Методичні вказівки
до виконання лабораторних робіт з дисципліни
«Комп’ютерна інженерія та основи робототехніки»
зі спеціальності «Комп’ютерні науки»

Міністерство освіти і науки України
Вінницький національний технічний університет

Методичні вказівки до виконання лабораторних робіт з дисципліни «Комп'ютерна інженерія та основи робототехніки» зі спеціальності «Комп'ютерні науки»

Вінниця
ВНТУ
2025

Рекомендовано до видання Радою з якості освіти Вінницького національного технічного університету Міністерства освіти і науки України (протокол № 11 від 19.06.2025 р.)

Рецензенти:

М. С. Юхимчук, доктор технічних наук, професор
Д. Х. Штофель, кандидат технічних наук, доцент

Методичні вказівки до виконання лабораторних робіт з дисципліни «Комп'ютерна інженерія та основи робототехніки» зі спеціальності «Комп'ютерні науки» [Електронний ресурс] / уклад. Р. С. Белзецький. – Вінниця : ВНТУ, 2025. – (PDF, 43 с.)

В методичних вказівках наведені матеріали, призначені для виконання лабораторних робіт здобувачами першого (бакалаврського) рівня вищої освіти спеціальності «Комп'ютерні науки», які навчаються за освітньою програмою «Комп'ютерні науки» з дисципліни «Комп'ютерна інженерія та основи робототехніки». Матеріали, викладені в методичних вказівках, мають практичне спрямування, сприяють закріпленню теоретичних знань з питань імітаційного моделювання мобільних роботів в середовищі Webots.

ЗМІСТ

ВСТУП

Мета вивчення дисципліни «Комп'ютерна інженерія та основи робототехніки» – опанування студентами теоретичних основ і набуття практичних навичок фізичних та логічних принципів побудови електронних схем цифрових елементів і функціональних вузлів ПК, а також оволодінні основними принципами і методами побудови робототехнічних систем та їх програмування.

Основні *завдання* вивчення дисципліни «Комп'ютерна інженерія та основи робототехніки» полягають в тому, щоб ознайомити здобувачів вищої освіти з основними методами і засобами побудови електронних цифрових схем і функціональних вузлів; ознайомити з основними методами і засобами програмування і керування робочими органами робототехнічних систем. Здобувачі мають практично застосувати: основи технології програмування для керування робочими органами та організації зворотного зв'язку робототехнічних систем для реалізації поставленої задачі.

Вивчення навчальної дисципліни «Комп'ютерна інженерія та основи робототехніки» передбачає формування та розвиток у студентів компетентностей:

– **Інтегральної:**

Здатність розв'язувати складні спеціалізовані задачі та практичні проблеми у галузі комп'ютерних наук або у процесі навчання, що передбачає застосування теорій та методів інформаційних технологій і характеризується комплексністю та невизначеністю умов.

– Загальних:

ЗК01. Здатність до абстрактного мислення, аналізу та синтезу.

ЗК02. Здатність застосовувати знання у практичних ситуаціях.

ЗК07. Здатність до пошуку, оброблення та аналізу інформації з різних джерел.

– Спеціальних (фахових):

СК18. Здатність проводити дослідження в області робототехніки. Здатність застосовувати методи аналізу та синтезу при дослідженні робототехнічних систем.

Програмні результати вивчення дисципліни

Результати вивчення даної дисципліни деталізують такі програмні результати навчання:

РН18. Виконувати синтез робототехнічних систем та здійснювати аналіз їх динамічних характеристик. Розробляти, програмувати та керувати роботами на основі давачів робототехнічних комплексів та правильно їх експлуатувати. Розробляти програмне забезпечення для сучасних робототехнічних систем.

Лабораторна робота № 1

МОДЕЛЮВАННЯ У WEBOTS

Мета роботи – ознайомитися з інтерфейсом користувача і основними концепціями Webots. Створити симуляцію, яка містить просте середовище: арену з підлогою і стінами, кілька об’єктів, робота і програму контролера, яка змусить робота рухатися.

Хід роботи

1. Запуск Webots.

Запустити Webots, двічі клацнувши кнопкою курсору на його значку на робочому столі (рис. 1.1). Якщо Webots запускається вперше на комп’ютері, можливо буде запропоновано опцію вибрати графічну тему. Також може бути запропоновано участь в екскурсії з гідом Webots. Після запуску відобразиться середовище розробки (рис. 1.2).

Рисунок 1.1 – Значок Webots

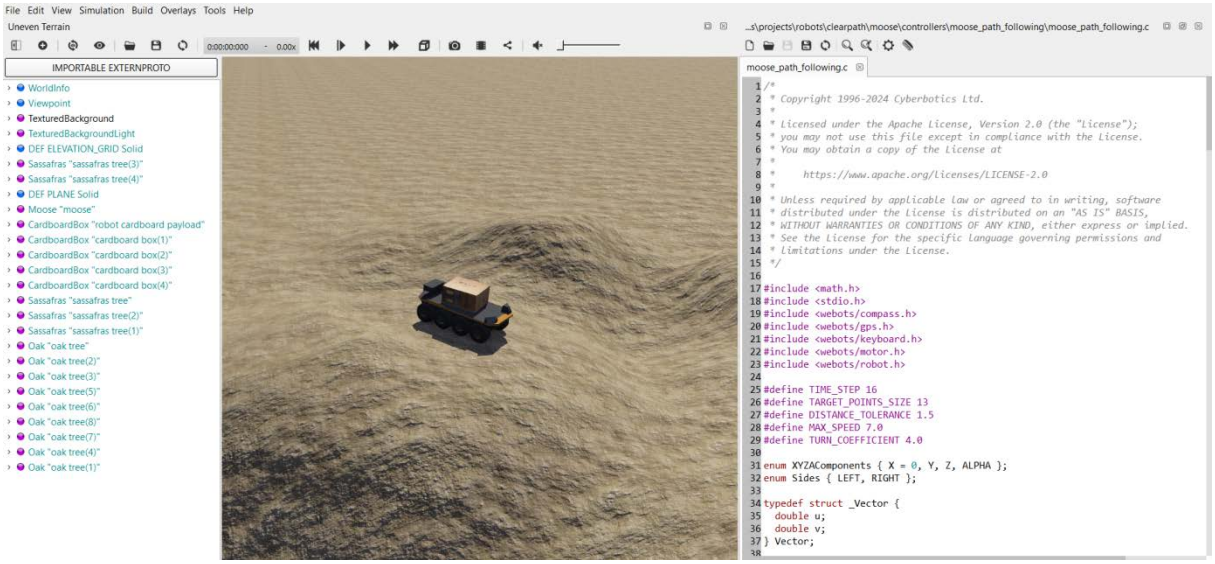

Рисунок 1.2 – Середовище розробки Webots

2. Створення нового світу.

Світ являє собою файл, що містить інформацію, такі як об’єкти, їх вигляд, взаємодію один з одним, колір неба, визначення сили тяжіння, тертя, маси і т.д. Світ визначає початковий стан моделювання. Такі елементи називаються вузлами та ієрархічно організовані в дереві сцени. Також вузол

може містити підвузли. Світ зберігається в файлі з розширенням .wbt. Формат файлу заснований на мові VRML97. Файли світу зберігаються в каталозі з іменем *worlds*.

Необхідно створити новий проєкт з меню, вибравши «*New Project Directory*» (рис. 1.3) та слідуючи інструкціям:

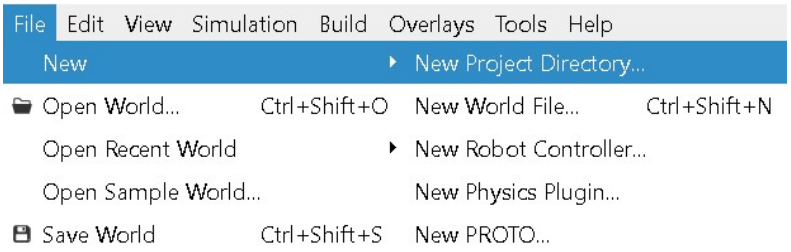

Рисунок 1.3 – Створення нового проєкту

Назвати каталог проєкту *my_first_simulation* (рис. 1.4).

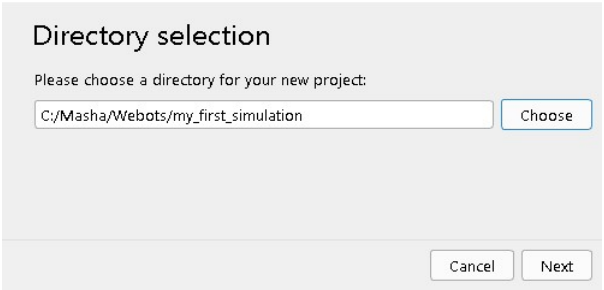

Рисунок 1.4 – Створення каталогу проєкту my_first_simulation

Назвати файл світу *my_first_simulation.wbt* замість назви *empty.wbt* (рис. 1.5).

Встановити всі прапорці, «*Add a rectangle arena*» замовчуванням не відзначений.

