнтів може бути обчислювально дорогим або аналітично складним [6].

Хоча генетичні та еволюційні алгоритми мають свої сильні сторони у вищезгаданих сценаріях, важливо зазначити, що вони можуть бути не найефективнішим або ефективним вибором для всіх проблем оптимізації. Для гладких і добре поведених функцій із відомими похідними, градієнтні методи, такі як градієнтний спуск або метод Ньютона, можуть забезпечити швидшу збіжність. Крім того, обчислювальна складність і час, необхідні для конвергенції, можуть бути вищими для генетичних і еволюційних алгоритмів порівняно з іншими методами оптимізації.

Вибір алгоритму оптимізації зрештою залежить від конкретних характеристик проблеми, наявних ресурсів і компромісів між розвідкою та розробкою. Часто буває корисно випробувати різні алгоритми та порівняти їх ефективність у конкретній проблемі, щоб визначити найбільш прийнятний підхід.

2.5 Функція differential_evolution

Функція differential_evolution з пакету SciPy [7] знаходить глобальний мінімум функції багатьох аргументів:

differential_evolution(func, bounds, args=(), strategy=best1bin, maxiter=1000, popsize=15, tol=0.01, mutation=(0.5, 1), recombination=0.7, seed=None, callback=None, disp=False, polish=True, init=latinhypercube, atol=0, updating=immediate, workers=1)

Алгоритм диференційної еволюції має стохастичний характер, тобто він не використовує градієнтні методи для пошуку мінімуму і може застососуватись для функцій з великою кількістю аргументів, але часто потребує більшої кількості викликів функції, аніж звичайні градієнтні методи. Алгоритм запропонований Сторном і Прайсом [8].

Параметри функції [7]:

func : функція, яка мінімізується. Повинна бути у вигляді f(x, *args), де x – це аргументи у вигляді одновимірного масиву, а args – це необов’язковий кортеж додаткових константних параметрів.

bounds : послідовність, яка визначає границі значень змінних. Пари (min, max) для кожного елементу в x, які визначають нижню і верхню границі.

args : необов’язковий кортеж для функції func.

strategy : необов’язковий рядок, що визначає стратегію еволюції ('best1bin', 'best1exp', 'rand1exp', 'randtobest1exp', 'currenttobest1exp', 'best2exp', 'rand2exp', 'randtobest1bin', 'currenttobest1bin', 'best2bin', 'rand2bin', 'rand1bin'). Значення за замовчуванням: 'best1bin'.

maxiter : необов’язкове ціле, що визначає максимальну кількість поколінь для еволюції популяції. Максимальна кількість викликів функції (без «полірування») визначається за формулою: (maxiter + 1) * popsize * len(x)

popsize : необов’язкове ціле, що визначає розмір популяції. Популяція має popsize * len(x) індивідів.

tol : необов’язкове дійсне, що визначає відносну точність для збіжності. Розрахунок завершується, коли np.std(pop) <= atol + tol * np.abs(np.mean(population_energies)), де `atol` і `tol` є абсолютною і відносною точністю відповідно.

mutation : необов’язковий кортеж дійсних (float, float). Константа мутації. У літературі також відома як диференціальна вага, позначається F. Якщо вказано як число з плаваючою комою, то має бути в діапазоні [0, 2]. Якщо вказано як кортеж (min, max), використовується дизерінг. Дізерінг випадково змінює константу мутації від покоління до покоління. Константа мутації для цього покоління береться з U[min, max). Змішування може значно прискорити конвергенцію. Збільшення константи мутації збільшує радіус пошуку, але сповільнює конвергенцію.

recombination : необов’язкове дісне. Константа рекомбінації повинна бути в діапазоні [0, 1]. У літературі також відома як ймовірність кросинговеру, яка позначається CR. Збільшення цього значення дозволяє більшій кількості мутантів переходити в наступне покоління, але під загрозою стабільності популяції.

seed : необов’язкове ціле або `np.random.RandomState`. Якщо `seed` не вказано, використовується синглтон `np.RandomState`. Якщо `seed` є int, то використовується новий екземпляр `np.random.RandomState`, заповнений початковим значенням. Якщо `seed` вже є екземпляром `np.random.RandomState`, тоді використовується цей екземпляр `np.random.RandomState`. Укажіть `seed` для повторюваної мінімізації.

disp : необов’язкове булеве. Показує повідомлення.

callback : необов’язкова функція, `callback(xk, convergence=val)` Функція для відстеження прогресу мінімізації. xk — поточне значення x0. val представляє дробове значення конвергенції сукупності. Коли значення більше одиниці, функція зупиняється. Якщо зворотний виклик повертає `True`, тоді мінімізація припиняється (будь-яке полірування все ще може бути виконано).

polish : необов’язкове булеве. Якщо True (за замовчуванням), тоді `scipy.optimize.minimize` з методом `L-BFGS-B` використовується для «полірування» (покращення) найкращого члена сукупності в кінці, що може трохи покращити мінімізацію.

init : необов’язковий рядок або масив. Укажіть, який тип ініціалізації популяції виконується. Має бути одним із: 'latinhypercube', 'random' - масив із зазначенням початкової сукупності. Масив повинен мати форму (M, len(x)), де len(x) – кількість параметрів. `init` обрізається до `bounds` перед використанням. Типовим є `latinhypercube`. Вибірка `latinhypercube` намагається максимізувати охоплення доступного простору параметрів. 'random' ініціалізує популяцію випадковим чином – але є недолік, що може статися кластеризація, що запобігає охопленню всього простору параметрів. Використання масиву для вказівки підмножини генеральної сукупності може використовуватися, наприклад, для створення вузької групи початкових припущень у місці, де, як відомо, існує рішення, таким чином скорочуючи час для конвергенції.

atol : необов’язкове дійсне. Абсолютна точність для конвергенції, розв’язання припиняється, коли np.std(pop) <= atol + tol * np.abs(np.mean(population_energies)), де `atol` і `tol` є абсолютною і відносною точністю відповідно.updating : {'immediate', 'deferred'}, optional Якщо 'immediate', то вектор найкращого рішення постійно оновлюється протягом одного покоління. Це може призвести до швидшої конвергенції, оскільки пробні вектори можуть використовувати переваги безперервного вдосконалення найкращого рішення. З 'deferred' вектор найкращого рішення оновлюється один раз на покоління. Лише 'deferred' сумісний із розпаралелюванням, а ключове слово «workers» може замінити цей параметр.

workers : необов’язкове ціле або «map-like» функція. Якщо `workers` є int, то сукупність підрозділяється на секції `workers` і оцінюється паралельно (використовується `multiprocessing.Pool`). Використовуйте -1 для використання всіх доступних ядер ЦП. В якості альтернативи надайте функцію виклику, подібну

до «map-like», наприклад `multiprocessing.Pool.map` для паралельної оцінки популяції.

Функція повертає:

res : OptimizeResult - результат оптимізації, представлений як об’єкт OptimizeResult. Важливими атрибутами є: x-масив рішення, успішність, логічне значення, що вказує, чи оптимізатор завершив успішно, і повідомлення, яке описує причину завершення. Перегляньте `OptimizeResult` для опису інших атрибутів. Якщо було використано `polish`, і «поліруванням» було отримано нижчий мінімум, тоді OptimizeResult також містить атрибут jac.

Примітка. Диференціальна еволюція — це стохастичний метод на основі популяції, який корисний для задач глобальної оптимізації [9]. При кожному проходженні популяції алгоритм змінює кожне рішення-кандидат, змішуючи з іншими рішеннями-кандидатами, щоб створити пробний кандидат. Існує кілька стратегій [10] для створення пробних кандидатів, які відповідають деяким проблемам більше, ніж іншим. Стратегія «best1bin» є хорошою відправною точкою для багатьох систем. У цій стратегії випадковим чином вибираються два члени популяції. Їх різниця використовується для зміни найкращого члена:

b' = b_0 + mutation * (population[rand0] - population[rand1])

Потім будується пробний вектор. Починаючи з випадково вибраного 'i'-го параметра, спроба послідовно заповнюється (за модулем) параметрами з b' або початкового кандидата. Вибір між використанням b' або вихідним кандидатом здійснюється за допомогою біноміального розподілу ('bin' у 'best1bin') – генерується випадкове число в [0, 1). Якщо це число менше константи `recombination`, тоді параметр завантажується з b', інакше він завантажується з вихідного кандидата. Кінцевий параметр завжди завантажується з b'. Після створення кандидата для випробування оцінюється його придатність. Якщо випробування краще, ніж початковий кандидат, воно займає своє місце. Якщо він також кращий за найкращого загального кандидата, він також замінює його. Щоб покращити ваші шанси знайти глобальний мінімум, використовуйте вищі значення `popsize`, з вищими значеннями `mutation` і (dithering), але меншими значеннями

`recombination`. Це збільшує радіус пошуку, але сповільнює конвергенцію. За замовчуванням вектор найкращого рішення постійно оновлюється протягом однієї ітерації (updating='immediate'). Це модифікація [11] оригінального алгоритму диференціальної еволюції, яка може призвести до швидшої конвергенції, оскільки пробні вектори можуть негайно отримати вигоду від покращених рішень. Щоб використовувати оригінальну поведінку Storn і Price, оновлюючи найкраще рішення один раз за ітерацію, установіть updating='deferred'.

Еволюційні алгоритми є потужним інструментом для розв'язання складних оптимізаційних задач, які можуть мати велику кількість можливих розв'язків та нелінійних або недиференційованих функцій цілі. Ці алгоритми базуються на природних механізмах еволюції та генетики, що дозволяє їм ефективно працювати в просторі великого обсягу та з великою кількістю можливих розв'язків [12].

Сутність еволюційного алгоритму відбувається у моделюванні процесу природного відбору, схрещування та мутації. Починаючи з початкової популяції розв'язків, алгоритм покращує розв'язки в кожній поколінні, використовуючи функцію пристосованості для оцінки якості розв'язків. Схрещування та мутація допомагають забезпечити генетичну різноманітність і дослідження різних областей простору параметрів, що погіршує наявність оптимальних розв'язків.

Один із найбільш важливих аспектів еволюційних алгоритмів – це їх здатність знаходити глобальні оптимуми в просторі параметрів. Це можливо за допомогою використання стохастичних методів, які дозволяють алгоритму "перестрибувати" через локальні мінімуми та досліджувати різні області простору розв'язків. Плюсом еволюційних алгоритмів є їхній широкий спектр програм.

Незважаючи на свою потужність, еволюційні алгоритми також мають свої обмеження і виклики. Одним з найважливіших аспектів є вибір правильних параметрів алгоритму, таких як розмір популяції, ймовірності схрещування та мутації, а також критерії зупинки. Неправильно вибрані параметри можуть призвести до поганої збіжності або преждевременної зупинки алгоритму.

Крім того, еволюційні алгоритми можуть вимагати значних обчислювальних ресурсів та тривалого часу для знаходження оптимального розв'язку, особливо в

складних задачах з великою кількістю параметрів. Покращення швидкості та ефективності алгоритмів є активним напрямком досліджень.

У майбутньому еволюційні алгоритми можуть продовжувати розвиватися та застосовуватися в нових галузях. Використання машинного навчання та штучного інтелекту може сприяти покращенню ефективності та автоматизації процесу оптимізації. Крім того, комбінація еволюційних алгоритмів з іншими методами оптимізації може привести до створення нових потужних інструментів для розв'язання складних задач.

Загалом, еволюційні алгоритми відіграють важливу роль у сфері оптимізації, дозволяючи знаходити глобальні оптимуми в складних просторах параметрів. Їхній потенціал використовується в різних галузях та їхній розвиток продовжується завдяки постійним дослідженням та інноваціям.

3 ІМІТАЦІЙНА МОДЕЛЬ ДЛЯ ОПТИМІЗАЦІЇ КОНСТРУКЦІЇ РОБОТА

3.1 Принципи розроблення алгоритму оптимізації

Генетичні та еволюційні алгоритми можуть бути застосовані для підвищення прохідності мобільного робота, який стикається з різними перешкодами, шляхом оптимізації його контролю або поведінки. Ось загальний підхід до застосування генетичних та еволюційних алгоритмів у цьому контексті:

Визначте проблему: Чітко визначте проблему прохідності для мобільного робота. Це включає визначення цільової функції, яка може базуватися на таких показниках, як успішна навігація, витрачений час, енергоефективність або будь-які інші відповідні критерії. Визначте обмеження та вимоги, наприклад типи перешкод, які повинен подолати робот. У нашому випадку робот повинен якнайшвидше подолати перешкоду, яка являє собою купу випадково розташованих і з випадковим розміром цеглин на прямій поверхні (рис. 3.1).

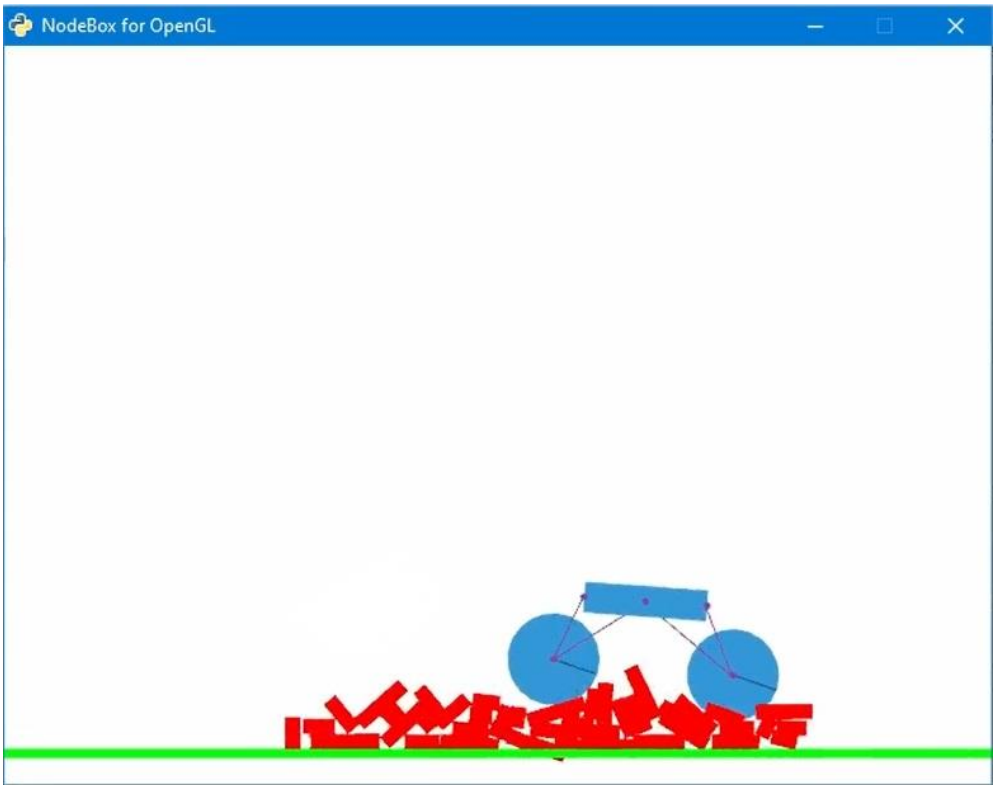


Рисунок 3.1 – Імітація подолання роботом перешкоди

Кодування: розробіть відповідну схему кодування для представлення керування або поведінки робота. Це кодування може містити параметри, які керують рухом робота, наприклад швидкість, радіус повороту або стратегії уникнення перешкод. Кодування повинно дозволяти варіації та поєднання різних стратегій контролю. Параметри робота, які оптимізуються (рис. 3.2):

- радіус заднього колеса;
- радіус переднього колеса;
- відстань між колесами (мінімальна);
- виліт заднього колеса;
- виліт переднього колеса.

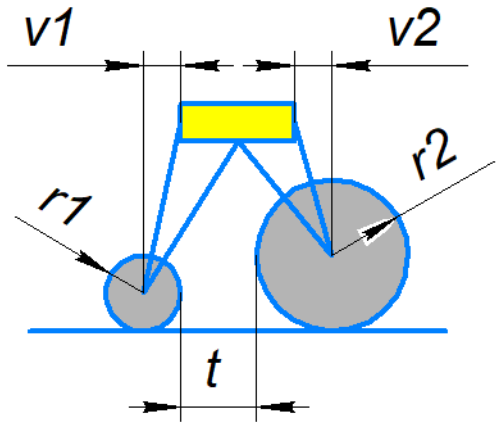


Рисунок 3.2 – Параметри для оптимізації

Фітнес-функція (цільова функція): визначте фітнес-функцію, яка оцінює продуктивність робота на основі цілей прохідності. Ця функція кількісно визначає, наскільки добре робот долає перешкоди та досягає бажаної мети. Фітнес-функція може враховувати такі фактори, як кількість успішно подоланих перешкод, пройдену відстань або плавність траєкторії робота. У нашому випадку цільова функція має аргументи, які є параметрами робота. Функція повертає значення часу в секундах, за який робот долає перешкоду.

Генетичні оператори: реалізуйте такі генетичні оператори, як кросовер і мутація, щоб створити нових особин (стратегії керування роботами) у кожному поколінні. Схрещування поєднує генетичний матеріал двох батьківських особин

для створення нащадків, що дозволяє обмінюватися бажаними ознаками. Мутація вносить невеликі випадкові зміни, щоб досліджувати нові можливості. Ці оператори полегшують дослідження та використання простору рішень. Ми будемо використовувати значення за замовчуванням, які рекомендує функція differential_evolution.

Ініціалізація популяції: ініціалізуйте популяцію індивідуумів (стратегії керування роботом) випадковим чином або використовуючи знання домену. Населення має бути достатньо різноманітним, щоб досліджувати різні підходи до подолання перешкод.

Відбір: Застосуйте механізм відбору, щоб вибрати особин із популяції для розмноження на основі їхніх значень придатності. Особи з вищою фізичною підготовленістю, що вказує на кращі результати в подоланні перешкод, з більшою ймовірністю будуть обрані. Можна використовувати різні методи відбору, такі як вибір турніру або вибір колеса рулетки.

Еволюція та ітерація: виконуйте еволюційний процес, постійно створюючи нові покоління особин за допомогою застосування генетичних операторів і відбору. Кожна ітерація представляє нову сукупність стратегій керування. З часом популяція еволюціонує, і фізична підготовленість особин має покращитися, коли будуть виявлені кращі стратегії подолання перешкод.

Критерії завершення: Визначте критерії завершення для процесу оптимізації. Це може бути максимальна кількість поколінь, певний досягнутий рівень придатності або конвергенція значень придатності. Алгоритм припиняє роботу, коли критерії виконуються. Критерієм завершення виберемо кількість ітерацій.

Оцінка та тестування: Оцініть найкращу особу (осіб) із кінцевої сукупності за набором сценаріїв валідації або тестування в реальному світі. Оцініть їх ефективність і при необхідності відрегулюйте параметри керування.

Завдяки ітераційній еволюції та оптимізації стратегій керування мобільного робота з використанням генетичних та еволюційних алгоритмів можна з часом покращити прохідність і навігаційні можливості робота через перешкоди.

Алгоритм заохочує виявлення ефективних стратегій уникнення перешкод і може адаптуватися до різних конфігурацій перешкод і середовищ.

Важливо зазначити, що успіх цього підходу залежить від таких факторів, як якість функції відповідності, вибір кодування та генетичних операторів, а також представлення можливостей і обмежень робота. Для досягнення задовільних результатів часто необхідні ітераційні експерименти та вдосконалення.

3.2 Програма на основі Nodebox і Pymunk

Розроблено програму для імітації подолання роботом перешкоди і оптимізації конструкції. Програма побудована на основі Python-пакетів Nodebox for OpenGL (простий пакет 2D-графіки) і Pymunk (симулятор 2D-фізики).

Нижче описано код програми. Спочатку виконується імпорт необхідних модулів:

- nodebox - для реалізації графіки;
- pymunk - для реалізації фізики;
- інші додаткові модулі.

```
#encoding: utf-8
from __future__ import division


from nodebox.graphics import *
import pymunk
import pymunk.pyglet_util
import random, time
```

Функція «create_moto(x):» призначена для створення візка робота, де x - це параметри

```
def create_moto(x):
    """Створює візок. Індекси списку параметрів x:
    0 - радіус заднього колеса
    1 - радіус переднього колеса
```

```
    2 - відстань між колесами (мінімальна)
    3 - виліт заднього колеса
    4 - виліт переднього колеса
    """
```

Процес створення наступний: першим чином створюється заднє колесо де вказується відповідні параметри

```
        y0=100 # висота осей над землею
            l=x[0]+x[1]+x[2] # відстань між осями
            b0 = pymunk.Body() # заднє колесо
            b0.position = 0-x[3],y0
            g0=pymunk.Circle(b0, x[0])
            g0.mass = 100
            g0.friction = 1
```

Таким самим чином створюється переднє колесо:

```
    b1 = pymunk.Body() # переднє колесо
    b1.position = l+x[4],y0
    g1=pymunk.Circle(b1, x[1])
    g1.mass = 100
    g1.friction = 1
```

Наступним етапом є створення корпусу робота та потрібних з’єднань:

```
    b2 = pymunk.Body() # корпус
    b2.position = l/2,y0+max([x[0],x[1]])+15
    g2 = pymunk.Poly.create_box(b2, size=(l,20))
    g2.mass = 100
    g2.friction = 0


# з'єднання:
    J=(
```

```
    pymunk.PinJoint(b0, b2, (0, 0), (0, 0)),
    pymunk.PinJoint(b1, b2, (0, 0), (0, 0)),
    pymunk.PinJoint(b0, b2, (0, 0), (-l/2, 0)),
    pymunk.PinJoint(b1, b2, (0, 0), (l/2, 0)))
```

Після створення об’єктів додаємо їх у простір симуляції та повертаємо їх:

```
    space.add(b0, b1, b2, g0, g1, g2, *J) # додати в space
    return (b0, b1, b2, g0, g1, g2)+J
```

За допомогою функції «create_poly» створюється випадкова цеглина (тіло) і задається її позиція x,y і швидкість x1,y1. Створюється геометрія даного тіла. Задається маса, тертя та колір. Після чого об’єкт додається в простір для симуляції.

```
def create_poly(x,y,x1,y1):
    """Створює випакову цеглину з позицією x,y і швидкістю x1,y1"""
    body = pymunk.Body()
    body.position = x,y
    body.velocity= x1,y1
    poly = pymunk.Poly.create_box(body, size=(random.randint(10,40),10))
    poly.mass = 10
    poly.friction = 1
    poly.color = (255, 0, 0, 255) # червоний колір (R,G,B,A)
    space.add(body, poly)
```

Функція «create_static» створює нерухоме тіло (дорога/площадка). Спершу створюється статичне тіло після чого задається його позиція. Створюється його геометрія (лінія за точками p1, p2, товщиною 3) задається тертя та колір. Після чого тіло та геометрія додається в простір для симуляції.

```
def create_static(pos=(300, 100), p1=(-200, 0), p2=(300, 0)):
    """Створює нерухоме тіло - лінію за точками p1, p2"""
    body = pymunk.Body(body_type = pymunk.Body.STATIC)
```

```
    body.position = pos
    line = pymunk.Segment(body, p1, p2, 3)
    line.friction = 1
    line.color = (0, 255, 0, 255)
    space.add(body, line)
```

Функція «draw» - це функція пакету nodebox, що рисує кожен кадр анімації. Перш за все оголошуються глобальні змінні: tm – час, space – простір, B – список усіх об’єктів. Зарисовується перший кард білим кольором та виводиться значення часу.

Якщо час більший за 10 секунд або робот проїхав далеко або натиснуто клавішу «a», то симуляція припиняється.

Далі виконуємо симуляцію фізики за допомогою функції «step», збільшуємо змінну часу на 0,05 секунди та вказуємо кутову швидкість заднього та переднього коліс, після рисуємо усі об’єкти, що знаходяться у просторі.

Після чого функція повторюється, поки не завершиться.

```
def draw(canvas):
    """Функція пакету nodebox, що рисує кожен кадр анімації"""
    global tm, space, B # глобальні змінні
    background(1) # зарисувати попередній кадр
    print tm
    if tm>10 or B[1].position[0]>600 or canvas.keys.char=="a":
        canvas.stop()


    space.step(0.05) # симуляція фізики
    tm+=0.05 # збільшити
    B[0].angular_velocity= -5 # швидкість колеса
    B[1].angular_velocity= -5
    space.debug_draw(draw_options) # рисувати усі об'єкти
```

Тут вказується розміри вікна - 700 на 500 пікселів:

```
canvas.size = 700, 500 # розміри вікна
```

f(x) – цільова функція мінімізації. Повертає час tm, за який візок з параметрами x подолав перешкоду.

Створюється вікно. Задається змінна часу. Створюється простір для симуляції. Задається гравітація по осі Y.

Створюється дорога, а також 50 цеглин із випадковими розмірами по координатами.

Створюється візок з параметрами та розпочинається анімація.

Після зупинки дана функція повертає значення часу tm.

```
def f(x):
    """Функція, що мінімізується. Повертає час tm, за який візок з параметрами x подолав перешкоду"""
    global tm, space, draw, canvas, B # глобальні змінні
    canvas=Canvas()
    tm=0
    space = pymunk.Space() # створити простір
    space.gravity = 0,-981 # гравітацію
    create_static(pos=(0, 10), p1=(-200, 10), p2=(700, 10)) # дорогу
    for i in range(50):
        create_poly(random.randint(300,400), 200, 0, 0) # цеглини

    B = create_moto(x) # візок з параметрами x
    canvas.run(draw) # розпочати анімацію

    for obj in B:
        obj.position=1000,1000
        space.remove(obj) # видалити усі об'єкти
    return tm
```

Одним із варіантів пошуку мінімуму значень параметрів, може бути класичний квазі-ньютонівський метод "L-BFGS-B", що знаходиться у пакеті «scipy» :

```
print minimize(f, x0=[25., 25, 17, 10, 10], method="L-BFGS-B", bounds=[(20,
30),(20, 30),(5, 30),(0, 20),(0, 20)])
```

Але як можна побачити нижче, що ця функція не справляється з даною задачею. Адже у нашій задачі багато параметрів і є стохастичність. Результати:

```
     fun: 10.050000000000008
 hess_inv: <5x5 LbfgsInvHessProduct with dtype=float64>
     jac: array([        0.        , -10000000.00000014,         0.        ,
             0.        ,         0.        ])
  message: 'ABNORMAL_TERMINATION_IN_LNSRCH'
     nfev: 126
      nit: 0
   status: 2
  success: False
        x: array([25., 25., 17., 10., 10.])
```

Тому ми використовуємо алгоритм диференціальної еволюції. Вказуємо, що потрібно знайти мінімум функції f(x), задаємо межі зміни параметрів (ріадіуси коліс, відстань між колесами та виліт коліс) . Ставимо обмежене число ітерації в нашому випадку це 10:

```
print differential_evolution(f, bounds=[(20, 30),(20, 30),(5, 30),(0, 20),(0, 20)],
maxiter=10) # знайти мінімум
```

Отримуємо наступні результати:

```
fun: 5.249999999999989
 message: 'Maximum number of iterations has been exceeded.'
    nfev: 981
```

```
    nit: 10
success: False
      x: array([29.66240257, 29.94598132, 20.04745796, 14.4104782 , 18.85834041])
```

Отже оптимальна конструкція (рисунок 3.2) робота буде мати такі значення параметрів:

- Радіус заднього колеса - 29.66240257
- Радіус переднього колеса - 29.94598132
- Відстань між колесами - 20.04745796
- Виліт заднього колеса - 14.4104782
- Виліт переднього колеса - 18.85834041

Після знаходження оптимальних параметрів мобільного робота було досліджено вплив параметрів на час подолання перешкоди t і отримано наступні залежності наведені у рисунка нижче. Ці залежності були отримані шляхом зміни одного параметра, в той час як інші мають оптимальне значення. З залежностей видно, що найбільш чіткий вплив на t мають радіуси коліс. У першу чергу переднє колесо.

З графіка залежності часу подолання перешкоди t від радіуса заднього колеса (рисунок 3.3) видно, що якщо радіус малий, умовно 10 мм., то робот не може проїхати через перешкоду за 10 с., тобто даний радіус нам не підходить, але при значеннях 25-27мм. робот починає швидко долати перешкоду. Отже з графіка видно, що більший радіус колеса забезпечує мінімальний час подолання перешкоди.

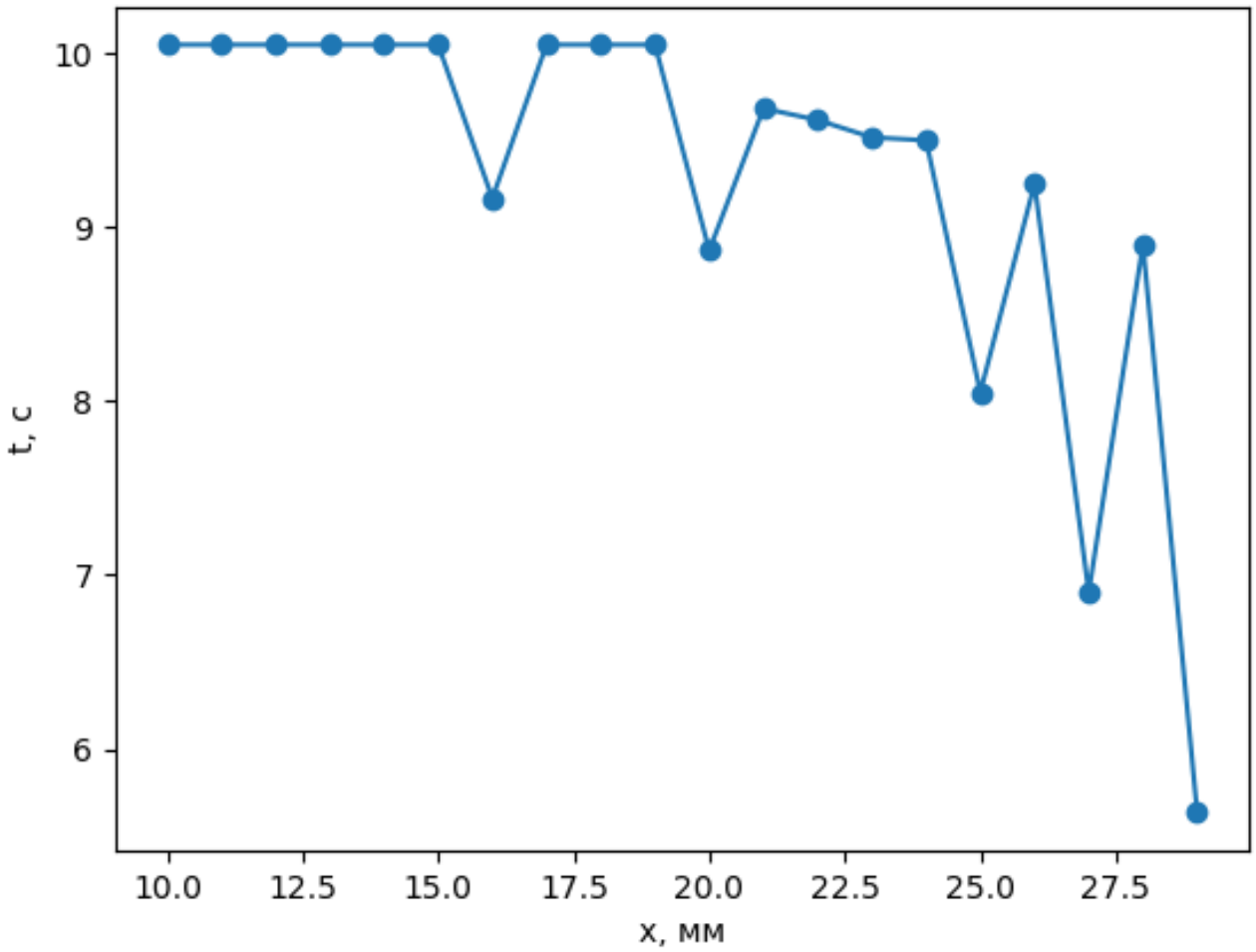


Рисунок 3.3 – Залежність часу подолання перешкоди t від радіуса заднього колеса

На графіку залежності часу подолання перешкоди t від радіуса переднього колеса (рисунок 3.4) ситуація схожа з заднім колесом. Більший радіус колеса забезпечує більшу швидкість подолання перешкоди.

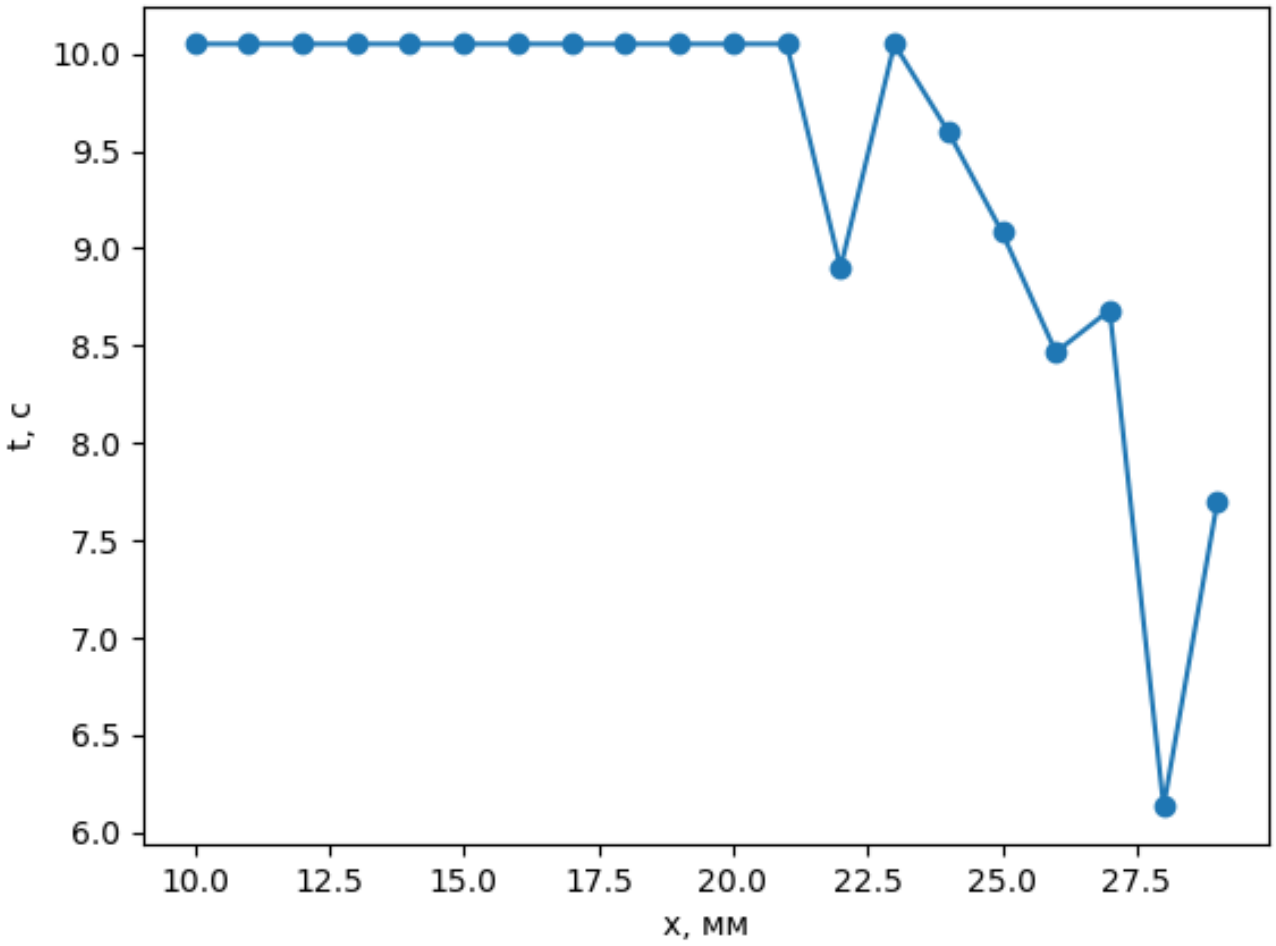


Рисунок 3.4 – Залежність часу подолання перешкоди t від радіуса переднього колеса

Залежність часу подолання перешкоди t від радіуса відстані між колесами (рисунок 3.5). Ця відстань не впливає на час подолання перешкоди так сильно, як радіуси коліс, проте все рівно на графіку видно, що існує певний мінімум у діапазоні 13-16 мм, тому можна обрати 15 мм, як оптимальне значення.