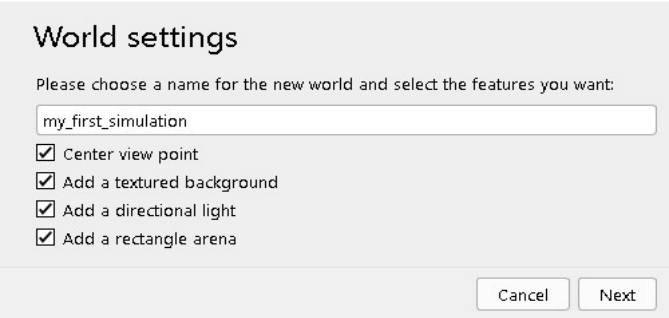

Рисунок 1.5 – Створення файлу світу *my_first_simulation.wbt*

На тривимірному зображенні має відображатися квадратна арена з шахматним візерунком (рис. 1.6). Є можливість переміщати точки огляду в 3D-вигляді за допомогою миші: лівою кнопкою, правою кнопкою і колеса.

Рисунок 1.6 – Створена квадратна арена з шахматною підлогою

Вузли Webots, що зберігаються в файлах світу, організовані в деревоподібну структуру – дерево сцени. Дерево сцени можна переглядати у двох підвікнах головного вікна: тривимірне уявлення, а уявлення дерева сцени – ліворуч. У дереві сцени можна змінювати вузли і поля.

Проєкт має містити наступні вузли:

- *World Info*: містить глобальні параметри моделювання.
- *Viewpoint*: визначає параметри камери основної точки огляду.
- *Textured Background*: визначає фон сцени (ви повинні побачити гори вдалині, якщо ні – трохи поверніть точку огляду).
- *Textured Background Light*: визначає світло, пов'язане з зазначеним вище фоном.
- *Rectangle Arena*: визначає об'єкт, який відображається в цій сцені.

Кожен вузол має кілька властивостей, які називаються полями. Необхідно змінити ці поля, щоб змінити прямокутну арену.

Щоб відкрити вузол і відобразити його поля, необхідно двічі натиснути *Rectangle Arena* в дереві сцени.

Наступний крок, вибрати поле *floor Tile Size* і встановити для нього значення 0.25 0.25.

Змінити значення поля *wall Height поле* на 0.05 замість 0.1 (рис. 1.7). Стіна арени має змінити висоту.

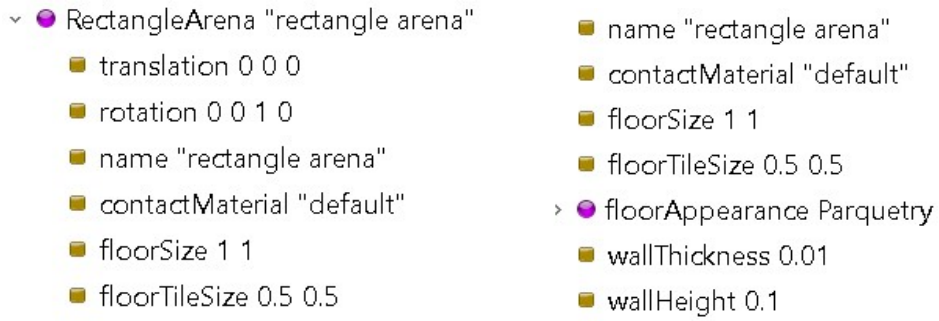

Рисунок 1.7 – Параметри *floor Tile Size* та *wall Height*

Щоб змінити вигляд арени, необхідно обрати параметр *floor Appearance Parquetry* та видалити його. Значення цього параметра змінить-ся на *NULL* (рис. 1.8).

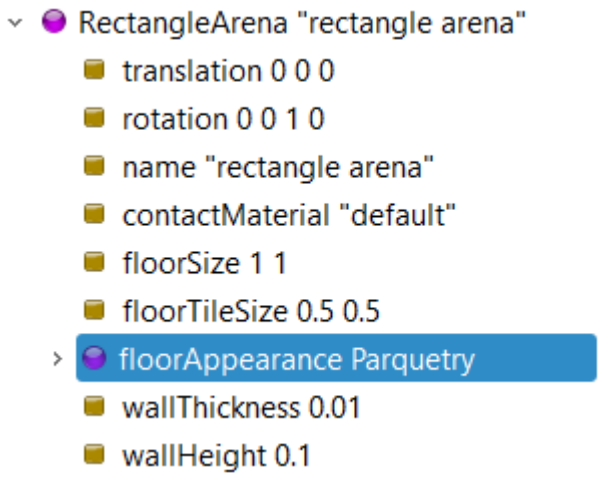

Рисунок 1.8 – Параметр *floor Appearance Parquetry*

Наступним кроком необхідно вибрати параметр *floor Appearance Parquetry* та натиснути *Add New* (рис. 1.9). Після цього можна змінювати вигляд арени.

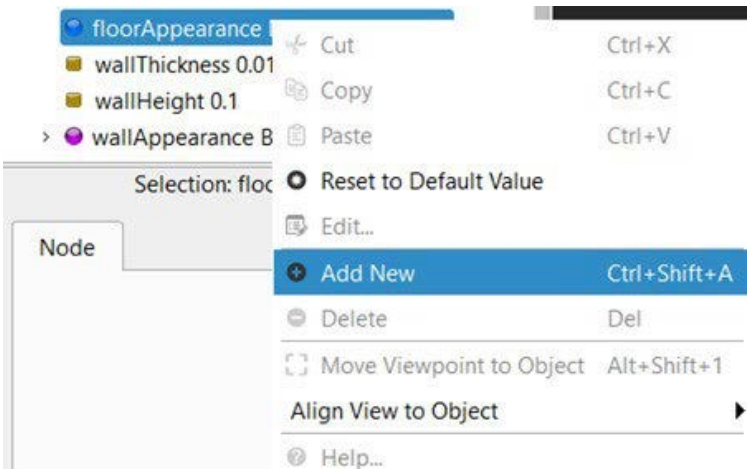

Рисунок 1.9 – Зміна параметру *floor Appearance Parquetry*

У діалоговому вікні вибрати *PROTO nodes* (*Webots Projects)/ appearances / Brushed Steel (PBR Appearance)*, та натиснути *Add* (рис. 1.10).

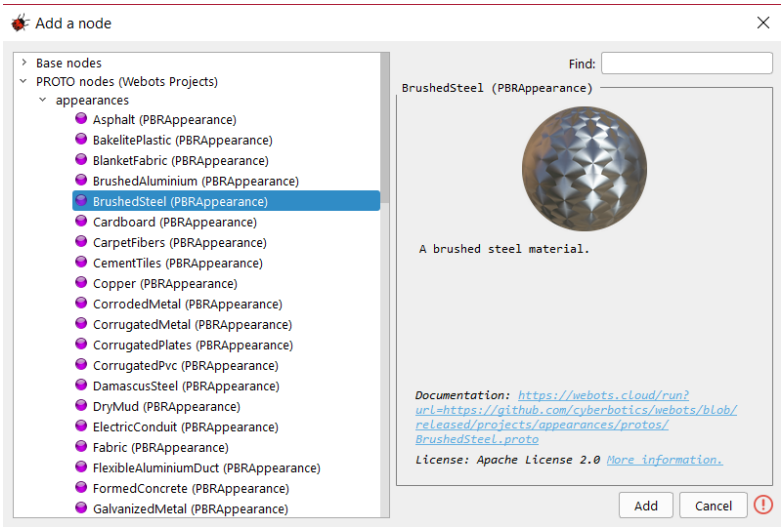

Рисунок 1.10 – Зміна параметру *floor Appearance Parquetry*

Оновлений вигляд арени (рис. 1.11).

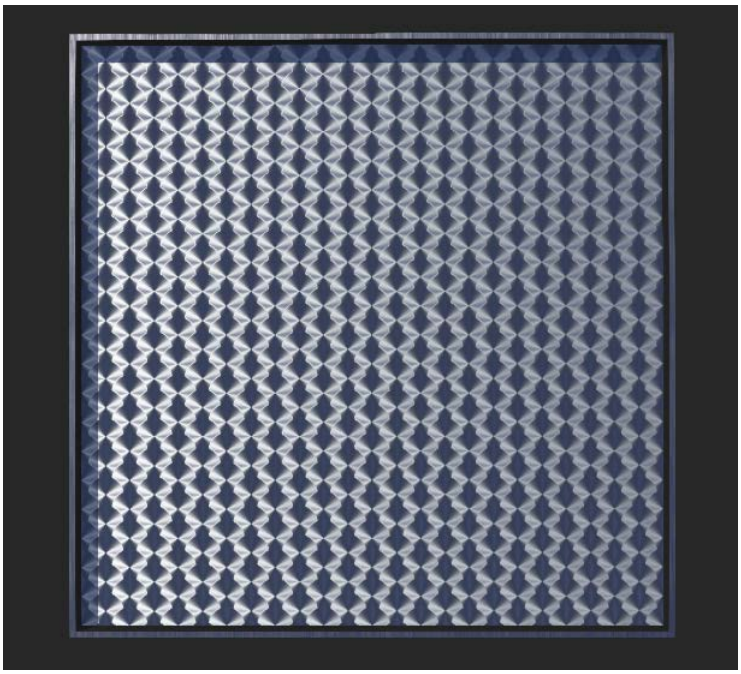

Рисунок 1.11 – Оновлений вигляд арени

Можна змінити параметр *Textured Background*. Для цього треба обрати «*factory*» (рис. 1.12). Також необхідно змінити параметр *Textured Background Light* (рис. 1.13).