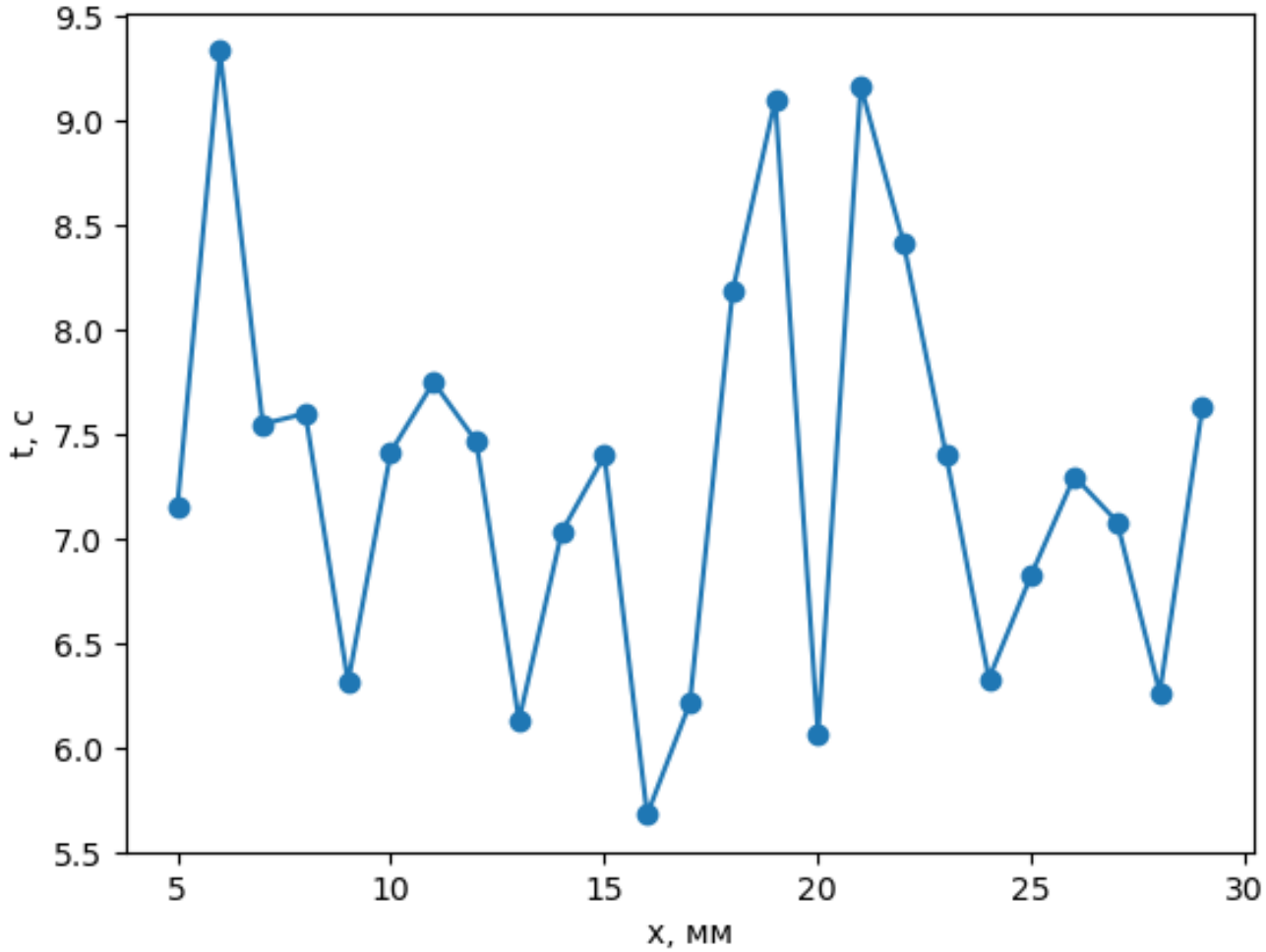


Рисунок 3.5 – Залежність часу подолання перешкоди t від радіуса відстані між колесами

Вплив вильоту заднього колеса на час подолання перешкоди (рисунок 3.6) відслідкувати складно, проте за графіком видно, що коли це параметр більший то це все ж краще. А от виліт переднього колеса (рисунок 3.7) має бути в межах 15-16 мм.

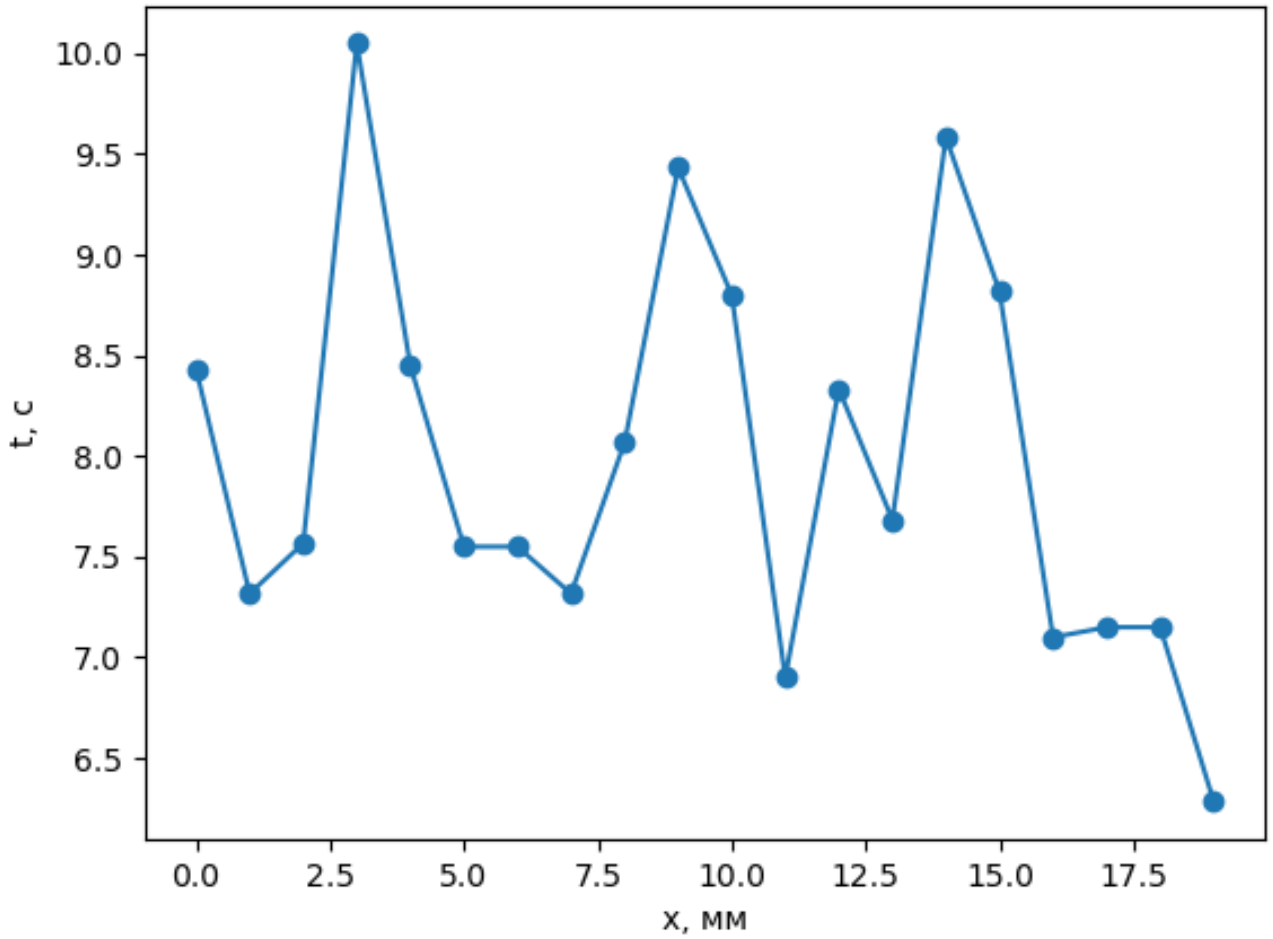


Рисунок 3.6 – Залежність часу подолання перешкоди t від вильоту заднього колеса

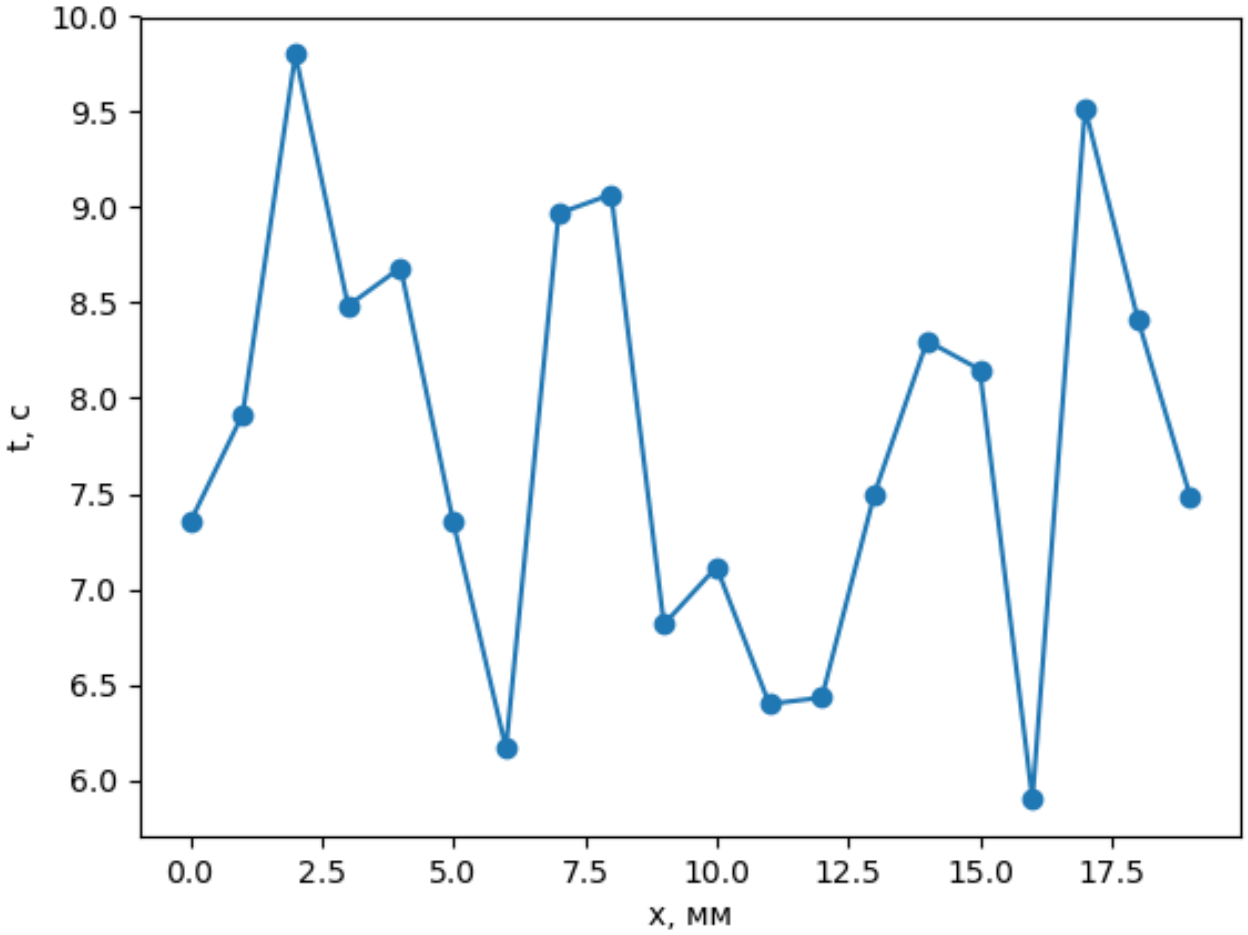


Рисунок 3.7 – Залежність часу подолання перешкоди t від вильоту переднього колеса

За графіками можна зробити висновок, що найбільший вплив на швидкість подолання перешкод роботом мають радіуси коліс, в першу чергу радіус переднього колеса, який може бути дещо більшим за радіус заднього колеса.

4 ПРОЕКТУВАННЯ МОБІЛЬНОГО РОБОТА

4.1 Параметрична модель робота у SolidWorks

Параметричну модель робота створюємо у SolidWorks. Програма Solidworks – це програмний комплекс системи автоматизованого проєктування і розрахунку (САПР) для автоматизації робіт підприємства на етапах розробки конструкторської та технологічної підготовки виробництва [13, 14].

Проєктування конструкції робота розпочинаємо зі створення платформи (рисунок 4.1) до якої вже будуть кріпитися усі необхідні деталі та елементи. Створення даної деталі просте та не вимагає значних зусиль, адже для створення необхідно тільки обрати одну із базових площин та побудувати на ній ескіз, задавши на ньому усі необхідні розміри, включаючи розміщення пазів для зміни положення вильоту осей робота та прямокутні отвори, через які в подальшому будуть прокладатися контакти для керування та живлення двигунів.

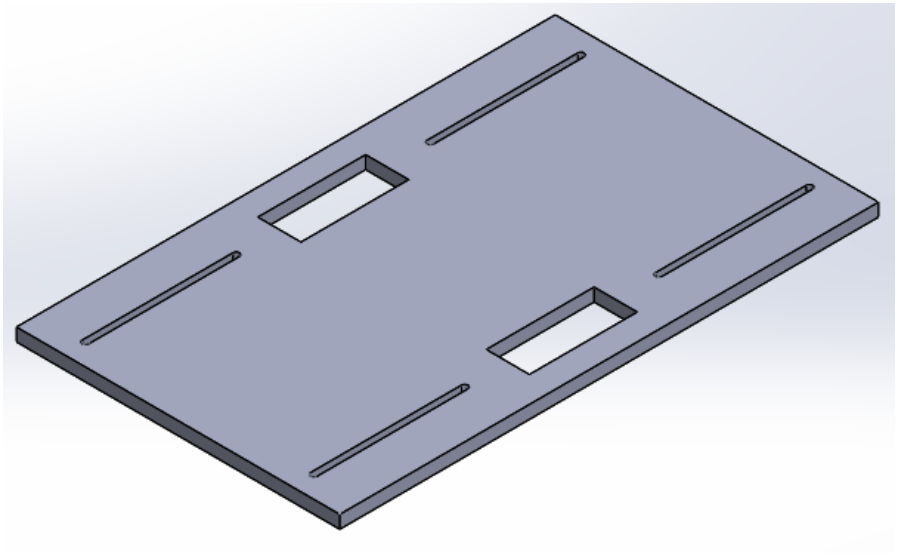


Рисунок 4.1 – Платформа робота

На рисунку 4.2 зображено 3-D модель деталі кріплення. На дану деталь кріпиться двигун. А за допомогою пари отворів на площадці деталі дана деталь кріпиться на основу робота де передбачено паз, в рамках якої можна переміщувати кріплення, що дозволяє нам корегувати один із параметрів робота, які описані вище. А саме відстань між колесами (виліт осей).

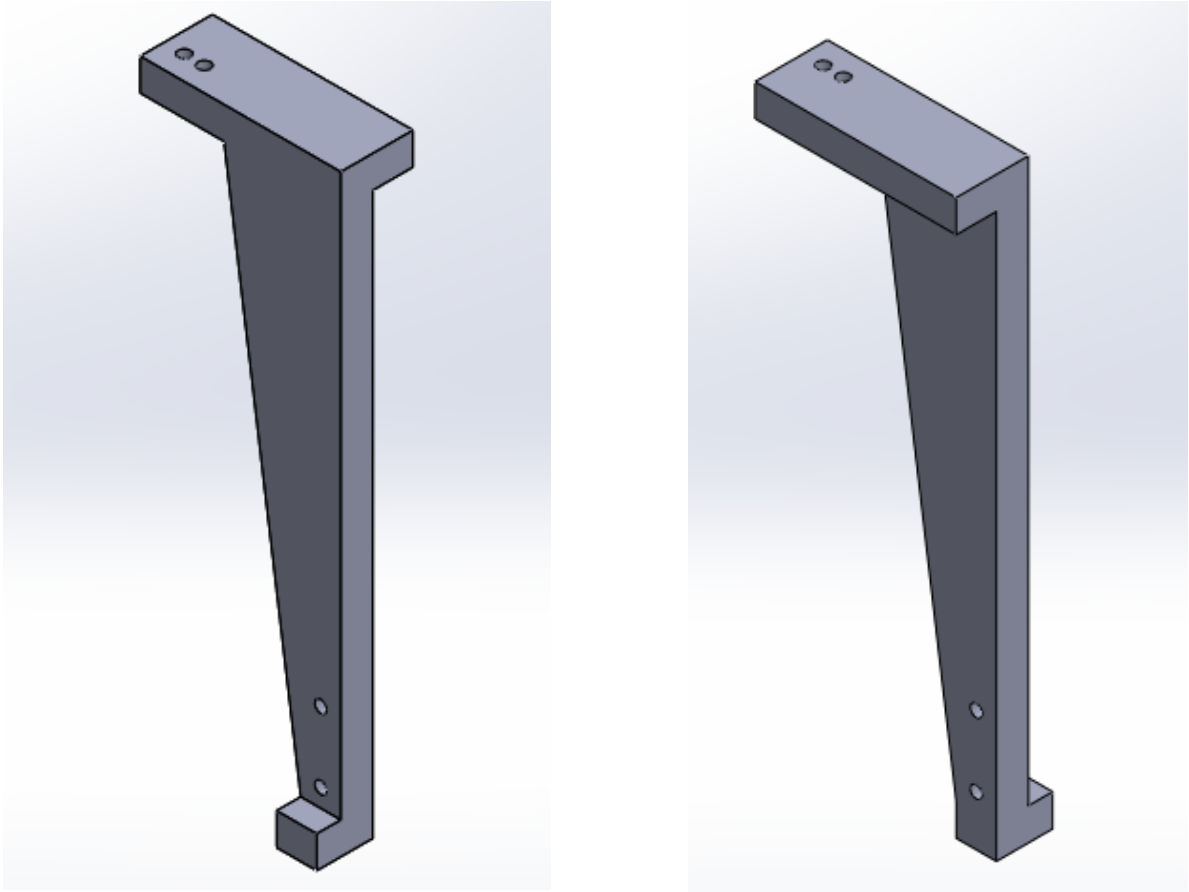


Рисунок 4.2 – 3-D модель деталі кріплення

Оскільки двигун DC Gear 48 (рисунок 4.3) є стандартизованою деталлю, то його модель можна взяти з відповідної бібліотеки чи завантажити готову модель з мережі Інтернет [15].

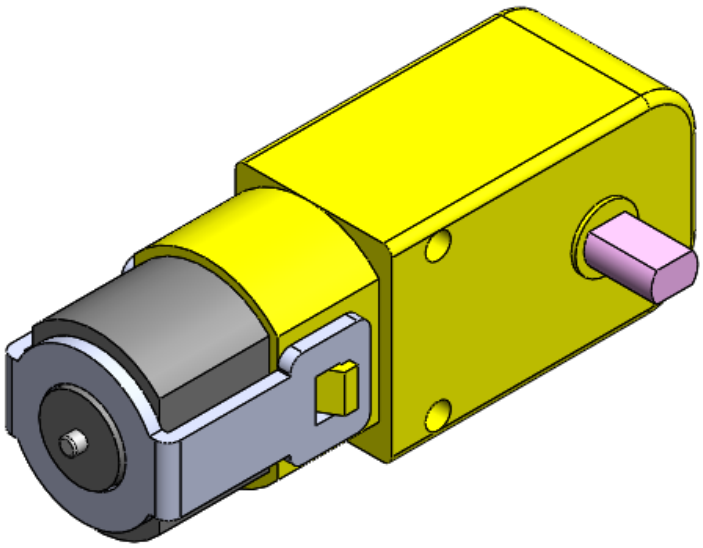


Рисунок 4.3 – Двигун DC Gear 48

Модель колеса потрібно розробляти самостійно, щоб забезпечити необхідній діаметр коліс та їх пружність.

Моделювання колеса починаємо із вибору базової площини, на якій створюємо ескіз (рисунок 4.4), задаючи необхідні нам розміри зовнішнього діаметру .

За допомогою інструменту «видавлювання» придаємо об’єм ескізу та отримуємо тверде тіло (рисунок 4.5).

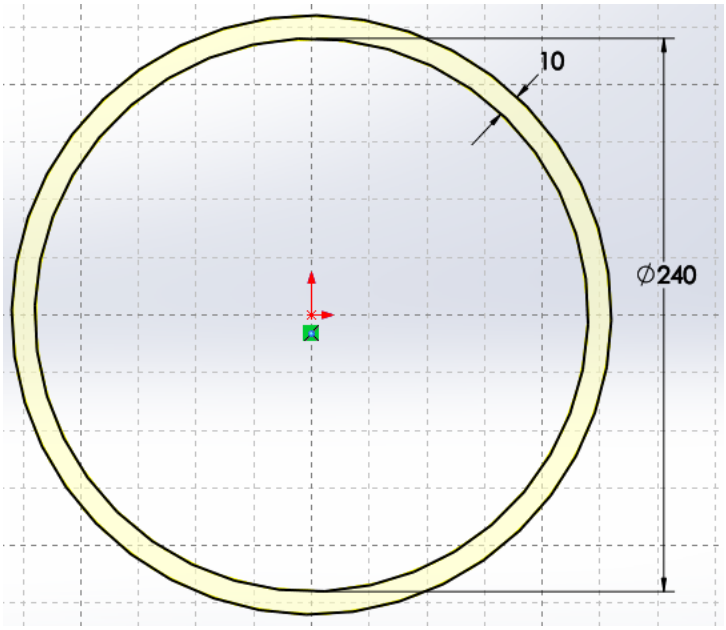


Рисунок 4.4 – Початковий ескіз колеса

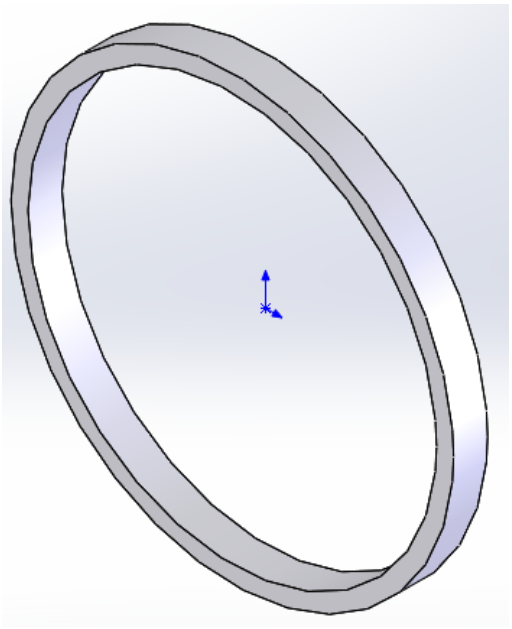
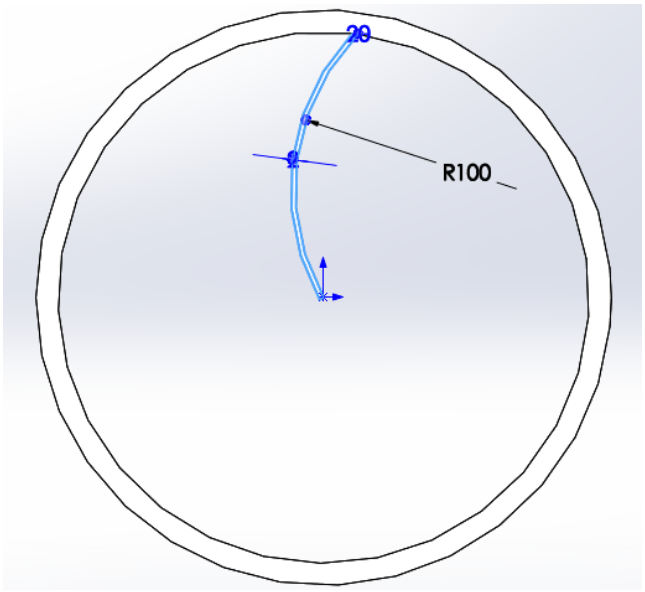


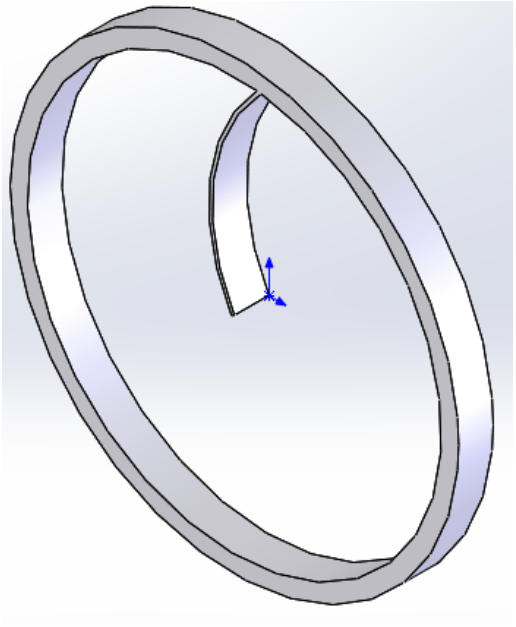
Рисунок 4.5 – Вигляд моделі колеса після видавлювання ескізу

За допомогою вище описаних функцій ми могли контролювати та задавати необхідний діаметр колеса.

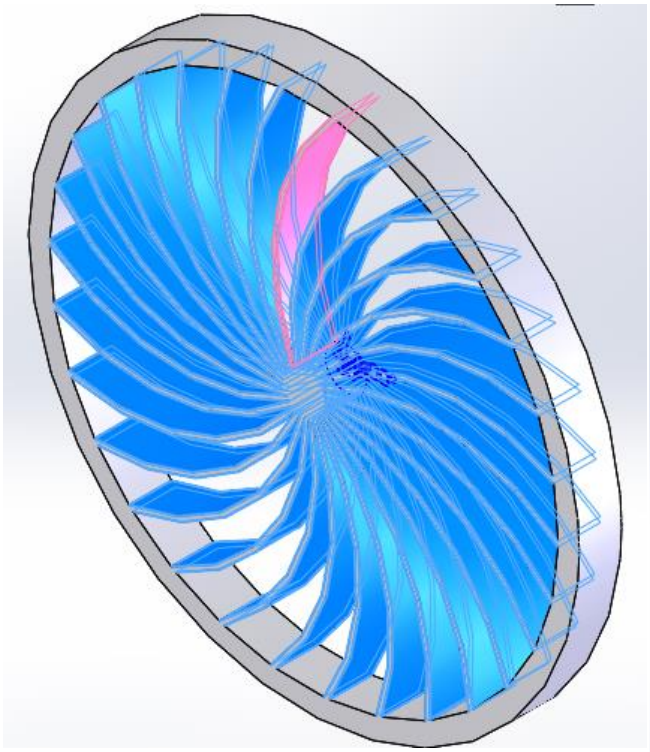
У даному пункті розглянемо елемент, за допомогою якого можна змінювати пружність колеса. Створюємо одну ланку, після чого здійснюємо видавлювання та «масив по колу» (рисунок 4.6).



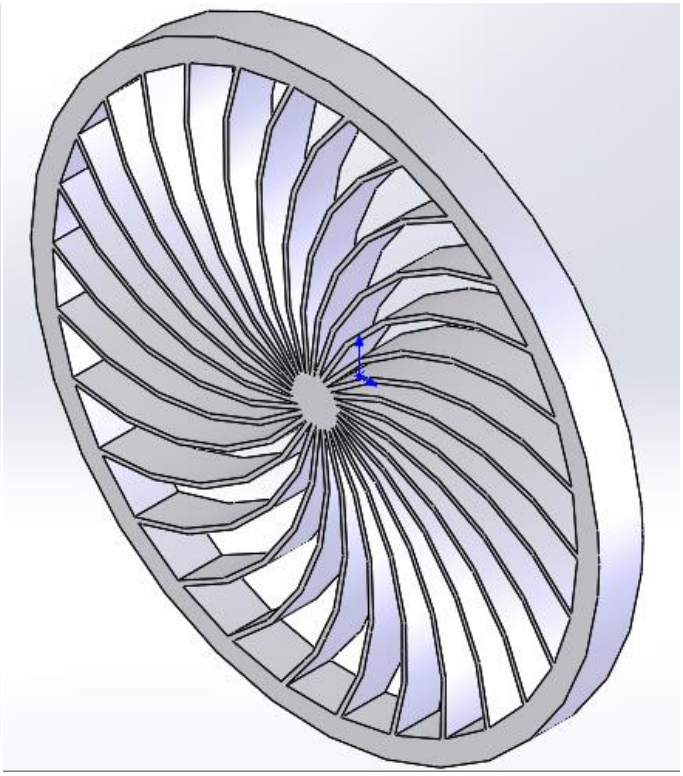
а



б



в



г

Рисунок 4.6 – Операції побудови моделі колеса

Виконуємо вирізання отвору із діаметром, що відповідає діаметру вихідного валу двигуна, на який кріпиться колесо (рисунок 4.7) та скруглення після чого отримуємо майже готову деталь колеса (рисунок 4.8).

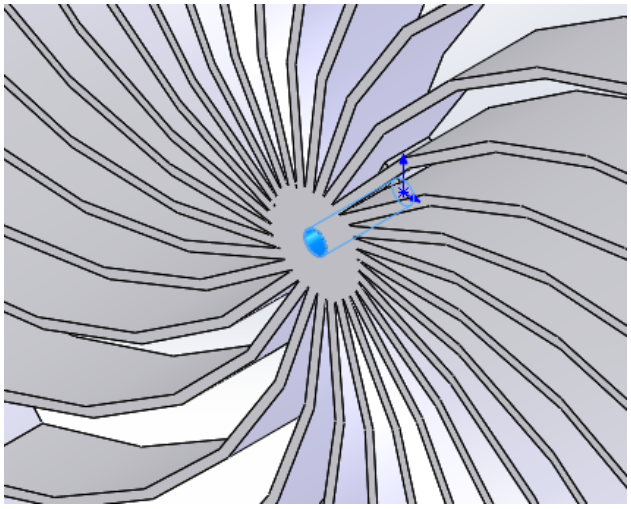


Рисунок 4.7 – отвір для кріплення колеса

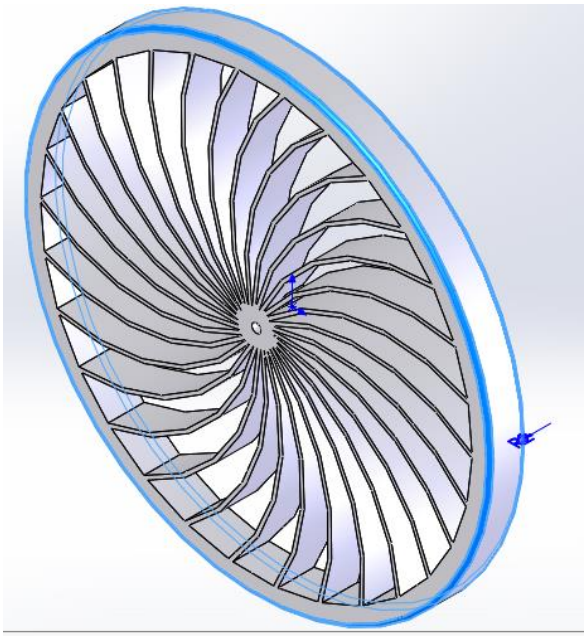


Рисунок 4.8 – Скруглення колеса

Для придання колесу рисунку протектора використовуємо функцію вирізання в результаті якої отримуємо паз (рисунок 4.9) після чого застосовуємо «масив по колу» та отримуємо готову модель колеса (рисунок 4.10).

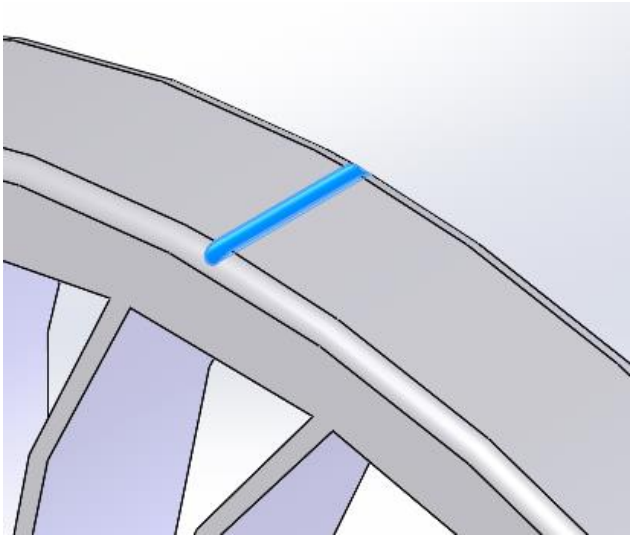


Рисунок 4.9 – Паз протектора колеса

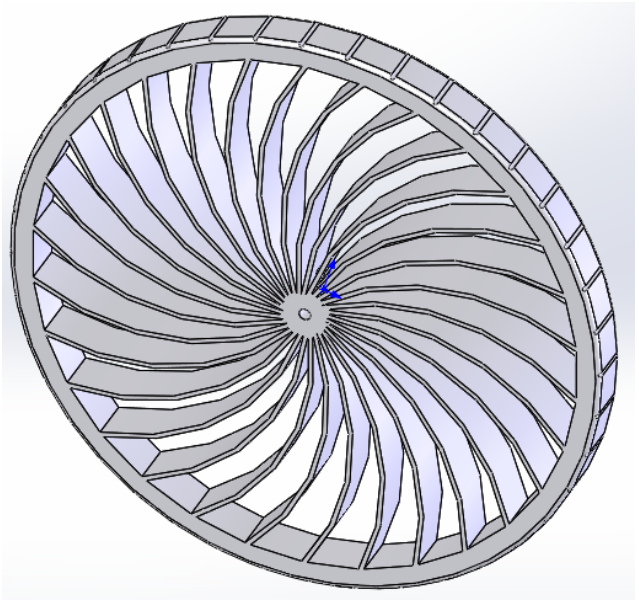


Рисунок 4.10 –Готова модель колеса

Моделі таких стандартизованих елементів як: драйвери двигунів L298N (рисунок 4.11), Arduino UNO R3 (рисунок 4.12), батарейний блок 4x18650 (рисунок 4.13), Bluetooth модуль HC-05 та ультразвуковий давач відстані HC-SR04 (рисунок

4.14). Знаходимо у вільному доступі [15] та завантажуємо. Модель маніпулятора взято з джерела [16].