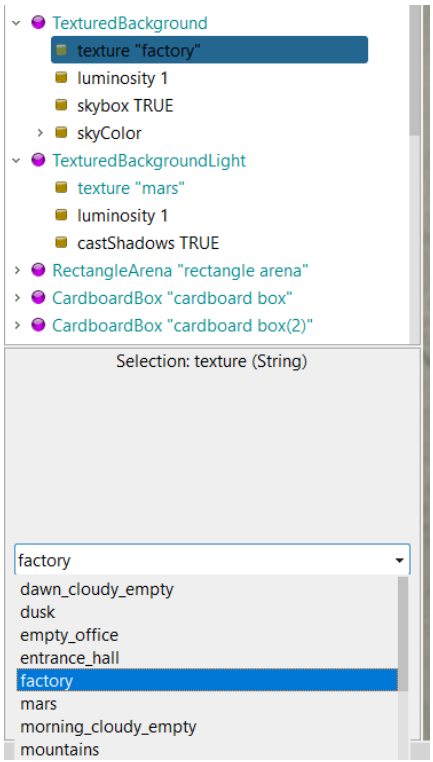

Рисунок 1.12 – Зміна параметру *Textured Background*

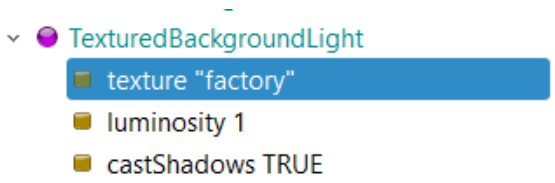

Рисунок 1.13 – Зміна параметру *Textured Background Light*

Після цього арена набуде іншого вигляду (рис. 1.14).

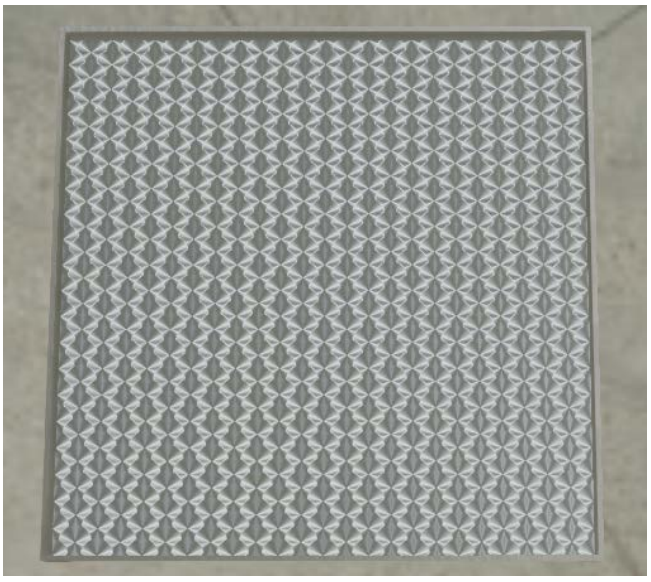

Рисунок 1.14 – Оновлений вигляд арени

У дереві сцени поля відображаються іншим кольором, якщо вони відрізняються від значень що вказані за замовчуванням.

Для додавання кількох об’єктів, необхідно двічі натиснути на значок *Rectangle Arena* в дереві сцени, щоб відкрити і вибрати його. Натиснути кнопку *Add* (рис. 1.15). У діалоговому вікні вибрати *PROTO nodes (Webots Projects) / objects / factory / containers / Cardboard Box (Solid)* (рис. 1.16). Посеред арени повинна з’явитися велика коробка.

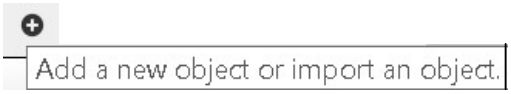

Рисунок 1.15 – Add кнопка для створення об’єкту

Рисунок 1.16 – Створення великої коробки

Наступним кроком необхідно двічі натиснути по ній в дереві сцени, щоб відкрити. Та змінити її параметр size на 0.1 0.1 0.1 замість 0.6 0.6 0.6. Значення параметра *translation* необхідно змінити на 0 0.05 0 замість 0 0.3 0 (рис 1.17).

Додатково можна скористатися зеленою стрілкою, що з’являється у 3D-вигляді, для налаштування її параметра *translation*.

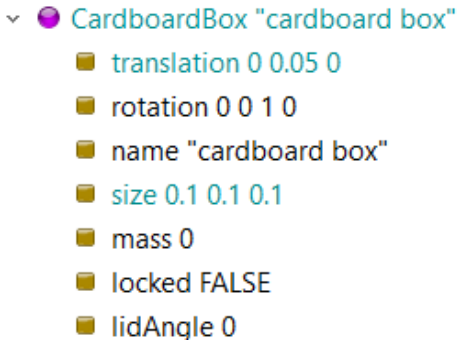

Рисунок 1.17 – Зміна параметрів для Cardboard Box

Утримуючи клавішу «Shift», слід перетягнути коробку в тривимірному вигляді та розмістити її на арені.

Далі необхідно вибрати коробку, скопіювати її і вставити. Повторне утримання клавіші «Shift» дозволить перемістити нове поле в інше місце.

Аналогічно створюються ще три коробки. В результаті формується арена, показана на рисунку 1.18.

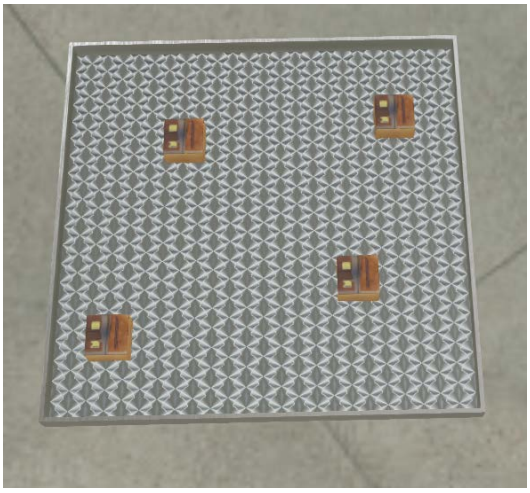

Рисунок 1.18 – Створена арена

Для обертання прямокутників навколо осі можна скористатися стрілками обертання. Також для обертання коробок можна змінити значення параметра rotation для вузлів *Cardboard Box* у дереві сцени.

Після завершення редагування сцени її необхідно зберегти.

3. Додавання робота

Elisa-3 – це мініатюрний робот Arduino від GCtronic, працює на базі процесора ATmega 2560 і має велику кількість сенсорів. Робот може самостійно заряджатися за допомогою зарядної станції.

На цьому етапі до сцени додається модель Elisa-3. Перед цим необхідно переконатися, що симуляцію зупинено, а віртуальний час завершено і його значення дорівнює нулю. Якщо ці умови не виконано, симуляцію слід скинути за допомогою кнопки *Reset*.

Коли світ Webots модифікується для подальшого збереження, важливо, щоб симуляція була припинена і перезавантажена в початковий стан, тобто віртуальний лічильник часу на головній панелі інструментів повинен показувати 0: 00: 00: 000. В іншому випадку при кожному збереженні стан кожного 3D-об’єкта може накопичувати помилки. Отже, будь-які модифікації світу повинні виконуватися в такому порядку: *зупинити → скинути → змінити → зберегти симуляцію*.

Не потрібно створювати робота з нуля, просто потрібно імпортувати Elisa-3 вузол. Цей вузол насправді є вузлом *PROTO*, як *Rectangle Arena* і такий, як *Cardboard Box*, що розглядались раніше.

Прототипування дозволяє створювати власні об’єкти і повторно використовувати їх.

Необхідно вибрати останній вузол *Cardboard Box* у дереві сцени. Далі слід натиснути кнопку *Add* у верхній частині дерева сцени. У діалоговому вікні потрібно обрати *PROTO nodes(Webots Projects)/robots/gctronic/elisa/ Elisa-3 (Robot)* (рис. 1.19).

На арені з’явиться робот. Його можна пересувати та розвертати, аналогічно як і з коробками.

Після розміщення робота симуляцію слід зберегти та запустити за допомогою кнопки *Run real-time*.

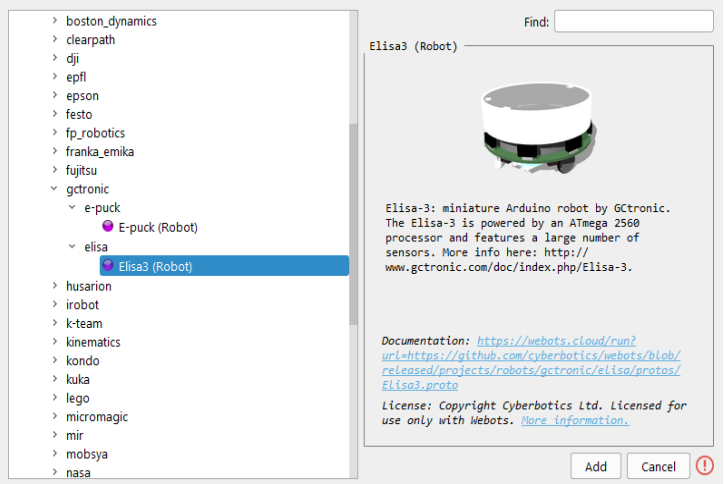

Рисунок 1.19 – Створення моделі Elisa-3

Робот повинен рухатися і уникати перешкод. Це тому, що у нього є контролер за замовчуванням з таким алгоритмом руху.

Далі можна налаштовувати фізику об’єктів. Щоб застосувати сили тяжіння до робота, необхідно натиснути Alt + ліву кнопку миші та виконати перетягування (на деяких клавіатурах Mac слід використовувати клавішу ⌥ Option). У середовищі Linux додатково потрібно натиснути клавішу Ctrl разом із Alt, лівою кнопкою миші та перетягуванням. Застосування сили тяжіння до вузлів *Cardboard Box* є неможливим, оскільки за замовчуванням вони не мають маси і вважаються прикріпленими до підлоги. Щоб активувати фізичну взаємодію з такими вузлами, необхідно надати масу об’єкту в полі *mass* (наприклад, 0,1 кг) (рис. 1.20). Після цього до вузлів *Cardboard Box* можна застосовувати силу.

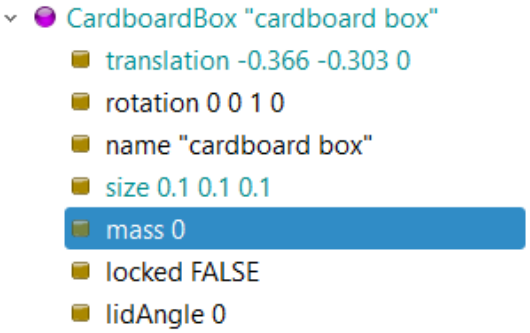

Рисунок 1.20 – Параметер *mass* на *Cardboard Box* вузлах

Моделювання можна припиняти, запускати покроково, в реальному часі або в пришвидшених режимах.

Наступним етапом є модифікація сцени зі зменшенням кроку симуляції фізики. Це забезпечить підвищену точність і стабільність симуляції, хоча й призведе до зниження її максимальної швидкості.

Перш за все, необхідно призупинити симуляцію та повернути її до початкового стану. У поданні «Дерево сцени» слід розгорнути вузол *World Info* та встановити значення поля *basic Time Step* рівним 16. Після цього симуляцію потрібно зберегти (рис. 1.21).

Рисунок 1.21 – Параметр *basic Time Step* у вузлі *World Info*

4. Створення нового контролера

У Webots контролер – це програма, яка визначає поведінку робота. Для створення контролера Webots можна використовувати наступні мови програмування: C, C++, Java, Python, MATLAB, ROS тощо. Контролери на C, C++ і Java потребують компіляції перед запуском, тоді як Python і MATLAB працюють без неї. В даній лабораторній роботі використовується мова програмування C, але код доступний також для C++, Java, Python і MATLAB.

Controller поле *Robot* вузла визначає, який контролер в даний момент пов'язаний з роботом. Звертаю Вашу увагу, що один і той же контролер може використовуватися декількома роботами, але робот може використовувати тільки один контролер одночасно. Кожен контролер виконується в окремому дочірньому процесі.

Необхідно створити новий контролер мовою C (або іншою за вибором), який має назву *elisa_go_forward* (для мов C++ і Java відповідно слід використовувати назву *Elisa Go Forward)*. Для цього слід скористатися меню *Wizards / New Robot Controller... (рис. 1.22).*

Результатом буде створення нового каталогу *elisa_go_forward* у папці *my_first_simulation/controllers*. Після створення слід обрати опцію відкриття вихідного файла в текстовому редакторі.

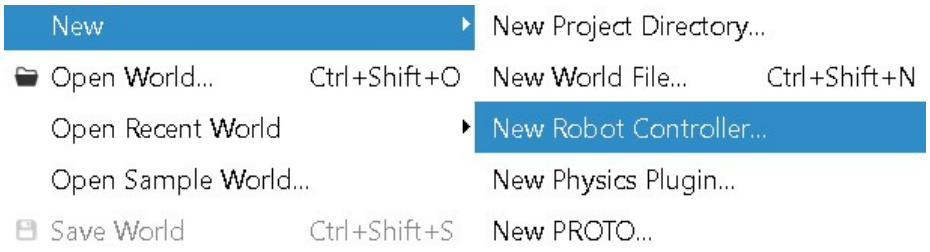

Рисунок 1.22 – Створення контролера

Новий вихідний файл відображається у вікні текстового редактора Webots. Тепер необхідно зв’язати з вузлом новий *elisa_go_forward* контролер Elisa-3.

У поданні дерева сцени треба вибрати поле *controller* вузла *Elisa-3*, потім використати редактор поля в нижній частині подання дерева сцени: натиснути кнопку *Select* . і вибрати зі списку *elisa_go_forward* (рис 1.23).

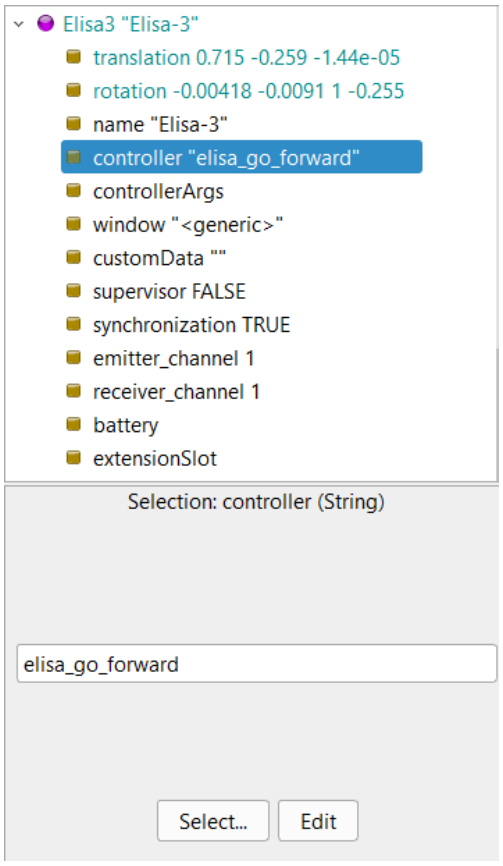

Рисунок 1.23 – Підключення контролера

Наступним кроком є зміна програми контролера, (зміни показано нижче) перекомпілювати її та запустити:

```
#include <webots/robot.h>
#include <webots/motor.h>
#include <webots/position_sensor.h>

#define TIME_STEP 64
#define MAX_SPEED 6.28
#define TARGET_ROTATION 10.0  // значення в радіанах

int main() {
  wb_robot_init();

  // Отримати пристрої двигунів
```

```
  WbDeviceTag left_motor = wb_robot_get_device("left wheel
motor");
  WbDeviceTag right_motor = wb_robot_get_device("right
wheel motor");
  // Отримати значення датчики положення
  WbDeviceTag left_sensor = wb_robot_get_device("left
wheel sensor");
  WbDeviceTag right_sensor = wb_robot_get_device("right
wheel sensor");

  // Увімкнути датчики положення
  wb_position_sensor_enable(left_sensor, TIME_STEP);
  wb_position_sensor_enable(right_sensor, TIME_STEP);

  // Встановити мотори в режим швидкості
  wb_motor_set_position(left_motor, INFINITY);
  wb_motor_set_position(right_motor, INFINITY);

  // Початково встановити швидкість двигунів на 0
  wb_motor_set_velocity(left_motor, 0.0);
  wb_motor_set_velocity(right_motor, 0.0);

  // Зачекати один крок для ініціалізації датчиків
  wb_robot_step(TIME_STEP);
   // Отримати початкове положення
   double start_left =
wb_position_sensor_get_value(left_sensor);
   double start_right =
wb_position_sensor_get_value(right_sensor);

   // Почати рух вперед
   wb_motor_set_velocity(left_motor, MAX_SPEED);
   wb_motor_set_velocity(right_motor, MAX_SPEED);

   // Запуск симуляції
   while (wb_robot_step(TIME_STEP) != -1) {
     double left_rot =
wb_position_sensor_get_value(left_sensor) - start_left;
     double right_rot =
wb_position_sensor_get_value(right_sensor) - start_right;

     // Середнє значення обертання
     double avg_rot = (left_rot + right_rot) / 2.0;

     if (avg_rot >= TARGET_ROTATION) {
       // Зупинити робота
       wb_motor_set_velocity(left_motor, 0.0);
```

```
      wb_motor_set_velocity(right_motor, 0.0);
      break;
    }
  }
  wb_robot_cleanup();
  return 0;
}
```

Змінений вихідний код необхідно зберегти (*File/Save Text File*) та скомпілювати (*Build/Build*). У разі наявності помилок компіляції слід їх виправити. Якщо під час виконання Webots пропонує скинути або перезавантажити світ, слід обрати варіант *Reset* і запустити симуляцію.

Якщо помилок не виявлено, робот почне рухатись прямо. Робот рухатиметься з максимальною швидкістю, доки колеса не зроблять оберт на 10 радіанів, після чого зупиниться.

У каталозі *controllers* проєкту створюється підкаталог, який містить контролер *elisa_go_forward*. Після компіляції в цьому каталозі формується бінарний файл із відповідною назвою (*.exe* – для середовища *Windows*). Назва каталогу контролера має відповідати імені створеного виконуваного файла.

5. Розширення функціоналу контролера

Колеса роботів часто керуються за допомогою швидкості, а не положенням, як в попередньому прикладі.

Здебільшого швидкість руху робота задається частотою обертання колеса, а не за положенням колеса, як це було показано в попередньому прикладі. Щоб керувати двигунами коліс за швидкістю, необхідно встановити цільове положення на нескінченність і встановити бажану швидкість.