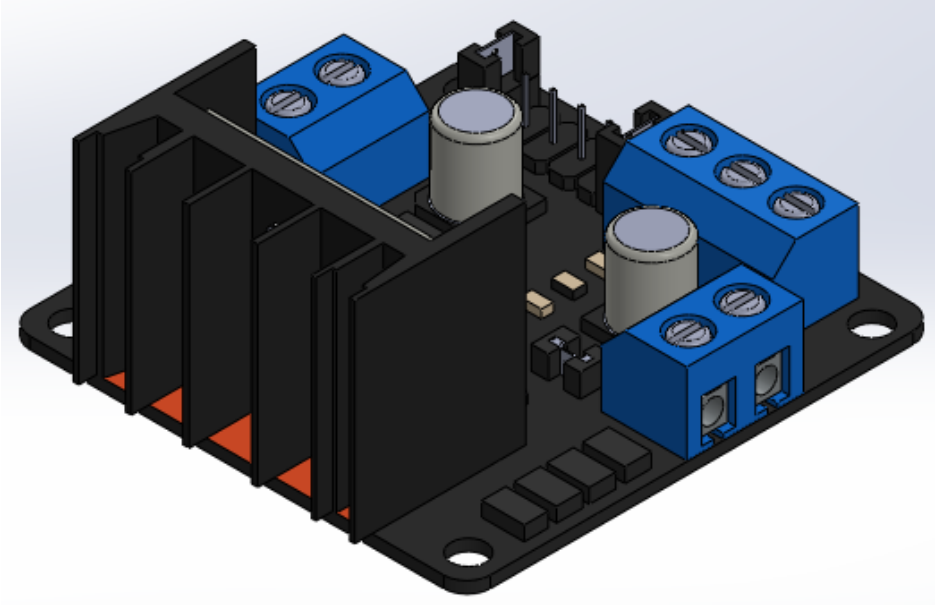


Рисунок 4.11 – Модель драйвера двигунів L298N

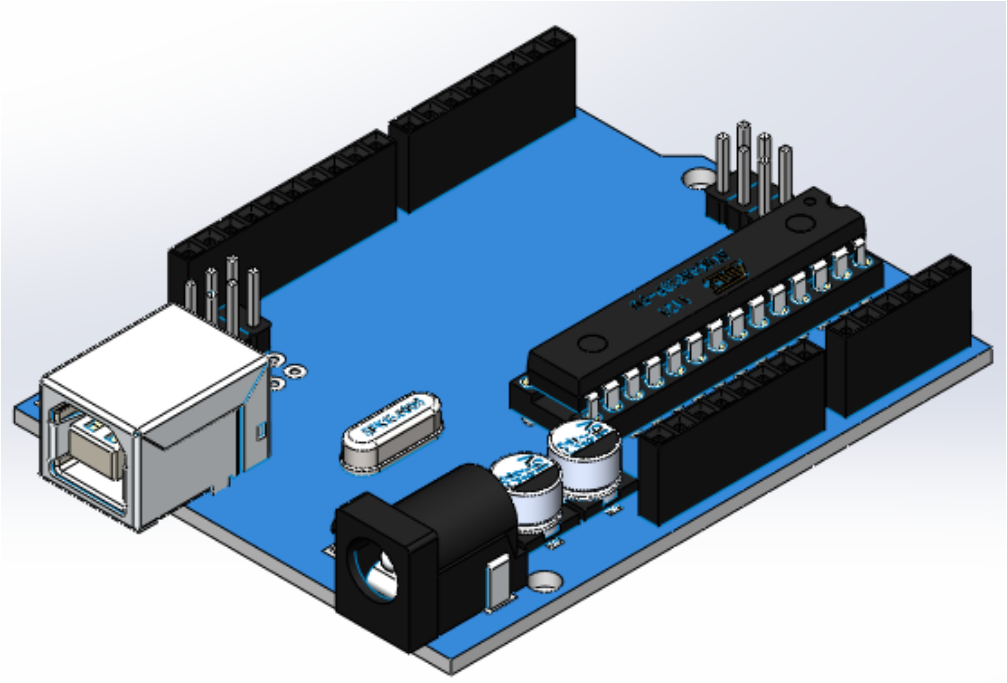


Рисунок 4.12 – Модель Arduino UNO R3

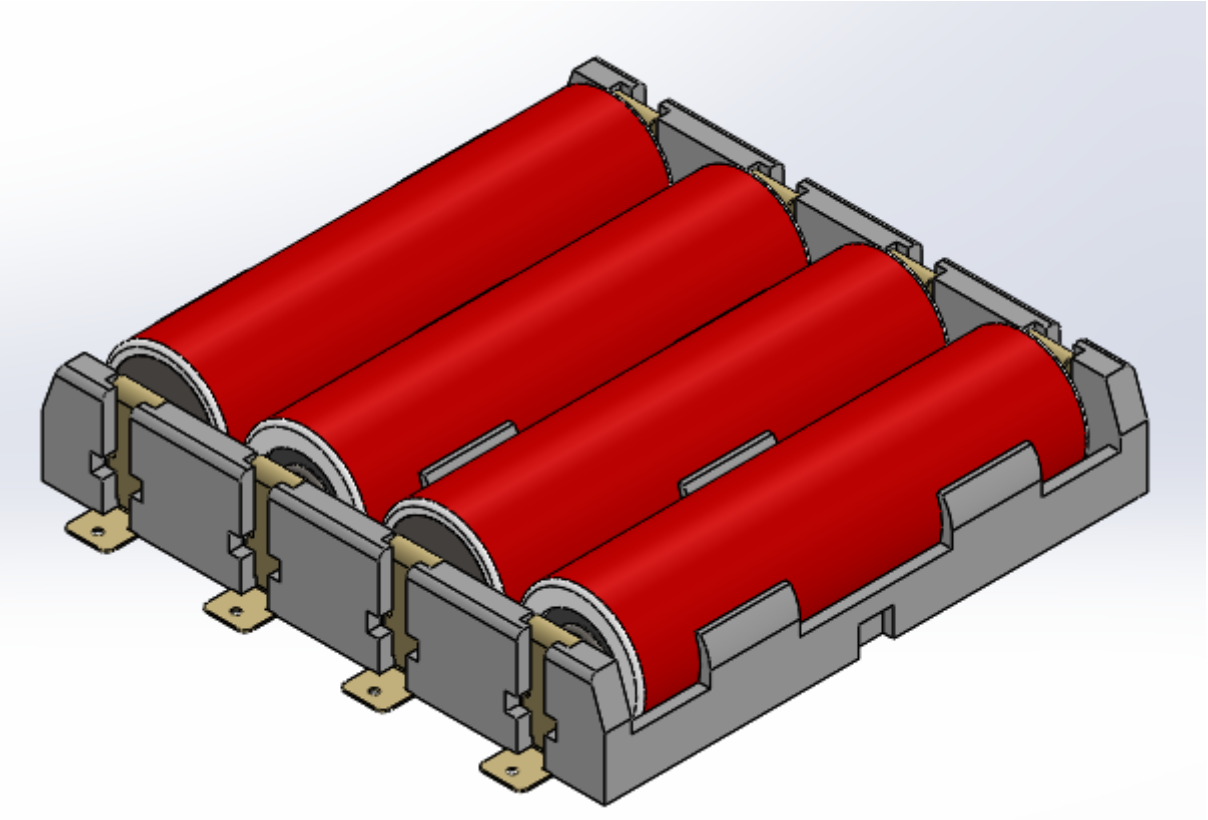


Рисунок 4.13 – Модель батарейного блоку 4x18650

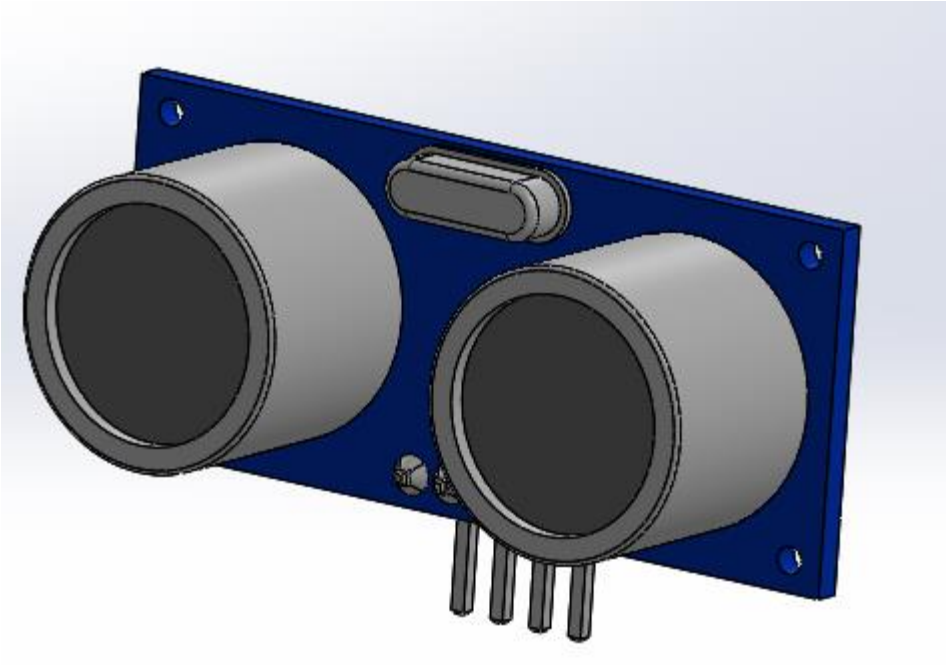


Рисунок 4.14 - Модель ультразвукового датчика відстані HC-SR04

Після моделювання усіх необхідних деталей, можна приступати до їх збірки у програмному середовищі SolidWorks. Для цього відкриваємо програму та вибираємо створити нову збірку. Після чого потрібно додати першу модель яка

буде фіксованою в нашому випадку це буде платформа робота, адже усі наступні деталі будуть кріпитися уже на неї. Спершу прикріплюємо двигуни за допомогою кріплень, а на двигуни монтуємо уже самі колеса. Усі деталі в збірці кріпимо за допомогою взаємозв'язків, що обмежують та фіксують рух деталі чи вузла в потрібних нам напрямках. Наступним етапом є прикріплення маніпулятора на центр основи. На захватному пристрої маніпулятора монтуємо ультразвуковий давач відстані. Після чого залишається додати плату Arduino Uno R3, живлення для неї а саме батарейний блок 4x18650 та драйвер двигунів L298N. В результаті отримуємо готову 3D-збірку моделі робота (рисунок 4.15).

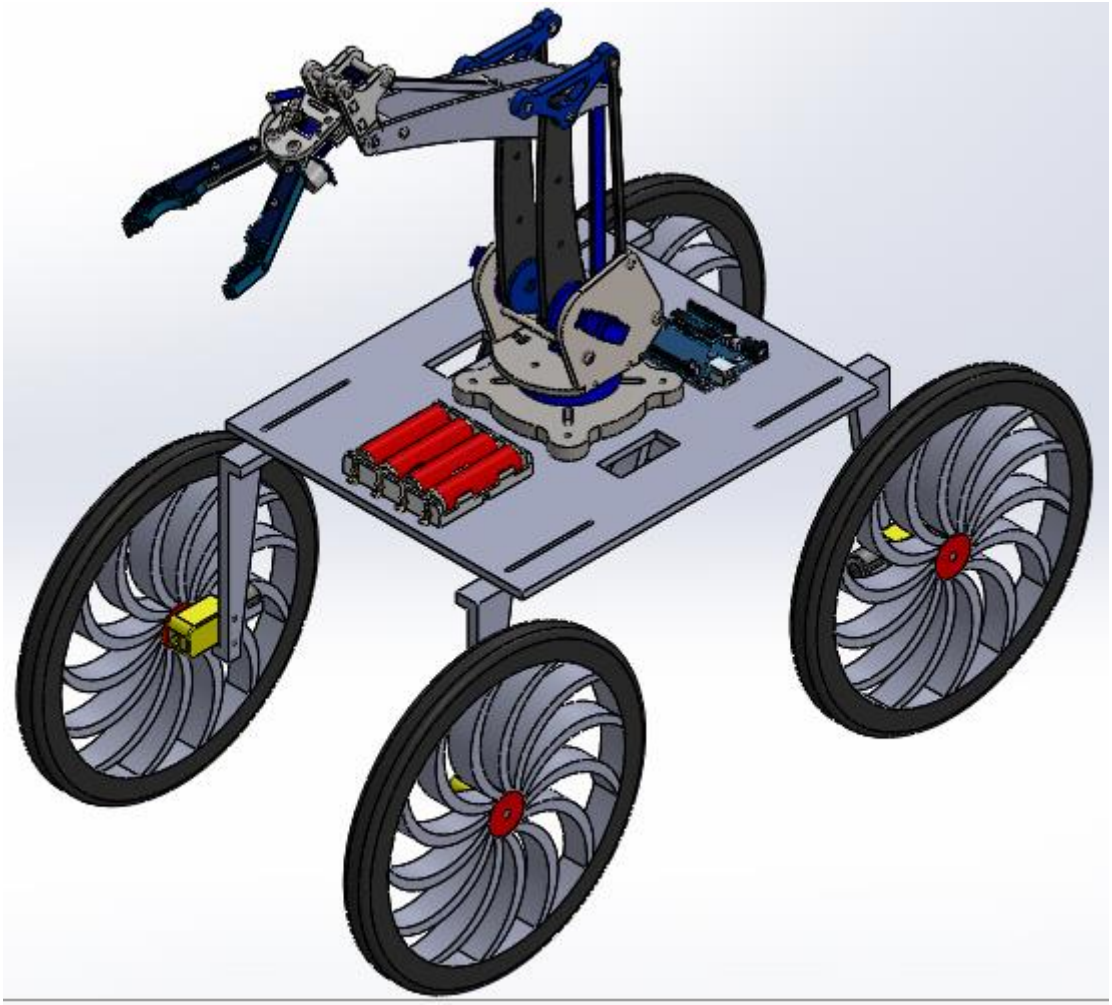


Рисунок 4.15 – Готова 3D-модель робота

Результати симуляції напружено-деформованого стану колеса в SOLIDWORKS Simulation, якщо на нього діє навантаження 100 Н (рис. 4.16)

показують, що максимальні напруження складають 15 МПа. Це безпечно для пластику з границею міцності 30-40 МПа.

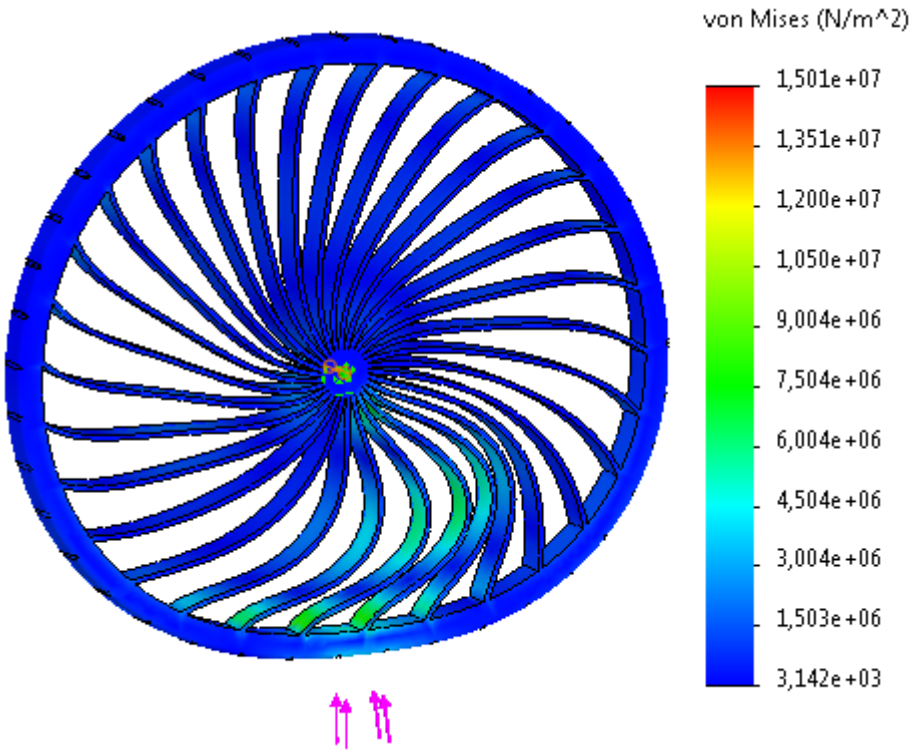


Рисунок 4.16 – Еквівалентні напруження в колесі (Па)

4.2 Принципова електрична схема і керуюча програма робота

Принципова електрична схема і відповідна модель розроблені за допомогою Proteus 8.13. На схемі (рис. 4.13) показано Arduino Uno на основі Atmel 328, ультразвуковий сенсор SRF04, сервоприводи sg90, UART-термінал, драйвер двигунів L298 [17] та електродвигуни постійного струму 12 В. Передні і задні електродвигуни з'єднані паралельно, що збільшує прохідність робота і його надійність. Сервоприводи призначені для руху маніпулятора (піни 8-12). Зокрема пін 8 дозволяє повертати маніпулятор навколо вертикальної осі, разом з сенсором SRF04. Сам сенсор під'єднаний до пінів 6, 7. Цифрові піни 2-5 керують напрямком обертання електродвигунів.

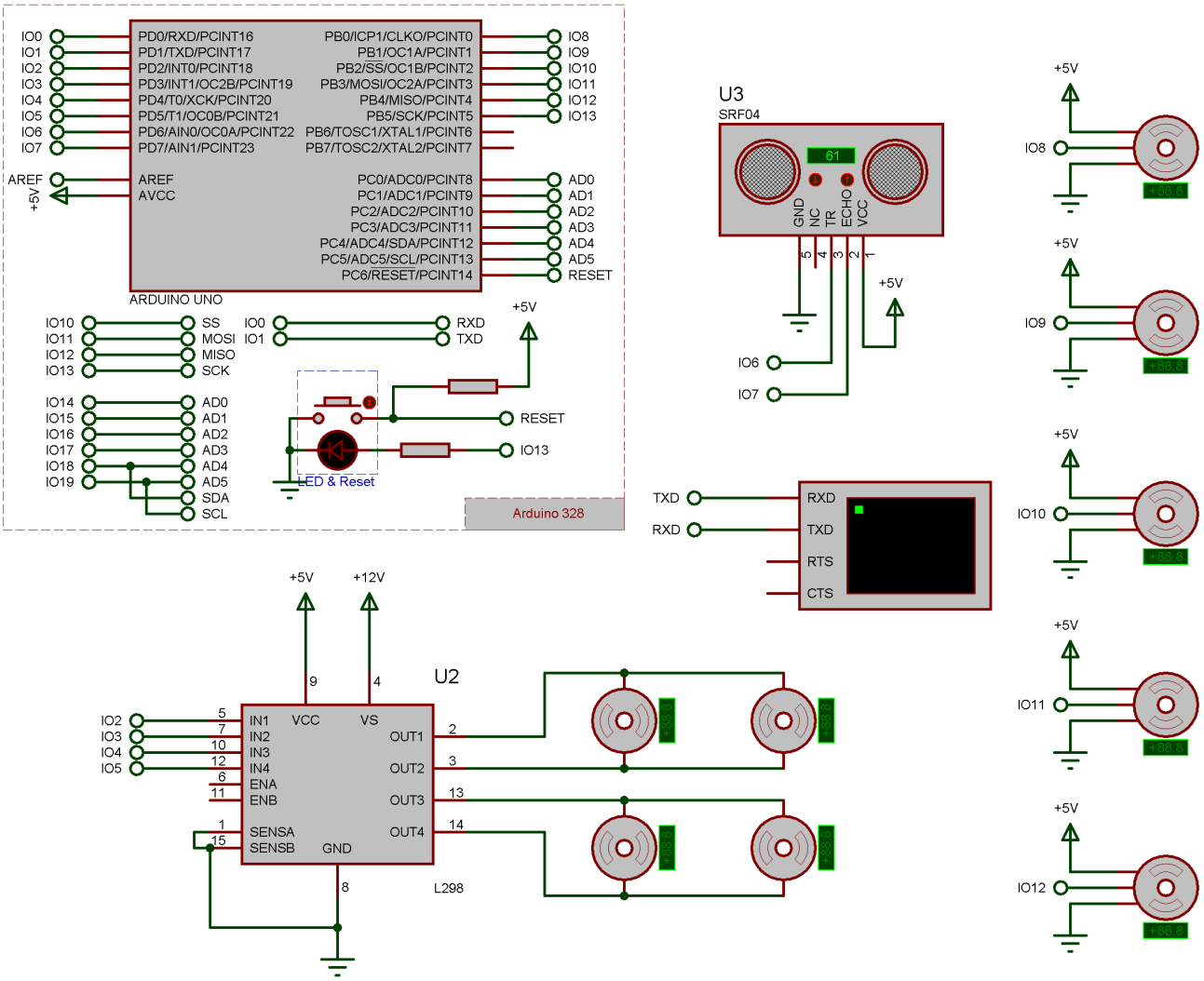


Рисунок 4.13 - Принципова електрична схема

Proteus 8.13 дозволяє виконати симуляцію цієї схеми разом зі скетчем Arduino. Скетч є керуючою програмою для робота. Спочатку в програмі створюються об’єкти сервоприводів:

```
#include <Servo.h>
Servo servo1;
Servo servo2;
Servo servo3;
Servo servo4;
Servo servo5;
```

Далі визначаються макроси:

```
#define motor1pin1 2
#define motor1pin2 3
#define motor2pin1 4
#define motor2pin2 5
#define ECHOPIN 7
#define TRIGPIN 6
```

Функція setup викликається один раз після включення. В ній установлюється режим роботи пінів, UART та серводвигунів:

```
void setup() {
  pinMode(motor1pin1, OUTPUT);
  pinMode(motor1pin2, OUTPUT);
  pinMode(motor2pin1, OUTPUT);
  pinMode(motor2pin2, OUTPUT);
  pinMode(ECHOPIN, INPUT);
  pinMode(TRIGPIN, OUTPUT);
  Serial.begin(9600);
  servo1.attach(8);
  servo2.attach(9);
  servo3.attach(10);
  servo4.attach(11);
  servo5.attach(12);
}
```