Для цього необхідно змінити програму контролера, як показано нижче, перекомпілювати її та запустити:

```
#include <webots/robot.h>
#include <webots/motor.h>

#define TIME_STEP 64
#define MAX_SPEED 6.28

int main() {
  wb_robot_init();

  // Отримання доступу до двигунів
  WbDeviceTag left_motor = wb_robot_get_device("left
wheel motor");
  WbDeviceTag right_motor = wb_robot_get_device("right
wheel motor");

  // Встановлення режиму керування швидкістю (режим не-
скінченного обертання)
```

```
  wb_motor_set_position(left_motor, INFINITY);
  wb_motor_set_position(right_motor, INFINITY);

  // Встановлення максимальної швидкості для неперервного
руху
  wb_motor_set_velocity(left_motor, MAX_SPEED);
  wb_motor_set_velocity(right_motor, MAX_SPEED);

  // Безперервне виконання симуляції
  while (wb_robot_step(TIME_STEP) != -1) {
    // Робот буде постійно рухатися вперед
  }

  wb_robot_cleanup();
  return 0;
}
```

Після успішної компіляції робот починає неперервно рухатися зі швидкістю обертання коліс 0,2 радіана за секунду. В іншому випадку необхідно переконатися, що вихідний код було скомпільовано. Для цього слід скористатися пунктом меню *Build/Build* або натиснути піктограму шестерні над областю коду. Повідомлення про помилки компіляції відображаються в консолі червоним кольором. У разі їх наявності слід виправити помилки та повторити компіляцію. Після цього необхідно перезавантажити світ.

Варіанти завдань

№	Арена	Фон і Світло	Об'єкти		Робот	Контролер
			назва	к-ть		
1	BakelitePlastic	Dusk	Ball	3	Elisa3	Оминання перешкод
2	BrushedAluminium	Empty_office	PlasticCrate	2	E-puck	Оминання перешкод
3	CementTiles	Entrance_hall	CardboardBox	4	Elisa3	Рух по колу
4	Asphalt	Noon_building_overcast	PingPongBall	3	E-puck	Рух по колу
5	CorrodedMetal	Mars	Ball	2	Elisa3	Рух вздовж стіни
6	CorrugatedPlates	Noon_park_empty	PlasticCrate	4	E-puck	Рух вздовж стіни
7	Copper	Noon_cloudy_countryside	CardboardBox	3	Elisa3	Рух заднім ходом
8	DamascusSteel	Music_hall	WoodenBox	2	E-puck	Рух заднім ходом
9	ElectricConduit	Stadium	RobocupSoccerBall	4	Elisa3	Рух зигзагом
10	HammeredCopper	Stadium_dry	PingPongBall	2	E-puck	Рух зигзагом
11	Marble	Dusk	PingPongBall	3	Elisa3	Рух квадратною траєкторією
12	MetalPipePaint	Empty_office	Ball	2	E-puck	Рух квадратною траєкторією

№	Арена	Фон і Світло	Об'єкти		Робот	Контролер
			назва	к-ть		
13	MetalStainlessSteelCable	Entrance_hall	PlasticCrate	4	Elisa3	Рух траєкторією цифри 8
14	OldSteel	Noon_building_overcast	CardboardBox	3	E-puck	Рух траєкторією цифри 8
15	PorcelainChevronTiles	Mars	WoodenBox	2	Elisa3	Рух трикутною траєкторією
16	RoughConcrete	Noon_park_empty	PlasticCrate	4	E-puck	Оминання перешкод
17	RoughPine	Noon_cloudy_countryside	CardboardBox	3	Elisa3	Рух по колу заднім ходом
18	RustyMetal	Music_hall	WoodenBox	2	E-puck	Рух по колу заднім ходом
19	ScratchedPaint	Stadium	RobocupSoccerBall	4	Elisa3	Рух вздовж стіни заднім ходом
20	ThreadMetalPlate	Stadium_dry	PingPongBall	3	E-puck	Рух вздовж стіни заднім ходом
21	VarnishedPine	Dusk	CardboardBox	2	Elisa3	Рух вздовж стіни
22	WornBurlap	Empty_office	PingPongBall	3	E-puck	Рух вздовж стіни
23	BrushedSteel	Entrance_hall	Ball	4	Elisa3	Рух трикутною траєкторією
24	CorrugaredPvc	Noon_building_overcast	PlasticCrate	2	E-puck	Рух трикутною траєкторією
25	DryMud	Mars	WoodenBox	3	Elisa3	Рух траєкторією цифри 8
26	FlexibleAluminiumDuct	Noon_park_empty	RobocupSoccerBall	4	E-puck	Рух траєкторією цифри 8
27	FormedConcrete	Noon_cloudy_countryside	PingPongBall	2	Elisa3	Рух заднім ходом
28	GalvanizedMetal	Music_hall	PingPongBall	3	E-puck	Рух заднім ходом

Зміст звіту

1. Звіт з лабораторної роботи має бути виконаний на аркушах формату А4.
2. Звіт має містити: назву лабораторної роботи, її мету і короткі теоретичні відомості.
3. У розділі «Результати виконання лабораторної роботи» студент має представити результати покрокового виконання лабораторної роботи.
4. Написати висновки до даної лабораторної роботи.
5. Оформити звіт та подати його викладачу на захист.

Контрольні запитання

1. Що таке Webots?
2. Які основні компоненти містить світ у Webots?
3. Що таке вузли в Webots і як вони організовані?
4. Які файли використовуються для збереження світу в Webots?
5. Як додати об’єкт або робота у світ Webots?
6. Як створити новий контролер для робота в Webots?
7. Чому для контролерів на C, C++ і Java потрібна компіляція?
8. Як пов’язати контролер з роботом у Webots?
9. Якими мовами програмування можна писати контролери в Webots?
10. Як налаштувати керування роботом через швидкість коліс?
11. Як змінити параметри симуляції в Webots для підвищення точності?
12. Як зберегти та скомпілювати зміни в контролері для робота?
13. Який крок симуляції фізики є оптимальним для підвищення стабільності?
14. Як додати власні об’єкти та налаштувати їх параметри у Webots?

Лабораторна робота № 2

ПРОГРАМУВАННЯ КОНТРОЛЕРІВ У WEBOTS

Мета роботи – ознайомитись з основами програмування роботів в середовищі Webots. Набути навичок побудови зв'язків між вузлами дерева сцени і API контролера, ініціалізації пристроїв робота, отримання значень сенсорів та керування виконавчими пристроями робота.

Хід роботи

1. *Новий світ і новий контролер*

Створити новий контролер мовою C (або іншою за вибором) з іменем *elisa_avoid_collision* (для мов C++ та Java — *Elisa Avoid Collision*) за допомогою майстра налаштування.

Перед створенням контролера необхідно додати до сцени модель *Elisa-3* (рис. 2.1). Після цього у властивостях вузла *Elisa-3* слід змінити значення поля *controller*, встановивши посилання на щойно створений контролер (рис. 2.2).

Для ініціалізації контролера потрібно скористатися пунктом меню *Wizards/New Robot Controller*..., після чого обрати мову програмування та вказати ім'я файлу (рис. 2.3).