Функція ping призначена для вимірювання відстані ультразвуковим сенсором. Установлюється на TRIGPIN 0 на 2 мкс, установлюється 1 на 10 мкс, знову установлюється 0. Так ми посилаємо ультразвуковий сигнал до перешкоди. За допомогою pulseIn визначається час, за який сигнал вертається. Час конвертується у відстань з врахуванням швидкості звуку.

```
int ping()
  {digitalWrite(TRIGPIN, 0);
  delayMicroseconds(2);
  digitalWrite(TRIGPIN, 1);
  delayMicroseconds(10);
  digitalWrite(TRIGPIN, 0);
  int dist = pulseIn(ECHOPIN, 1);
  dist= dist/58;
  delay(50);
  return dist;}
```

Функція scan сканує простір шляхом повороту ультразвукового сенсора на певні кути. У циклі сервопривід 1 поступово повертається на 13 градусів і відбувається сканування. Функція повертає кут, в напрямку якого є перешкоди або повертає -1:

```
int scan(){
    int angle=0;
    int y;
    while (angle<=130)
       {servo1.write(angle);
        delay(100);
          y=ping();
          if (0<y && y<60) return angle;
        angle+=13;}
    return -1;
}
```

Наступна функція stop повністю зупиняє електродвигуни шляхом виведення 0 на піни 2-5. Відбувається затримка на час t:

```
void stop(int t){
  digitalWrite(motor1pin1, 0);
  digitalWrite(motor1pin2, 0);
  digitalWrite(motor2pin1, 0);
  digitalWrite(motor2pin2, 0);
  delay(t);
}
```

Функція moveF заставляє робота їхати прямо протягом часу t:

```
void moveF(int t){
  digitalWrite(motor1pin1, 1);
  digitalWrite(motor1pin2, 0);
  digitalWrite(motor2pin1, 1);
  digitalWrite(motor2pin2, 0);
  delay(t);
}
```

Функція moveR заставляє робота повертати праворуч протягом часу t:

```
void moveR(int t){
  digitalWrite(motor1pin1, 1);
  digitalWrite(motor1pin2, 0);
  digitalWrite(motor2pin1, 0);
  digitalWrite(motor2pin2, 1);
  delay(t);
}
```

Функція moveL заставляє робота повертати ліворуч протягом часу t:

```
void moveL(int t){
  digitalWrite(motor1pin1, 0);
  digitalWrite(motor1pin2, 1);
  digitalWrite(motor2pin1, 1);
  digitalWrite(motor2pin2, 0);
  delay(t);
}
```

Функція moveA заставляє робота повертати на кут angle:

```
void moveA(int angle){
Serial.println(angle);
if (0<=angle && angle<=43)
  moveL(43-angle);
else if (43<angle && angle<=86)
  moveF(43);
else
  moveR(angle-86);
}
```

Головна функція, яка виконується циклічно – loop. В ній відбувається сканування. Якщо нічого не знайдено, то відбувається поворот на довільний кут, інакше робот іде в напрямку перешкоди. Крім того в алгоритмі передбачено керування маніпулятором через послідовний порт (UART). Наприклад, якщо через UART надходить число 180 то здійснюється поворот першого сервоприводу на 180 градусів, якщо 1180 то здійснюється поворот другого (бо перша цифра 1) сервоприводу на 180 градусів і т.д. Подібним чином можна організувати також

керування електродвигунами (піни 2-5). Також Arduino надсилає через UART визначений кут. За допомогою бездротових адаптерів UART робот може працювати під керуванням людини або зовнішньої програми.

```
void loop() {
  int angle;
  angle=scan();
  Serial.println(angle);
  if (angle==-1) // якщо не знайдено
    moveA(random(0, 130)); // їхати в випадковому напрямку
  else
    moveA(angle); // їхати в напрямку angle
  stop(100);

  while (Serial.available() > 0)
  {
  int x = Serial.parseInt();
  if (x<1000) servo1.write(x); // x=180 - поворот на 180 градусів
  else if (x<2000) servo2.write(x-1000); // x=1180 - поворот на 180 градусів
  else if (x<3000) servo3.write(x-2000);
  else if (x<4000) servo4.write(x-3000);
  else if (x<5000) servo5.write(x-4000);
  }
}
```

Спроектований робот може бути використаний для участі в змаганнях роботів, які проводяться на кафедрі КМВ ІФНТУНГ [18], або для інших практичних цілей, зокрема для роботи в важкодоступних чи небезпечних місцях.

ВИСНОВКИ

1.Виконано аналіз перспективних конструкцій мобільних роботів. Виявлено, що колісна платформа є найпростішою для реалізації. Але для підвищення можливостей робота, його прохідності і надійності слід ставити електродвигун на кожне колесо.

2.Виконано огляд методів оптимізації та виявлено, що еволюційні алгоритми є перспективними у випадку задач з великою кількістю параметрів і стохастичністю. Тому їх доцільно застосовувати для оптимізації конструкції мобільних роботів, які долають довільні перешкоди.

3.За допомогою мови Python та її пакетів Nodebox for OpenGL та Pymunk розроблено програму для імітаційного моделювання подолання роботом випадкової перешкоди та оптимізації його конструкції, а також проведено його оптимізацію з використанням функції differential_evolution з пакету SciPy.

4. За допомогою SOLIDWORKS спроектовано параметричну модель мобільного робота з маніпулятором. Значення параметрів робота змінено на знайдені оптимальні.

5. За допомогою Proteus 8.13 розроблено принципову електричну схему робота, яка містить Arduino Uno на основі Atmel 328, ультразвуковий сенсор SRF04, сервоприводи sg90, UART-термінал, драйвер двигунів L298. Такі компоненти дозволяють здешевити виробництво робота. Розроблено керуючу програму робота - скетч для Arduino. Робот може працювати в автономному режимі пошуку об’єктів або керуватись за командами з UART. Виконано перевірку її працездатності шляхом симуляції.

6. Спроектований робот може бути використаний для участі в змаганнях роботів, які проводяться на кафедрі КМВ ІФНТУНГ, або для інших практичних цілей, зокрема для роботи в важкодоступних чи небезпечних місцях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Промислова та мобільна робототехніка: URL: https://conf.ztu.edu.ua/wp-content/uploads/2020/05/6.-promyslova-ta-mobilna-robototehnika.pdf (access 20.06.23)

2. Електронний архів Харківського національного університету радіоелектроніки. URL: https://openarchive.nure.ua/home (access 20.06.23)

3. Ешлок Д. Еволюційні обчислення для моделювання та оптимізації. Нью-Йорк : Springer, 2006. doi:10.1007/0-387-31909-3 ISBN 0-387-22196-4.

4. Саймон Д. Алгоритми еволюційної оптимізації. Wiley & Sons, 2013. ISBN 978-0-470-93741-9.

5. Копей В. Б. Мова програмування Python для інженерів і науковців : навчальний посібник. Івано-Франківськ : ІФНТУНГ, 2019. 272 с.

6. Бек Т., Фогель Д., Міхалевич З. Еволюційне обчислення 1: Основні алгоритми та оператори. Бока-Ратон, США : CRC Press, 1999. ISBN 978-0-7503-0664-5 .

7. SciPy documentation. URL: https://docs.scipy.org/doc/scipy/ (access 20.06.23)

8. Storn R., Price K. Differential Evolution - a Simple and Efficient Heuristic for Global Optimization over Continuous Spaces // Journal of Global Optimization, 1997, 11, P.341 - 359.

9. Differential_evolution : URL: http://en.wikipedia.org/wiki/Differential_evolution (access 20.06.23)

10. Differential evolution. URL: http://www1.icsi.berkeley.edu/~storn/code.html (access 20.06.23)

11. Wormington M., Panaccione C., Matney K. M., Bowen D. K. Characterization of structures from X-ray scattering data using genetic algorithms // Phil. Trans. R. Soc. Lond. A, 1999, 357, P.2827-2848.

12. Das S., Suganthan P. N. Differential Evolution: A Survey of the State-of-the-art // IEEE Trans. on Evolutionary Computation, Vol. 15, No. 1, Feb. 2011. P. 4-31. DOI: 10.1109/TEVC.2010.2059031.

13.Копей Б.В., Копей В.Б. Використання методу скінченних елементів та тривимірного комп’ютерного моделювання для конструювання та оптимізації

параметрів нафтогазового обладнання: Навчальний посібник. Івано-Франківськ : Факел, 2008. 117 с.

14.Копей В.Б., Онисько О.Р., Борущак Л.О., Роп’як Л.Я. Автоматизоване проектування різальних інструментів: Навчальний посібник. Івано-Франківськ : ІФНТУНГ, 2012. 208 с.

15. Grabcad Library. URL: https://grabcad.com/library (access 20.06.23)

16. Farial Tasnim. Robotic Arm T310 (6 DOF). URL: https://grabcad.com/library/robotic-arm-t310-6-dof-1 (access 20.06.23)

17. How to use the L298n motor driver. URL: https://www.hackster.io/ryanchan/how-to-use-the-l298n-motor-driver-b124c5 (access 20.06.23)

18. Mechatronics3. URL: https://github.com/vkopey/mechatronics3 (access 20.06.23)

ДОДАТКИ

Додаток – Програма для симуляції і оптимізації мобільного робота

```
#encoding: utf-8
from __future__ import division

from nodebox.graphics import *
import pymunk
import pymunk.pyglet_util
import random, time

def create_moto(x):
    """Створює візок. Індекси списку параметрів x:
    0 - радіус заднього колеса
    1 - радіус переднього колеса
    2 - відстань між колесами (мінімальна)
    3 - виліт заднього колеса
    4 - виліт переднього колеса
    """
    y0=100 # висота осей над землею
    l=x[0]+x[1]+x[2] # відстань між осями
    b0 = pymunk.Body() # заднє колесо
    b0.position = 0-x[3],y0
    g0=pymunk.Circle(b0, x[0])
    g0.mass = 100
    g0.friction = 1

    b1 = pymunk.Body() # переднє колесо
    b1.position = l+x[4],y0
    g1=pymunk.Circle(b1, x[1])
    g1.mass = 100
```

```
    g1.friction = 1

    b2 = pymunk.Body() # корпус
    b2.position = l/2,y0+max([x[0],x[1]])+15
    g2 = pymunk.Poly.create_box(b2, size=(l,20))
    g2.mass = 100
    g2.friction = 0

    # з'єднання:
    J=(
    pymunk.PinJoint(b0, b2, (0, 0), (0, 0)),
    pymunk.PinJoint(b1, b2, (0, 0), (0, 0)),
    pymunk.PinJoint(b0, b2, (0, 0), (-l/2, 0)),
    pymunk.PinJoint(b1, b2, (0, 0), (l/2, 0)))

    space.add(b0, b1, b2, g0, g1, g2, *J) # додати в space
    return (b0, b1, b2, g0, g1, g2)+J

def create_poly(x,y,x1,y1):
    """Створює випакову цеглину з позицією x,y і швидкістю x1,y1"""
    body = pymunk.Body()
    body.position = x,y
    body.velocity= x1,y1
    poly = pymunk.Poly.create_box(body, size=(random.randint(10,40),10))
    poly.mass = 10
    poly.friction = 1
    poly.color = (255, 0, 0, 255) # червоний колір (R,G,B,A)
    space.add(body, poly)

def create_static(pos=(300, 100), p1=(-200, 0), p2=(300, 0)):
```

```
    """Створює нерухоме тіло - лінію за точками p1, p2"""
    body = pymunk.Body(body_type = pymunk.Body.STATIC)
    body.position = pos
    line = pymunk.Segment(body, p1, p2, 3)
    line.friction = 1
    line.color = (0, 255, 0, 255)
    space.add(body, line)


draw_options = pymunk.pyglet_util.DrawOptions()

def draw(canvas):
    """Функція пакету nodebox, що рисує кожен кадр анімації"""
    global tm, space, B # глобальні змінні
    background(1) # зарисувати попередній кадр
    print tm
    if tm>10 or B[1].position[0]>600 or canvas.keys.char=="a":
        canvas.stop()

    space.step(0.05) # симуляція фізики
    tm+=0.05 # збільшити
    B[0].angular_velocity= -5 # швидкість колеса
    B[1].angular_velocity= -5
    space.debug_draw(draw_options) # рисувати усі об'єкти


canvas.size = 700, 500 # розміри вікна
def f(x):
    """Функція, що мінімізується. Повертає час tm, за який візок з параметрами x
подолав перешкоду"""
```

```
    global tm, space, draw, canvas, B # глобальні змінні
    canvas=Canvas()
    tm=0
    space = pymunk.Space() # створити простір
    space.gravity = 0,-981 # гравітацію
    create_static(pos=(0, 10), p1=(-200, 10), p2=(700, 10)) # дорогу
    for i in range(50):
        create_poly(random.randint(300,400), 200, 0, 0) # цеглини

    B = create_moto(x) # візок з параметрами x
    canvas.run(draw) # розпочати анамацію

    for obj in B:
        obj.position=1000,1000
        space.remove(obj) # видалити усі об'єкти
    return tm

def opti_grid(X=range(0,20)):
    """Функція для оптимізації однієї змінної сітковим методом. Використовується
для дослідження впливу обраного параметра на tm"""
    Y=[] # значення часу tm
    for x in X:
        y=[f([30,30,20,14.4,x]) for i in range(3)] # симуляція 3 рази
        Y.append(sum(y)/len(y)) # додати середній час
    plt.plot(X,Y,'o-') # нарисувати графік
    plt.xlabel(u"x, мм"); plt.ylabel(u"t, c")
    plt.show()



import numpy as np
```

```
import matplotlib.pyplot as plt
from scipy.optimize import minimize, differential_evolution
#f([30,30,20,14.4,18.9])
#opti_grid()
#print minimize(f, x0=[25., 25, 17, 10, 10], method="L-BFGS-B", bounds=[(20, 30),(20, 30),(5, 30),(0, 20),(0, 20)])
print differential_evolution(f, bounds=[(20, 30),(20, 30),(5, 30),(0, 20),(0, 20)], maxiter=10) # знайти мінімум
```

Додаток – Керуюча програма робота

```
#include <Servo.h>
Servo servo1;
Servo servo2;
Servo servo3;
Servo servo4;
Servo servo5;

#define motor1pin1 2
#define motor1pin2 3
#define motor2pin1 4
#define motor2pin2 5
#define ECHOPIN 7
#define TRIGPIN 6

void setup() {
  pinMode(motor1pin1, OUTPUT);
  pinMode(motor1pin2, OUTPUT);
  pinMode(motor2pin1, OUTPUT);
  pinMode(motor2pin2, OUTPUT);
  pinMode(ECHOPIN, INPUT);
```

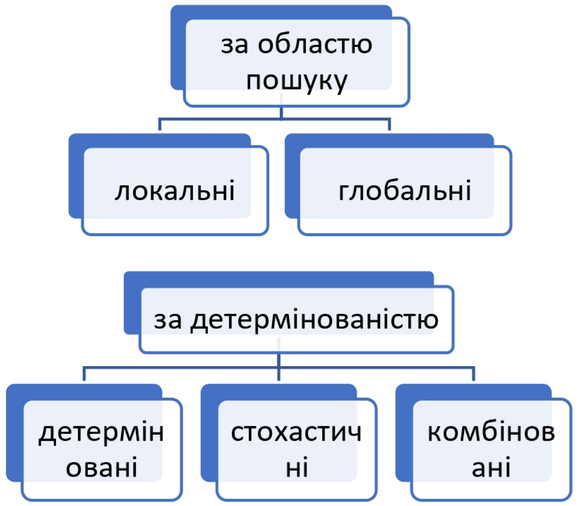
```
  pinMode(TRIGPIN, OUTPUT);
  Serial.begin(9600);
  servo1.attach(8);
  servo2.attach(9);
  servo3.attach(10);
  servo4.attach(11);
  servo5.attach(12);
}

int ping()
  {digitalWrite(TRIGPIN, 0);
  delayMicroseconds(2);
  digitalWrite(TRIGPIN, 1);
  delayMicroseconds(10);
  digitalWrite(TRIGPIN, 0);
  int dist = pulseIn(ECHOPIN, 1);
  dist= dist/58;
  delay(50);
  return dist;}

int scan(){
    int angle=0;
    int y;
    while (angle<=130)
       {servo1.write(angle);
        delay(100);
          y=ping();
        //Serial.println(y);
          if (0<y && y<60) return angle;
        angle+=13;}
```

```
    return -1;
}

void stop(int t){
  digitalWrite(motor1pin1, 0);
  digitalWrite(motor1pin2, 0);
  digitalWrite(motor2pin1, 0);
  digitalWrite(motor2pin2, 0);
  delay(t);
}

void moveF(int t){
  digitalWrite(motor1pin1, 1);
  digitalWrite(motor1pin2, 0);
  digitalWrite(motor2pin1, 1);
  digitalWrite(motor2pin2, 0);
  delay(t);
}

void moveR(int t){
  digitalWrite(motor1pin1, 1);
  digitalWrite(motor1pin2, 0);
  digitalWrite(motor2pin1, 0);
  digitalWrite(motor2pin2, 1);
  delay(t);
}

void moveL(int t){
  digitalWrite(motor1pin1, 0);
  digitalWrite(motor1pin2, 1);
```

```
  digitalWrite(motor2pin1, 1);
  digitalWrite(motor2pin2, 0);
  delay(t);
}

void moveA(int angle){
Serial.println(angle);
if (0<=angle && angle<=43)
  moveL(43-angle);
else if (43<angle && angle<=86)
  moveF(43);
else
  moveR(angle-86);
}


void loop() {
  int angle;
  angle=scan();
  Serial.println(angle);
  if (angle==-1) // якщо не знайдено
    moveA(random(0, 130)); // їхати в випадковому напрямку
  else
    moveA(angle); // їхати в напрямку angle
  stop(100);

 while (Serial.available() > 0)
  {
  int x = Serial.parseInt();
  if (x<1000) servo1.write(x); // x=180 - поворот на 180 градусів
```

```
  else if (x<2000) servo2.write(x-1000); // x=1180 - поворот на 180 градусів
  else if (x<3000) servo3.write(x-2000);
  else if (x<4000) servo4.write(x-3000);
  else if (x<5000) servo5.write(x-4000);
   }
}
```

Методи оптимізації



Цільова функція

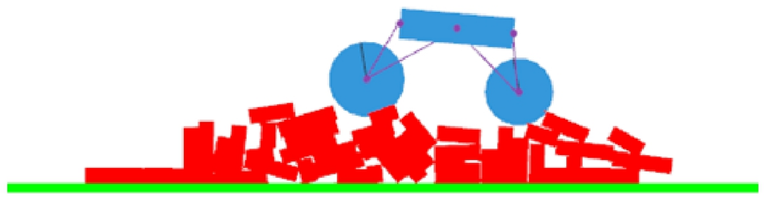
```
def f(x):
    global tm, space, draw, canvas, B
    canvas=Canvas()
    tm=0
    space = pymunk.Space()
    space.gravity = 0,-981
    create_static(pos=(0, 10), p1=(-200, 10),
                  p2=(700, 10))
    for i in range(50):
        create_poly(random.randint(300,400),
                    200, 0, 0)
    B = create_moto(x)
    canvas.run(draw)
    for obj in B:
        obj.position=1000,1000
        space.remove(obj)
    return tm
```