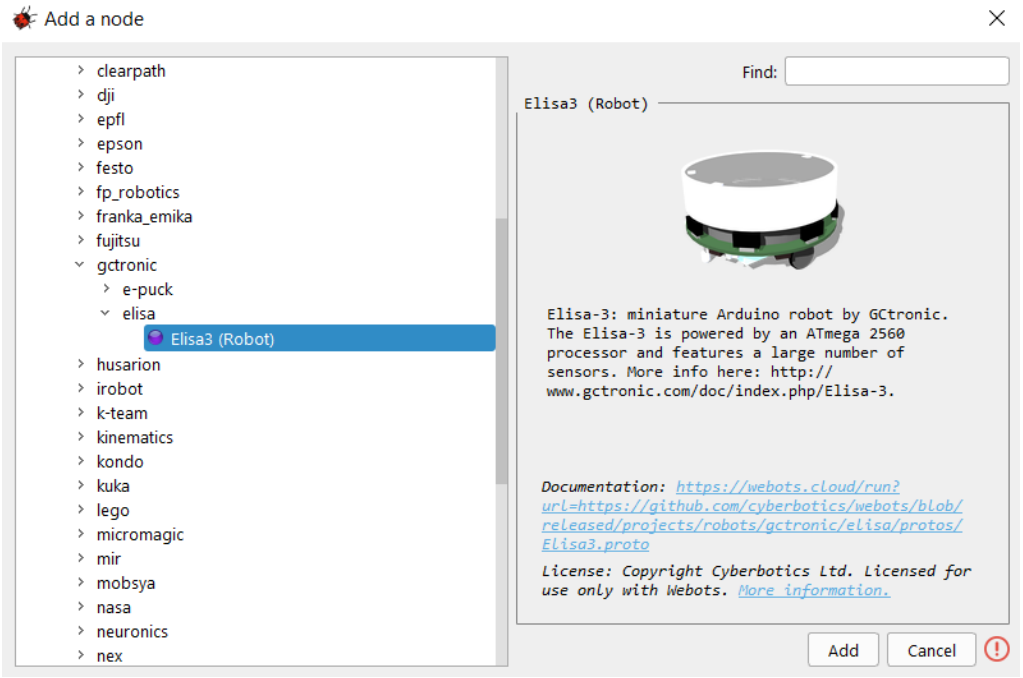

Рисунок 2.1 – Створення моделі Elisa-3

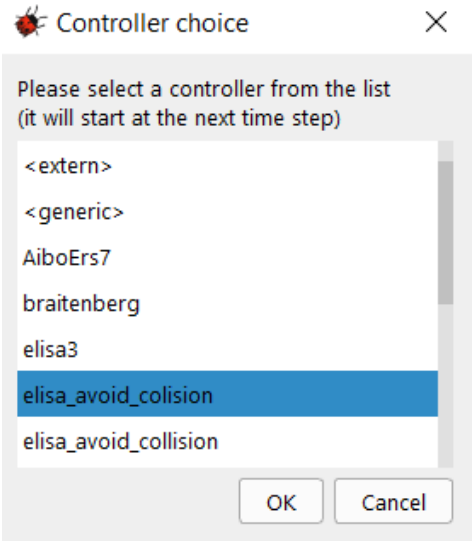

Рисунок 2.2 –Підключення контролера

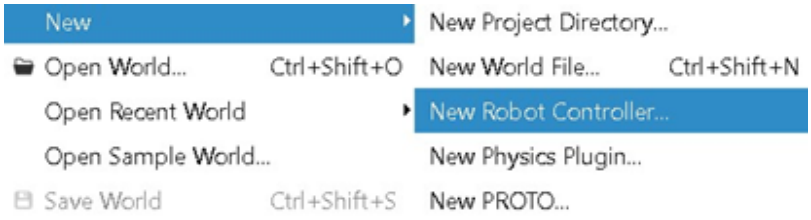

Рисунок 2.3– Створення контролера

2. Робот Elisa-3.

Elisa-3 – це наступна версія робота Elisa, розроблений на іншому мікроконтролері та включає наступні компоненти:

- мікроконтролер Atmel 2560 (сумісний з Arduino);
- центральний світлодіод RGB;
- 8 зелених світлодіодів навколо робота;
- ІЧ випромінювачі;
- 8 ІЧ-сенсорів наближення (Vishay Semiconductors Reflective Optical Sensor);
- 4 наземні сенсори (Fairchild Semiconductor Miniature Reflective Object Sensor);
- 3-осьовий акселерометр (Freescale MMA7455L);
- РЧ радіо для зв’язку (Nordic Semiconductor nRF24L01+);
- роз’єм micro USB для програмування, налагодження та зарядки;
- ІЧ-приймач;
- 2 двигуни постійного струму.

Робот може самостійно заряджатися за допомогою зарядної станції, розташування сенсорів на корпусі робота показано на рисунку 2.4.

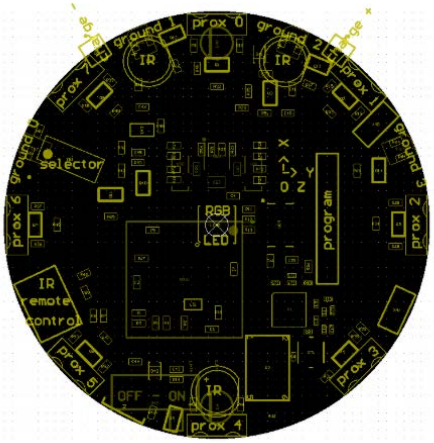

Рисунок 2.4 – Сенсори моделі Elisa-3

Основні компоненти робота Elisa-3, і їх фізичне розміщення показано на рисунку 2.5.

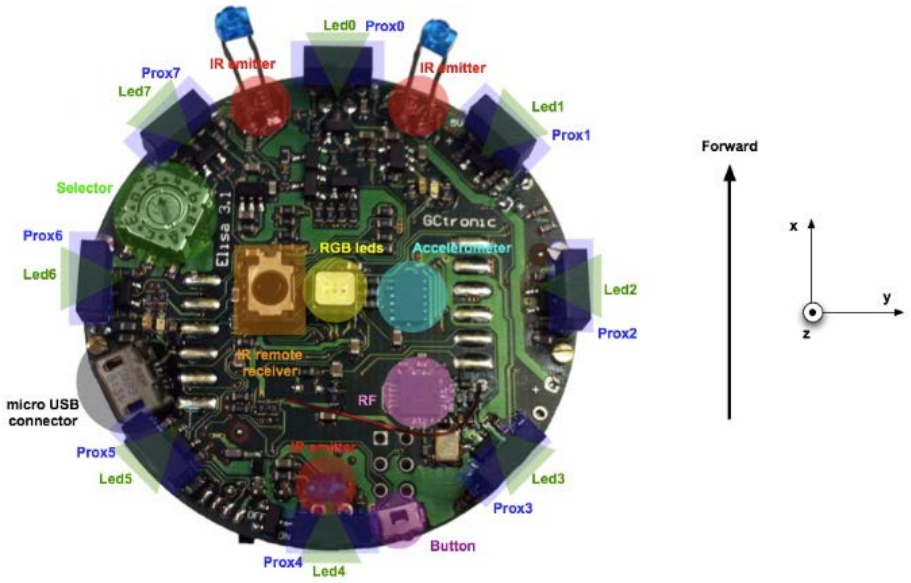

Рисунок 2.5 – Вид згори на модель Elisa-3

Програмування контролера

В даній роботі необхідно реалізувати Базовий алгоритм дій мобільного робота – уникнення перешкод. Алгоритм полягає в тому, що робот повинен рухатися вперед до моменту, поки сенсори відстані не зафіксують наявність перешкоди. Після виявлення перешкоди має бути виконано поворот у напрямку, вільному від об’єктів.

Підключення бібліотек

Необхідно підключити стандартні заголовочні файли Webots:

- robot.h – основні функції управління роботом (наприклад, ініціалізація, завершення роботи);

- distance_sensor.h – функції для роботи з сенсорами відстані;
- motor.h – функції для управління двигунами.

```
#include <webots/robot.h>
#include <webots/distance_sensor.h>
#include <webots/motor.h>
```

Оголошення змінних

- TIME_STEP 64 – часова затримка для кожного кроку симуляції (64 мілісекунди – типовий стандарт Webots).
- WbDevice Tag ds[8] – масив для зберігання ідентифікаторів 8 сенсорів відстані.
- WbDevice Tag left_motor, right_motor – змінні для лівого і правого двигунів.
- ds_names – масив імен сенсорів (ps0 до ps7), які використовуються.

```
#define TIME_STEP 64
WbDeviceTag ds[8];
WbDeviceTag left_motor, right_motor;
const char *ds_names[8] = {
"ps0", "ps1", "ps2", "ps3", "ps4", "ps5", "ps6", "ps7"
};
```

Ініціалізація Webots API і пристроїв

Це стандартна команда для запуску контролера. Без неї робот не зможе працювати в симуляції.

```
wb_robot_init();
```

Ініціалізація сенсорів

Для реалізації функціональності уникнення перешкод необхідно виконати ініціалізацію сенсорів відстані. У циклі здійснюється доступ до кожного з восьми інфрачервоних сенсорів за їхніми іменами (наприклад, ps0, ps1 і т.д.) з використанням функції wb_robot_get_device(ds_names[i]). Активізація сенсорів здійснюється за допомогою команди wb_distance_sensor_enable(ds[i], TIME_STEP), що забезпечує отримання даних кожні 64 мілісекунди.

```
for (int i = 0; i < 8; i++) {
  ds[i] = wb_robot_get_device(ds_names[i]);
  wb_distance_sensor_enable(ds[i], TIME_STEP);
}
```

Ініціалізація двигунів

Доступ до виконавчих механізмів – лівого та правого двигунів – виконується через відповідні імена пристроїв. Двигуни переводяться в режим керування швидкістю за допомогою функції wb_motor_set_position(..., INFINITY). Початкова швидкість встановлюється рівною нулю, що зупиняє обертання коліс:

```
left_motor = wb_robot_get_device("left wheel motor");
right_motor = wb_robot_get_device("right wheel motor");
wb_motor_set_position(left_motor, INFINITY);
wb_motor_set_position(right_motor, INFINITY);
wb_motor_set_velocity(left_motor, 0.0);
wb_motor_set_velocity(right_motor, 0.0);
```

Головний цикл управління

wb_robot_step(TIME_STEP) кожні 64 мс виконує один крок симуляції. Поки симуляція працює (!= -1), код у циклі виконується.

```
while (wb_robot_step(TIME_STEP) != -1) {
```

Основна логіка руху та уникнення перешкод

За замовчуванням встановлюється однакова швидкість для обох двигунів – робот рухається прямо.

```
double left_speed = 5.0;
double right_speed = 5.0;
```

Необхідно зчитати значення з двох передніх сенсорів:

- ds[0] – передній лівий сенсор (ps0);
- ds[7] – передній правий сенсор (ps7).

```
double front_left=wb_distance_sensor_get_value(ds[0]);
double front_right=wb_distance_sensor_get_value(ds[7]);
```

Якщо будь-який з фронтальних сенсорів виявив перешкоду (значення більше 100) – потрібна дія (поворот).

```
if (front_left > 100 || front_right > 100) {
```

Якщо ліворуч ближче перешкода, аніж праворуч (front_left > front_right), поворот здійснюється праворуч. Інакше – якщо з правого боку перешкода ближче – повертаємо ліворуч.

Необхідно встановити відповідні швидкості коліс: одне вперед, друге назад, щоб робот розвернувся.

```
if (front_left > front_right) {
  left_speed = 5.0;
  right_speed = -5.0;
} else {
  left_speed = -5.0;
  right_speed = 5.0;
}
```

Встановлення нової швидкості двигуна

Після обчислень треба оновити швидкість лівого і правого двигунів.

```
wb_motor_set_velocity(left_motor, left_speed);
wb_motor_set_velocity(right_motor, right_speed);
```

Завершення роботи контролера

- *wb_robot_cleanup()* – правильно завершує роботу контролера і очищає пам’ять.

return 0 – завершення програми.

```
wb_robot_cleanup();
return 0;
```

Для того, щоб скомпілювати код, необхідно вибрати *Build / Build* пункт меню. Помилки компіляції відображаються в консолі та виділяються червоним кольором.

Повний код контролера, наведено нижче:

```
#include <webots/robot.h>
#include <webots/distance_sensor.h>
#include <webots/motor.h>

#define TIME_STEP 64

int main() {
  // Ініціалізація Webots
  wb_robot_init();

  // Імена 8 інфрачервоних датчиків
  const char *ds_names[8] = {
 "ps0", "ps1", "ps2", "ps3", "ps4", "ps5", "ps6", "ps7"
  };

  // Оголошення масиву датчиків
  WbDeviceTag ds[8];

  // Ініціалізація кожного датчика
  for (int i = 0; i < 8; i++) {
```

```
      ds[i] = wb_robot_get_device(ds_names[i]);
      wb_distance_sensor_enable(ds[i], TIME_STEP);
    }
      // Ініціалізація двигунів
    WbDeviceTag left_motor = wb_robot_get_device("left
wheel motor");
    WbDeviceTag right_motor = wb_robot_get_device("right
wheel motor");
    wb_motor_set_position(left_motor, INFINITY);
    wb_motor_set_position(right_motor, INFINITY);
    wb_motor_set_velocity(left_motor, 0.0);
    wb_motor_set_velocity(right_motor, 0.0);

    // Основний цикл
    while (wb_robot_step(TIME_STEP) != -1) {
      double left_speed = 5.0;
      double right_speed = 5.0;

      // Читання значень передніх датчиків
      double front_left =
wb_distance_sensor_get_value(ds[0]);
      double front_right =
wb_distance_sensor_get_value(ds[7]);

     // Перевірка на наявність перешкод
        if (front_left > 100 || front_right > 100) {
          if (front_left > front_right) {
            // Більша перешкода зліва → поворот направо
            left_speed = 5.0;
            right_speed = -5.0;
          } else {
     // Більша перешкода справа → поворот наліво
            left_speed = -5.0;
            right_speed = 5.0;
          }
        }
    // Встановлення швидкостей моторів
        wb_motor_set_velocity(left_motor, left_speed);
        wb_motor_set_velocity(right_motor, right_speed);
      }
    // Завершення роботи
      wb_robot_cleanup();
      return 0;
    }
```

Точка входу контролера – це main функція.

Перед викликом будь-якої функції *Webots API* повинна бути викликана wb_robot_init.

Остання функція, яку потрібно викликати перед виходом з основної функції – це *wb_robot_cleanup*.

Посилання на пристрій здійснюється через поле *name* вузла пристрою. Посилання на вузол можна отримати завдяки функції *wb_robot_get_device*.

Кожна програма контролера виконується як дочірній процес процесу Webots. Процес контролера не поділяє пам'ять з Webots (окрім зображення з камери) і може працювати на іншому процесорі (або ядрі CPU), окремо від Webots.

Код контролера пов'язаний з динамічною бібліотекою *lib Controller*. Ця бібліотека керує обміном даних між контролером і Webots.

Варіанти завдань

№	Арена	Фон і Світло	Об'єкти		Робот	Контролер
			назва	к-ть		
1	BakelitePlastic	Dusk	Ball	3	Elisa3	Оминання перешкод
2	BrushedAluminium	Empty_office	PlasticCrate	2	E-puck	Оминання перешкод
3	CementTiles	Entrance_hall	CardboardBox	4	Elisa3	Рух по колу
4	Asphalt	Noon_building_overcast	PingPongBall	3	E-puck	Рух по колу
5	CorrodedMetal	Mars	Ball	2	Elisa3	Рух вздовж стіни
6	CorrugatedPlates	Noon_park_empty	PlasticCrate	4	E-puck	Рух вздовж стіни
7	Copper	Noon_cloudy_countryside	CardboardBox	3	Elisa3	Рух заднім ходом
8	DamascusSteel	Music_hall	WoodenBox	2	E-puck	Рух заднім ходом
9	ElectricConduit	Stadium	RobocupSoccerBall	4	Elisa3	Рух зигзагом
10	HammeredCopper	Stadium_dry	PingPongBall	2	E-puck	Рух зигзагом
11	Marble	Dusk	PingPongBall	3	Elisa3	Рух квадратною траєкторією
12	MetalPipePaint	Empty_office	Ball	2	E-puck	Рух квадратною траєкторією
13	MetalStainlessSteelCable	Entrance_hall	PlasticCrate	4	Elisa3	Рух траєкторією цифри 8
14	OldSteel	Noon_building_overcast	CardboardBox	3	E-puck	Рух траєкторією цифри 8
15	PorcelainChevronTiles	Mars	WoodenBox	2	Elisa3	Рух трикутною траєкторією
16	RoughConcrete	Noon_park_empty	PlasticCrate	4	E-puck	Оминання перешкод
17	RoughPine	Noon_cloudy_countryside	CardboardBox	3	Elisa3	Рух по колу заднім ходом
18	RustyMetal	Music_hall	WoodenBox	2	E-puck	Рух по колу заднім ходом
19	ScratchedPaint	Stadium	RobocupSoccerBall	4	Elisa3	Рух вздовж стіни заднім ходом

№	Арена	Фон і Світло	Об'єкти		Робот	Контролер
			назва	к-ть		
20	ThreadMetalPlate	Stadium_dry	PingPongBall	3	E-puck	Рух вздовж стіни заднім ходом
21	VarnishedPine	Dusk	CardboardBox	2	Elisa3	Рух вздовж стіни
22	WornBurlap	Empty_office	PingPongBall	3	E-puck	Рух вздовж стіни
23	BrushedSteel	Entrance_hall	Ball	4	Elisa3	Рух трикутною траєкторією
24	CorrugaredPvc	Noon_building_overcast	PlasticCrate	2	E-puck	Рух трикутною траєкторією
25	DryMud	Mars	WoodenBox	3	Elisa3	Рух траєкторією цифри 8
26	FlexibleAluminiumDuct	Noon_park_empty	RobocupSoccerBall	4	E-puck	Рух траєкторією цифри 8
27	FormedConcrete	Noon_cloudy_countryside	PingPongBall	2	Elisa3	Рух заднім ходом
28	GalvanizedMetal	Music_hall	PingPongBall	3	E-puck	Рух заднім ходом

Зміст звіту

1. Звіт з лабораторної роботи має бути виконаний на аркушах формату А4.
2. Звіт має містити: назву лабораторної роботи, її мету і короткі теоретичні відомості.
3. У розділі «Результати виконання лабораторної роботи» студент має представити результати покрокового виконання лабораторної роботи.
4. Написати висновки до даної лабораторної роботи.
5. Оформити звіт та подати його викладачу на захист.

Контрольні запитання

1. Яка мета використання API Webots для програмування контролерів?
2. Як налаштовується новий контролер у середовищі Webots?
3. Яку роль виконують вузли Distance Sensor у моделі робота?
4. Чому важливо ініціалізувати API Webots за допомогою функції wb_robot_init()?
5. Як задти швидкість обертання коліс робота в коді контролера?
6. Яка мета використання функції wb_robot_cleanup()?
7. Опишіть процес визначення перешкод за допомогою інфрачервоних сенсорів відстані.
8. Що таке макрос TIME_STEP, і чому він використовується в симуляції?
9. Як компілювати код контролера в середовищі Webots?
10. Які основні кроки потрібно виконати для створення простого контролера, який уникає перешкод?

Лабораторна робота № 3

РОЗРОБКА РОБОТА У WEBOTS

Мета роботи – ознайомитись з основами створення роботів в середовищі Webots. Створити робота у відповідності до завдання лабораторної роботи.

Дана лабораторна робота спрямована на створення робота в середовищі Webots. Робот складатиметься з корпусу, коліс і двох сенсорів відстані. 3D зображення створеного робота показано на рисунку 3.1.