Функція differential_evolution з пакету SciPy

differential_evolution(func, bounds, args=(), strategy=best1bin, maxiter=1000, popsize=15, tol=0.01, mutation=(0.5, 1), recombination=0.7, seed=None, callback=None, disp=False, polish=True, init=latinhypercube, atol=0, updating=immediate, workers=1)

Ітерація 1

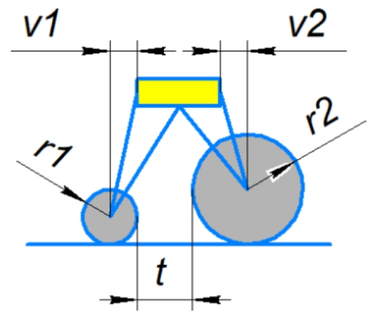
Ітерація 2



Ітерація 3

Ітерація 10

Параметри для оптимізації

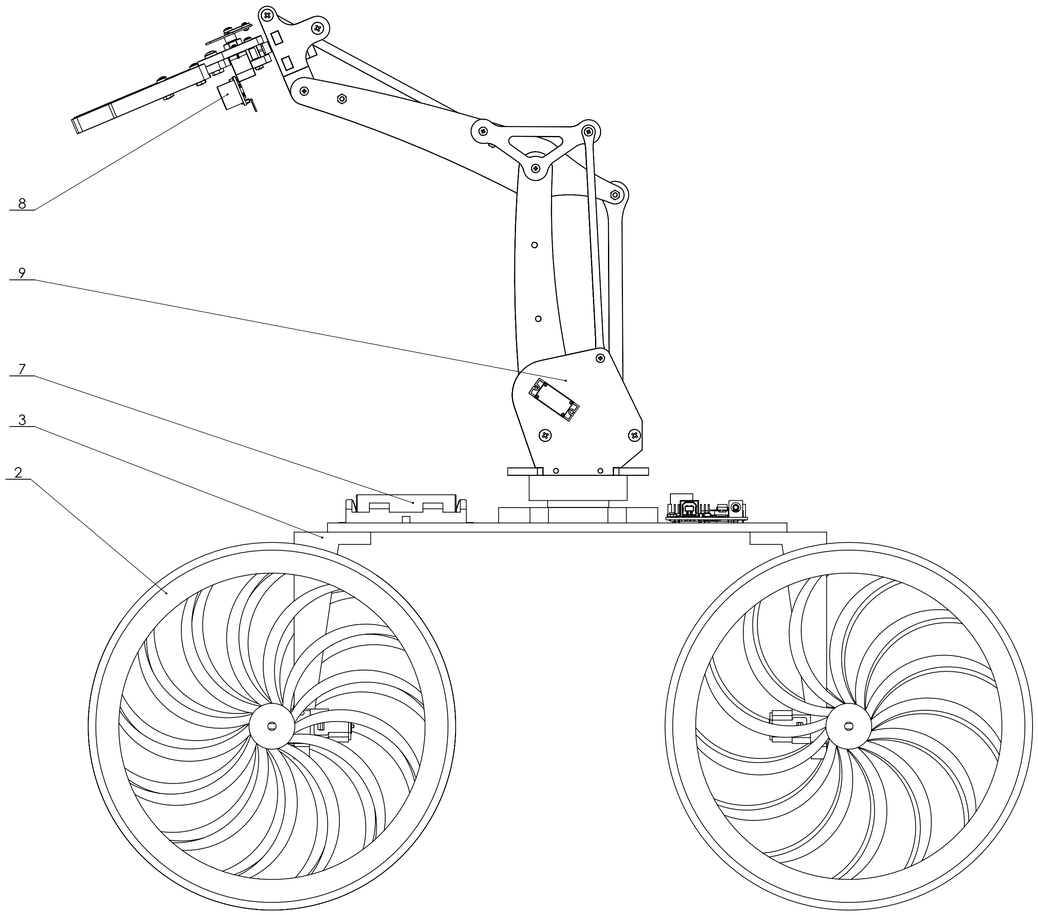


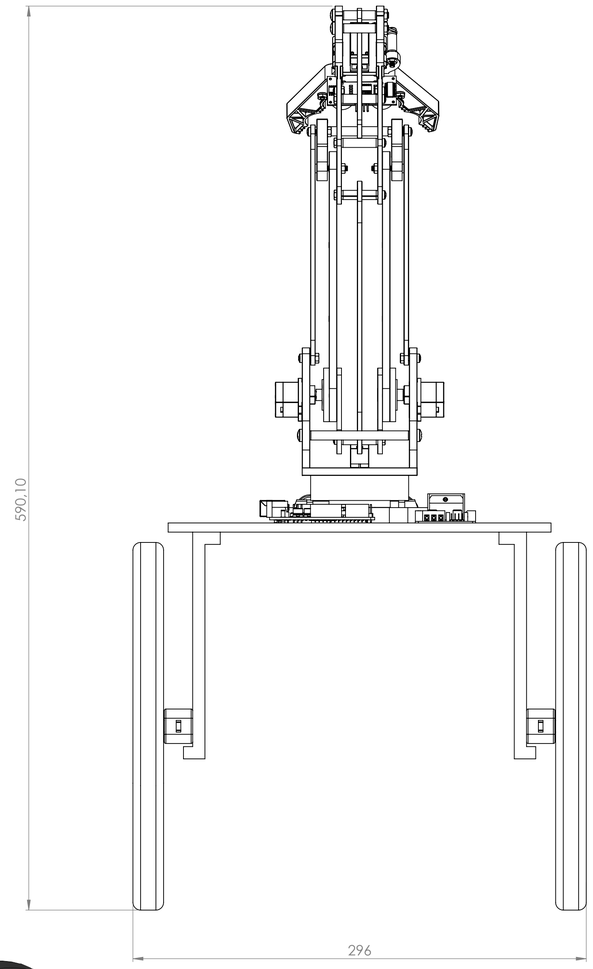
Модель робота для Pymunk

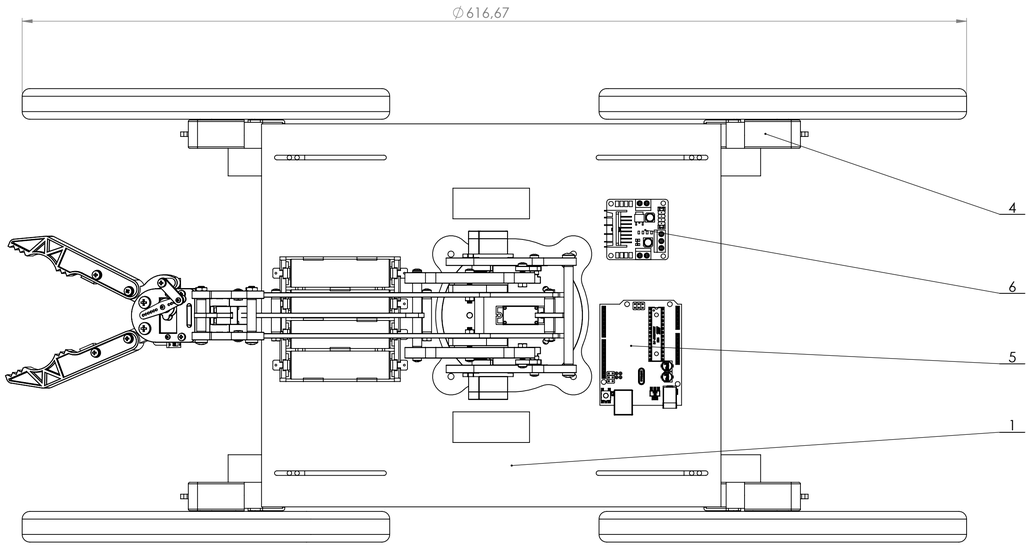
```
def create_moto(x):
    y0=100
    l=x[0]+x[1]+x[2]
    b0 = pymunk.Body()
    b0.position = 0-x[3],y0
    g0=pymunk.Circle(b0, x[0])
    g0.mass = 100
    g0.friction = 1
    b1 = pymunk.Body()
    b1.position = l+x[4],y0
    g1=pymunk.Circle(b1, x[1])
    g1.mass = 100
    g1.friction = 1
    b2 = pymunk.Body()
    b2.position =
l/2,y0+max([x[0],x[1]])+15
    g2 = pymunk.Poly.create_box(b2,
size=(l,20))
    g2.mass = 100
    g2.friction = 0
    J=(
    pymunk.PinJoint(b0, b2, (0, 0), (0,
0)),
    pymunk.PinJoint(b1, b2, (0, 0), (0,
0)),
    pymunk.PinJoint(b0, b2, (0, 0), (-l/2,
0)),
    pymunk.PinJoint(b1, b2, (0, 0), (l/2,
0)))
    space.add(b0, b1, b2, g0, g1, g2, *J)
    return (b0, b1, b2, g0, g1, g2)+J
```

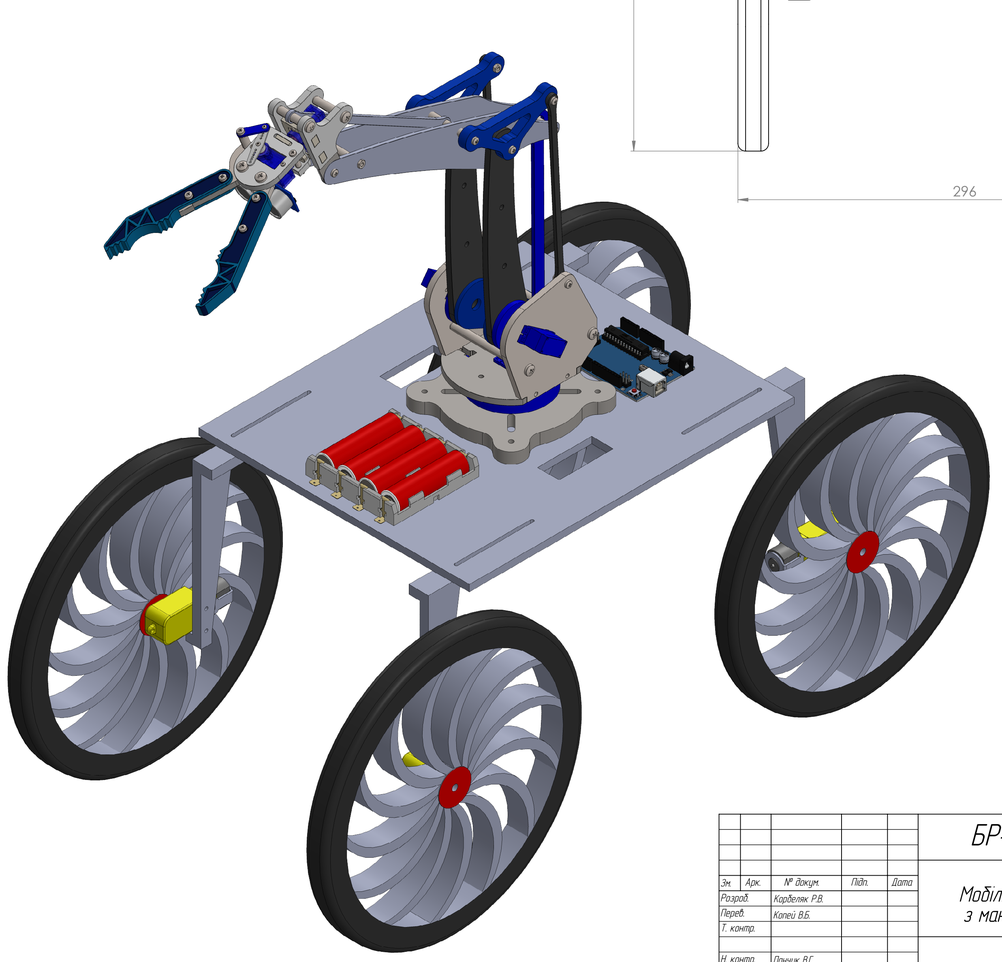
Зм.	Арк.	№ докум.	Підп.	Дата	БР.ПМІ-04.00.00.000			
Розроб.		Корбелюк Р.В.			Алгоритми оптимізації	Літ.	Маса	Масштаб
Перев.		Копей В.Б.						
Т. контр.						Арк.	Аркушів	
Н. контр.					ІФНТУНГ ПМІ-21-1К			
Затв.								

Формат А1









						БР–ПМІ–04.00.001 СК	
Зм.	Арк.	№ докум.	Підп.	Дата			
Розроб.		Корбелюк Р.В.			Мобільний робот з маніпулятором	Літ.	Маса	Масштаб
Перев.		Копеї В.Б.						1:2
Т. контр.						Арк.	Аркушів	
Н. контр.		Панчук В.Г.				ІФНТУНГ ПМІ–21–1К		
Затв.								

Формат А1

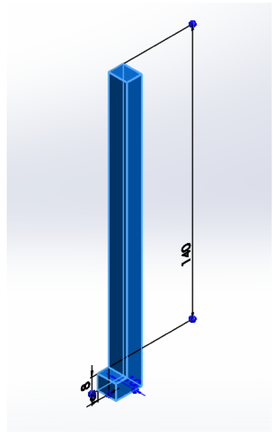


Рисунок 1.1 – Початковий ескіз кріплення

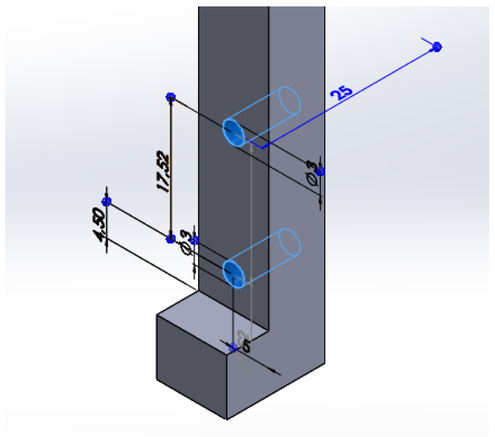


Рисунок 1.2– Отвори для кріплення двигуна

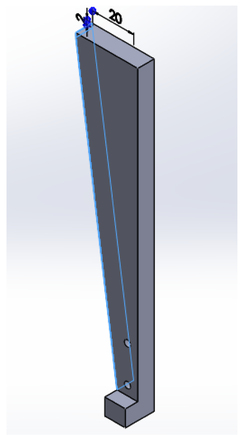


Рисунок 1.3 – Придання жорсткості кріпленню

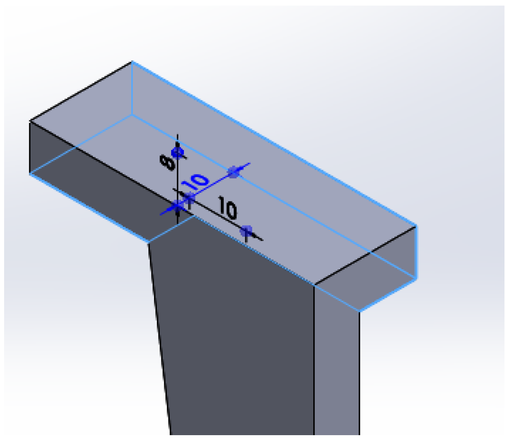


Рисунок 1.4– Створення площадки

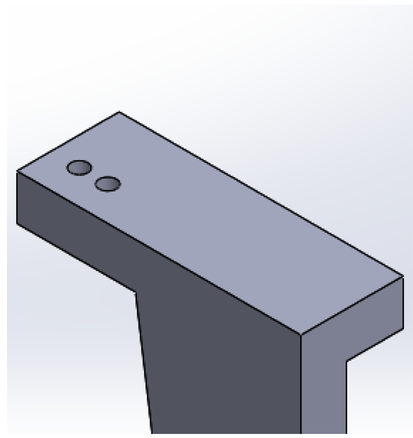


Рисунок 1.5 – Вирізання отворів для кріплення до основи

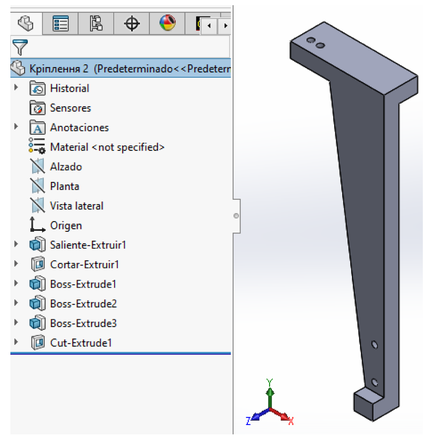


Рисунок 1.6 – Древо моделі "Кріплення"

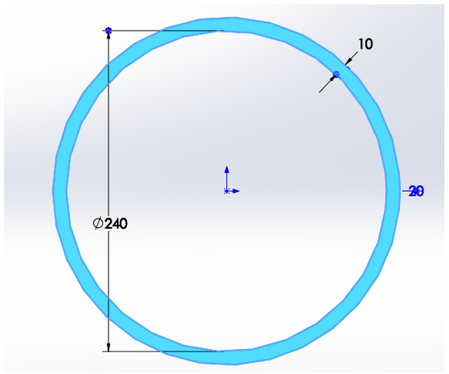


Рисунок 2.1 – Початковий ескіз "Колеса"

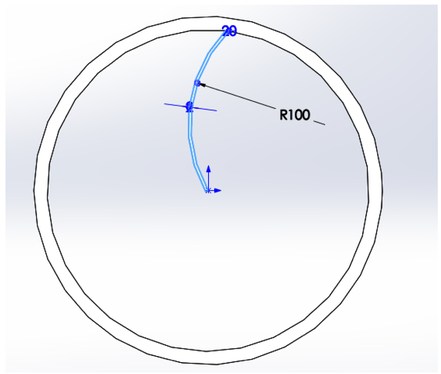


Рисунок 2.2 – Створення ескізу лопки

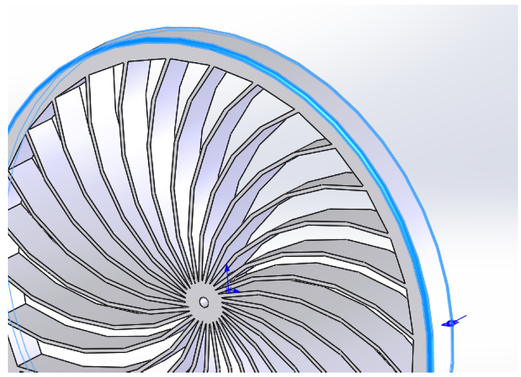


Рисунок 2.3 – Скруглення колеса

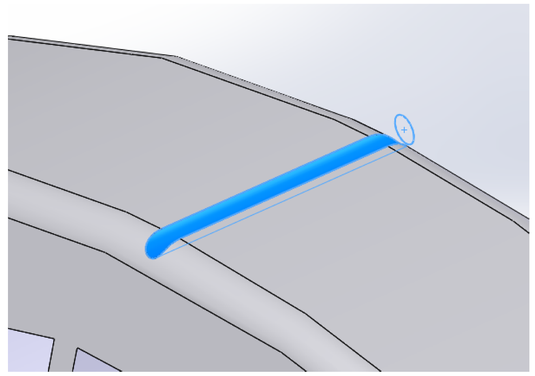


Рисунок 2.4 – Створення канавки для протектора

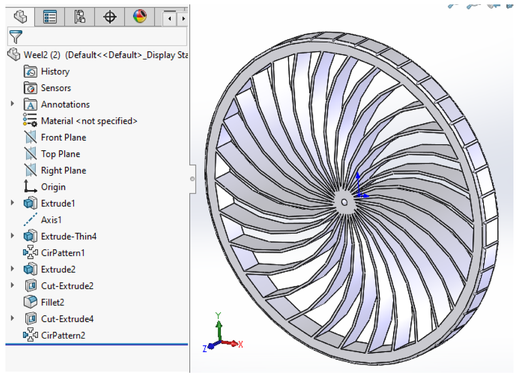


Рисунок 2.5 – Готова модель колеса з деревом побудови

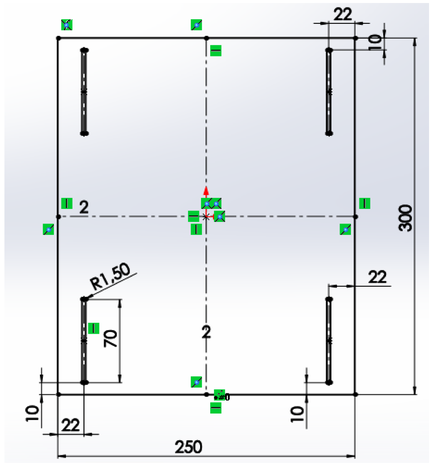


Рисунок 3.1 – Початковий ескіз "Основи"

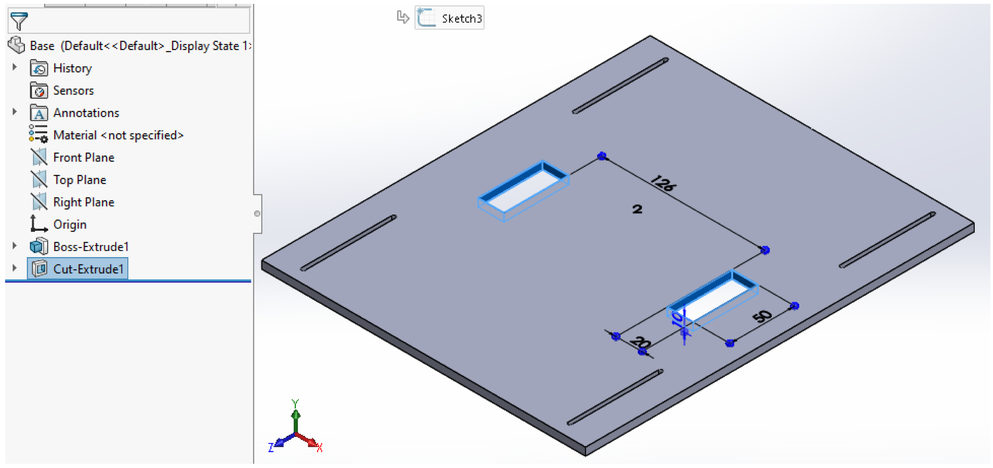


Рисунок 3.2 – Вирізання отворів для прокладання контактів, які утвердять двигуни

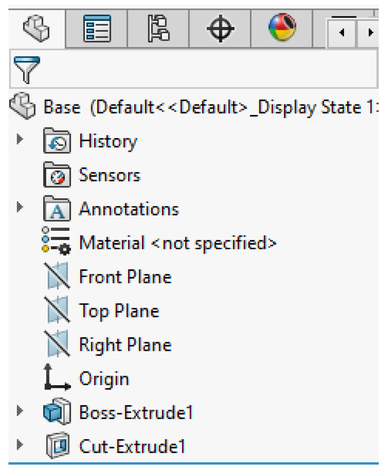


Рисунок 3.1 – Дерево побудови моделі "Основа"

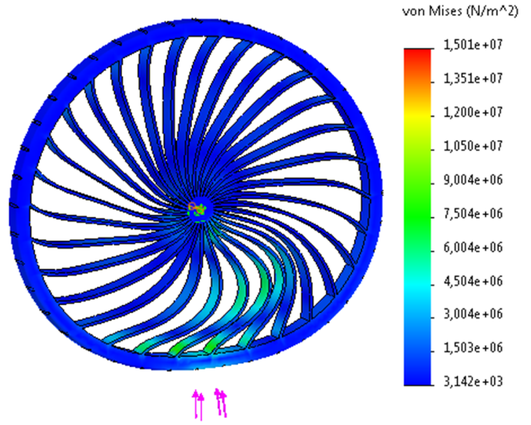
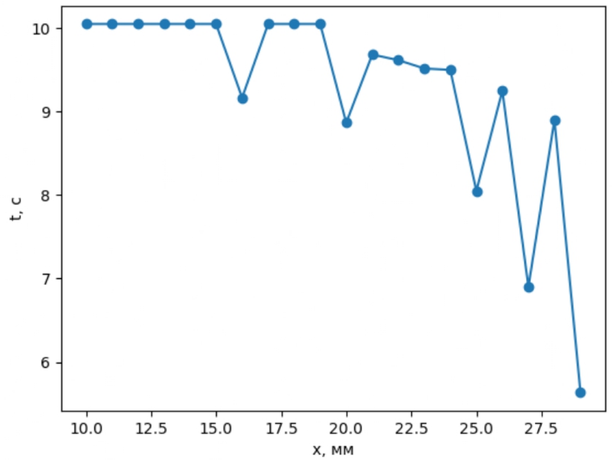


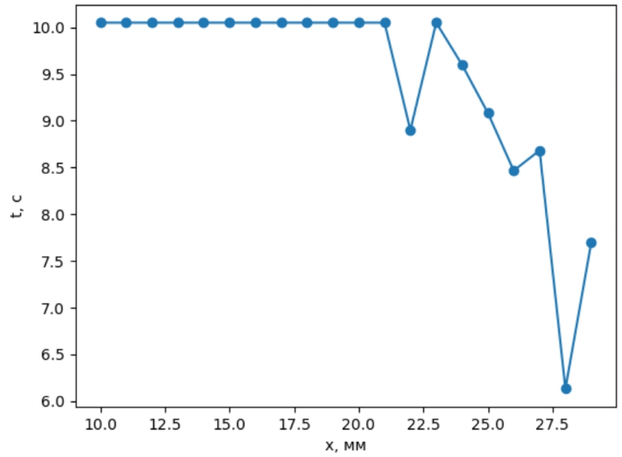
Рисунок 4.1 – Симуляція FEA

						БР.ПМІ-04.00.00.000		
Зм.	Арк.	№ докум.	Підп.	Дата				
Розроб.		Корделяк Р.В.			Проектування деталей робота в SOLIDWORKS	Літ.	Маса	Масштаб
Перев.		Колей В.Б.						
Т. контр.						Арк.	Аркушів	
Н. контр.						ІФНТУНГ ПМІ-21-1К		
Затв.								

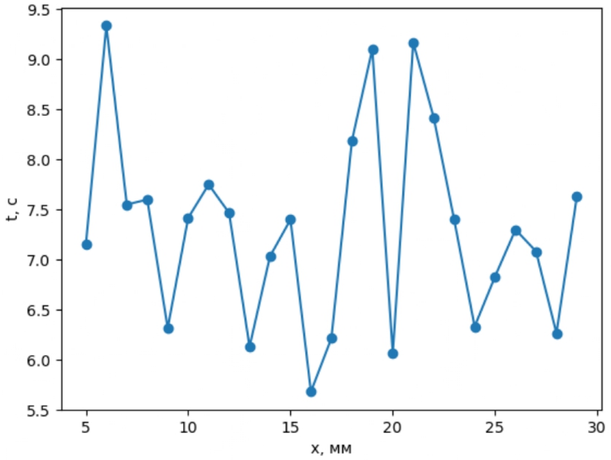
Формат А1



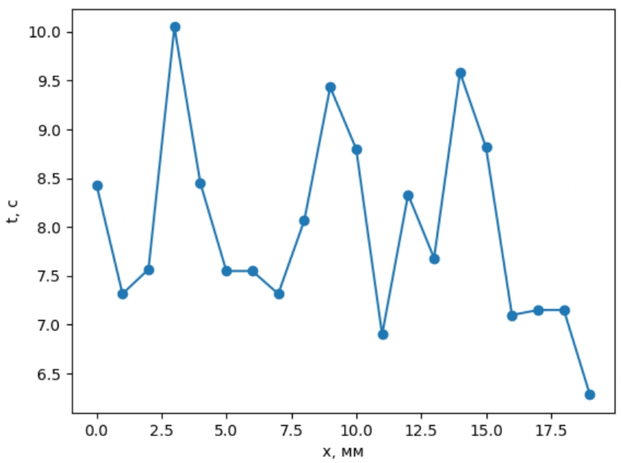
Залежність часу подолання перешкоди t від радіуса заднього колеса



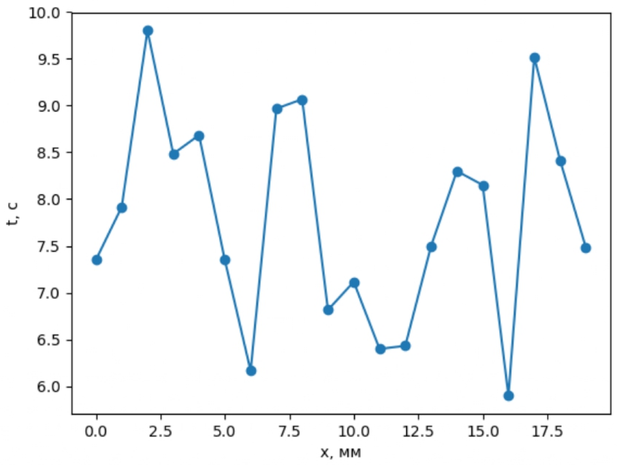
Залежність часу подолання перешкоди t від радіуса переднього колеса



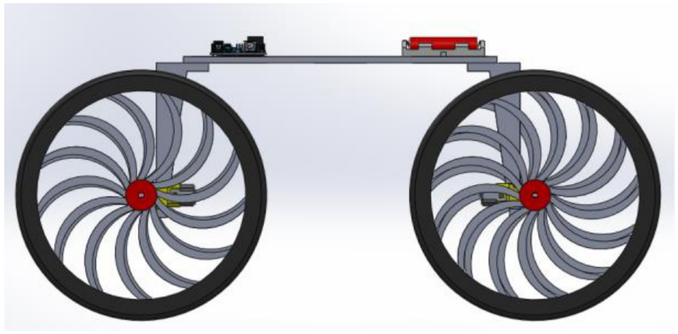
Залежність часу подолання перешкоди t від радіуса відстані між колесами



Залежність часу подолання перешкоди t від вильоту заднього колеса



Залежність часу подолання перешкоди t від вильоту переднього колеса



Модель колісної платформи з оптимальними параметрами у SOLIDWORKS

Оптимальні значення параметрів (мм)

- Радіус заднього колеса - 29.66
- Радіус переднього колеса - 29.94
- Відстань між колесами - 20.05
- Виліт заднього колеса - 14.41
- Виліт переднього колеса - 18.86

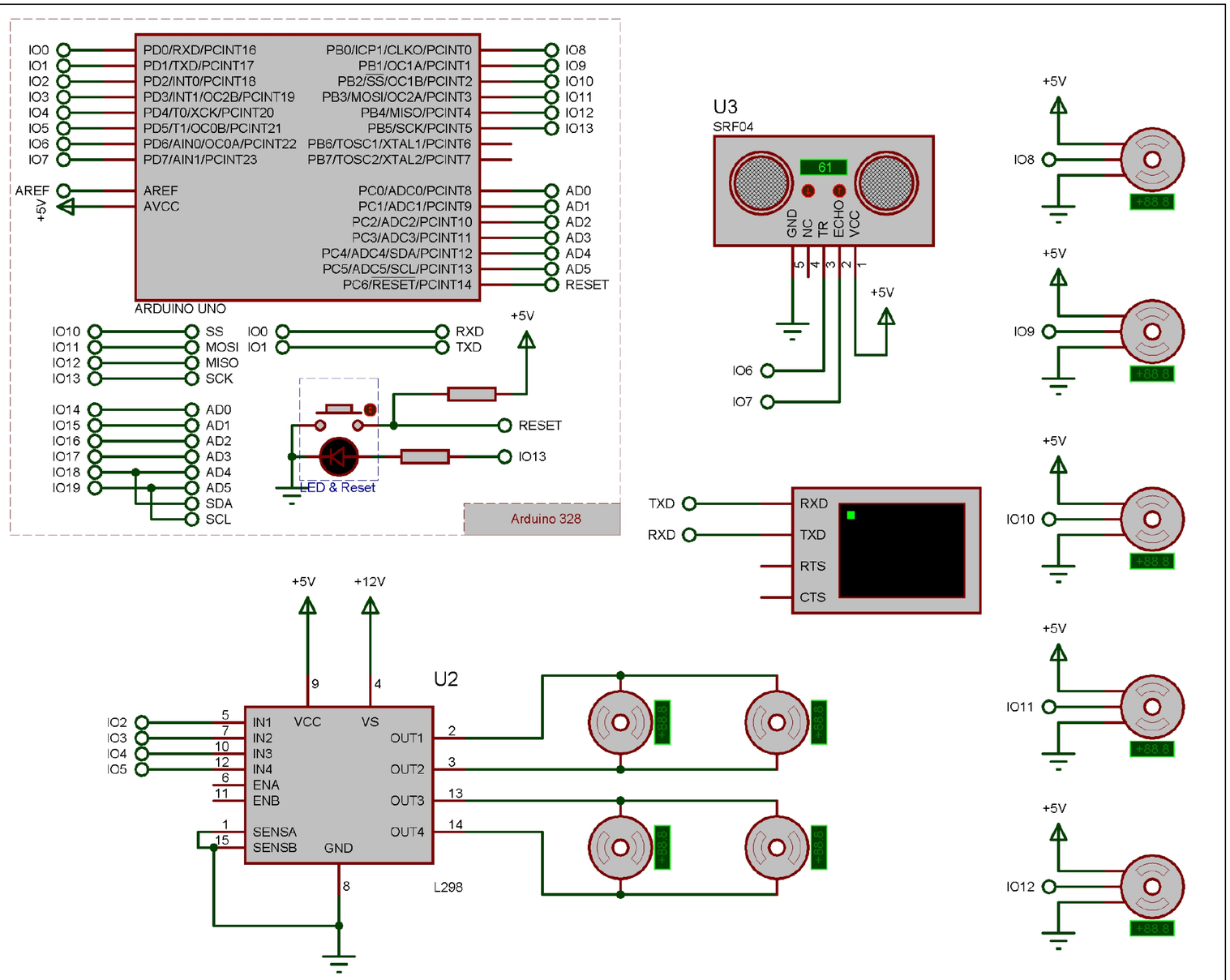
Оптимізований робот долає перешкоду за 5,25 секунди

					БР.ПМІ-04.00.00.000	
Зм.	Арк.	№ докум.	Підп.	Дата	Результати оптимізації	Літ. / Маса / Масштаб
Розроб.		Корбелюк Р.В.				
Перев.		Колеї В.Б.				
Т. контр.						Арк. / Аркушів
Н. контр.					ІФНТУНГ ПМІ-21-1К	
Затв.						

Формат А1

```
#include <Servo.h>
Servo servo1; Servo servo2;
Servo servo3; Servo servo4;
Servo servo5;

#define motor1pin1 2
#define motor1pin2 3
#define motor2pin1 4
#define motor2pin2 5
#define ECHOPIN 7
#define TRIGPIN 6

void setup() {...}

int ping(){
  digitalWrite(TRIGPIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIGPIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIGPIN, LOW);
  int distance =
pulseIn(ECHOPIN, HIGH);
  distance= distance/58;
  delay(50);
  return distance;}

int scan(){
   int angle=0;
   int y;
   while (angle<=130)
     {servo1.write(angle);
      delay(100);
            y=ping();
      //Serial.println(y);
            if (0<y && y<60)
return angle;
      angle+=13;}
   return -1;}

void stop(int t){
  digitalWrite(motor1pin1, 0);
  digitalWrite(motor1pin2, 0);
  digitalWrite(motor2pin1, 0);
  digitalWrite(motor2pin2, 0);
  delay(t);
}
```

```
void moveF(int t){
  digitalWrite(motor1pin1, 1);
  digitalWrite(motor1pin2, 0);
  digitalWrite(motor2pin1, 1);
  digitalWrite(motor2pin2, 0);
  delay(t);
}

void moveR(int t){
  digitalWrite(motor1pin1, 1);
  digitalWrite(motor1pin2, 0);
  digitalWrite(motor2pin1, 0);
  digitalWrite(motor2pin2, 1);
  delay(t);
}

void moveL(int t){
  digitalWrite(motor1pin1, 0);
  digitalWrite(motor1pin2, 1);
  digitalWrite(motor2pin1, 1);
  digitalWrite(motor2pin2, 0);
  delay(t);
}

void moveA(int angle){
Serial.println(angle);
if (0<=angle && angle<=43)
  moveL(43-angle);
else if (43<angle &&
angle<=86)
  moveF(43);
else
  moveR(angle-86);
}

void loop() {
  int angle;
  angle=scan();
  Serial.println(angle);
  if (angle==-1)
    moveA(random(0, 130));
  else
    moveA(angle);
  stop(100);
```




```
Serial.println(angle);
while (Serial.available() > 0)
{
int x = Serial.parseInt();
if (x<1000) servo1.write(x);
else if (x<2000) servo2.write(x-1000);
else if (x<3000) servo3.write(x-2000);
else if (x<4000) servo4.write(x-3000);
else if (x<5000) servo5.write(x-4000);
} }
```

					БР.ПМІ–04.00.00.000	
Зм.	Арк.	№ докум.	Підп.	Дата		
Розроб.		Корделяк Р.В.			Принципова електрична схема	Літ. / Маса / Масштаб
Перев.		Колеш В.Б.				Арк. / Аркушів
Т. контр.						
Н. контр.					ІФНТУНГ ПМІ–21–1К	
Затв.						

Формат А1