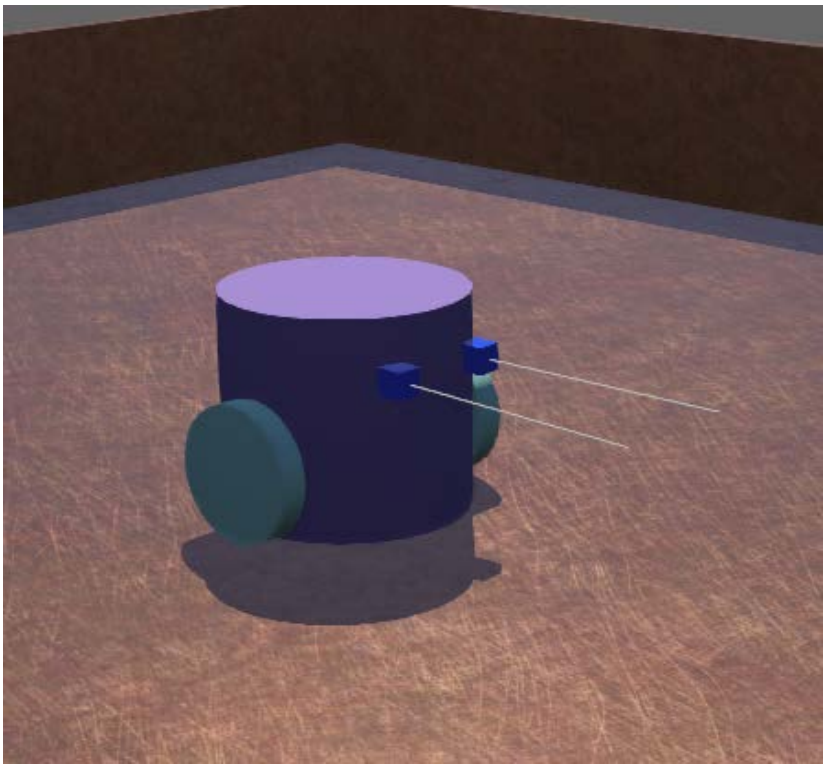

Рисунок 3.1 – 3D-зображення створеного робота

Хід роботи

1. Новий світ і новий контролер

Необхідно створити новий світ та зберегти його як robot.wbt. Створити платформу, по якій буде рухатися робот, стіни до платформи та освітлення.

2. Поділ робота на тверді вузли (Solid nodes)

Основні визначення та правила створення моделі робота:

Множина, що містить твердий вузол і всі його похідні вузли, називається твердими вузлами (*Solid nodes*). Аналогічне визначення застосову-

ється до вузлів типів *Device, Robot, Joint* і *Motor*. Для кращого взаємозв'язків, варто ознайомитися з ієрархією вузлів на відповідній діаграмі.

Більшість сенсорів і виконавчих механізмів одночасно є як твердими вузлами (*Solid*), так і пристроями (*Device*).

Основна структура моделі робота – це дерево твердих вузлів, пов'язаних між собою. Кореневим вузлом цього дерева завжди є вузол типу *Robot*.

Тверді тіла (*Solid*) з'єднуються між собою за допомогою вузлів з'єднань (*Joint nodes*). Вузол *Device* завжди має бути безпосереднім дочірнім вузлом для одного з таких вузлів: *Robot, Solid* або *Joint*.

Вузол-шарнір (*Joint*) використовується для додавання одного або двох ступенів свободи (*DOF*) між його батьківським і дочірнім вузлом. Прямі батьківські та дочірні вузли для *Joint* завжди є твердими вузлами (*Solid*).

Вузли, що наслідуються від *Joint*, дозволяють створювати різні типи обмежень між пов'язаними вузлами *Solid*. Найпоширеніший тип у робототехніці – шарнір (*Hinge Joint*), який дозволяє моделювати, наприклад, обертові двигуни, включаючи колеса.

Шарнір (*Joint*) можна контролювати або активувати, додавши у його поле *device* вузол типу *Position Sensor* або *Motor* відповідно.

3.Етапи побудови моделі робота:

• Визначити, які частини робота моделювати як твердий вузол (*Solid node*). У нашому прикладі це досить очевидно: робот має 2 ступеня свободи (DOF), що відповідають його колесам з двигунами. Його можна розбити на 3 основних твердих вузла: корпус та два колеса.

• Оптимізувати кількість DOF залежно від задачі моделювання. Якщо модель вимагає високої точності, наприклад для реалістичного моделювання колеса, можна використати 2 DOF. Якщо ж така точність не потрібна, можна застосувати спрощення – наприклад, змоделювати колесо як сферу (*Sphere*) з нульовим коефіцієнтом тертя.

• Вибрати вузол *Solid*, який буде кореневим вузлом моделі *Robot*. Цей вибір є довільним, але часто кращим буде використання вузла, що представляє корпус робота. Наприклад, у гуманоїдних роботах вузол *Robot* зазвичай відповідає тілу, бо симетрія значно спрощує розрахунок параметрів кінематичних пар.

Вибираємо корпус робота як вузол Робот (*Robot*).

На рисунках 3.2 та 3.3 представлено ієрархію вузлів робота, яка показує структуру робота в деталях.

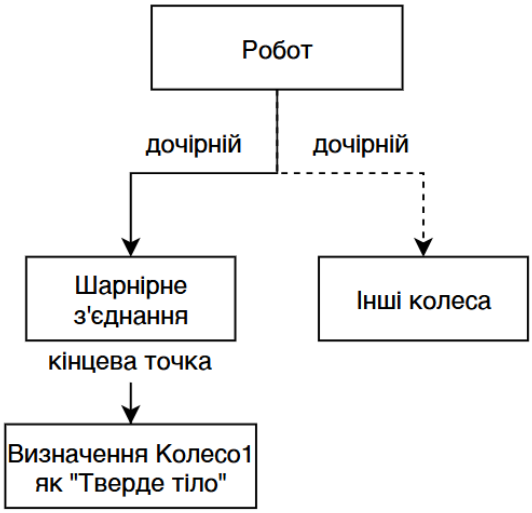

Рисунок 3.2 – Представлення на високому рівні колісного робота

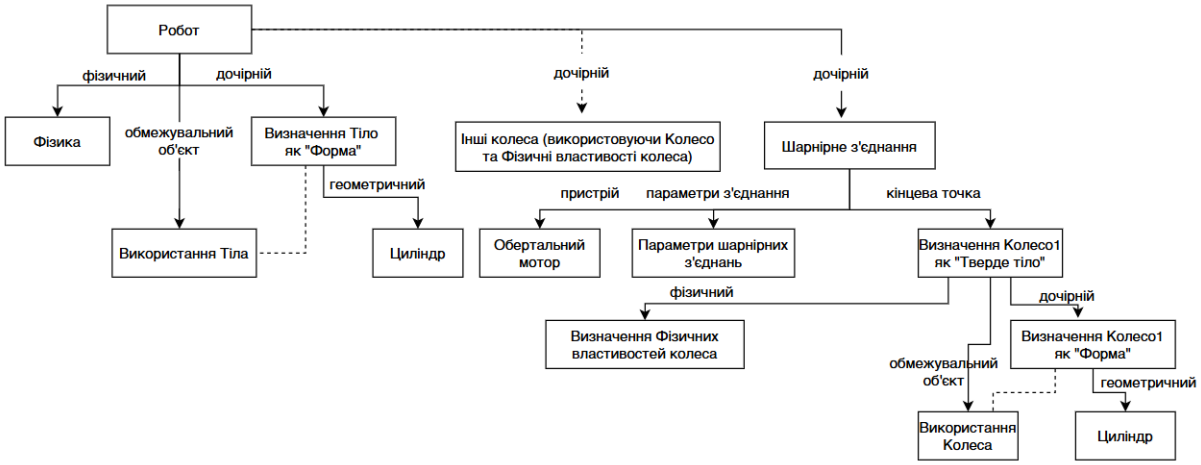

Рисунок 3.3 – Представлення на низькому рівні колісного робота

Для реалізації структури мобільного робота на основі Webots необхідно додати вузол «*Робот*» (*Robot*) наприкінці дерева сцени. Даний вузол є кореневим об’єктом, що репрезентує мобільного агента, та містить низку дочірніх компонентів.

У межах вузла «*Робот*» (*Robot*) створюються два «*Дочірні вузли*» (*children*) типу «*Шарнірне з’єднання*» (*Hinge Joint*), кожен з яких відповідає за обертальний ступінь свободи одного з коліс. Кінцевою точкою (*end Point*) кожного шарнірного з’єднання (*Hinge Joint*) є вузол типу «*Тверде тіло*» (*Solid*), що відображає фізичну конструкцію колеса. Таким чином забезпечується інтеграція кінематики обертання коліс у фізичну модель робота.

Для візуалізації та фізичного представлення тіла робота до вузла «*Робот*» (Robot) додається вузол «*Форма*» (Shape). Всередині нього оголошується геометричний підвузол типу «Циліндр» (Cylinder), що описує

циліндричну форму корпусу робота. Геометричні параметри циліндра встановлюються наступним чином:

– radius: 0.045;
– height: 0.08.

Для забезпечення візуальної ідентифікації тіла робота циліндр фарбують. Це здійснюється шляхом додавання поля *appearance* до вузла «*Форма*» (*Shape*), у якому задається значення кольору (наприклад, через RGB-палітру) Color { color 0.5 0.0 0.5 }.

Вузол «*Форма*» (*Shape*), що був доданий раніше, також використовується для опису поля «*Обмежувальний об'єкт*» (*bounding Object*) вузла «*Робот*» (*Robot*). Це необхідно для коректного визначення фізичних взаємодій об'єкта в середовищі Webots.

На завершальному етапі до вузла «*Робот*» (*Robot*) додається вузол «*Фізика*» (*Physics*), що забезпечує включення об'єкта до фізичної симуляції. Це дозволяє роботу реагувати на дію фізичних сил, взаємодіяти з іншими об'єктами з урахуванням законів механіки Ньютона.

Початкове положення колеса визначається полями трансляції та обертання вузла типу «*Базове тверде тіло*» (*Parent solid*). Точка з'єднання (*anchor*) і вісь обертання (*axis*) визначаються дочірнім елементом «*Параметри шарнірного з'єднання*» (*Hinge Joint Parameters*) вузла шарнірного з'єднання (*Hinge Joint*) (рис. 3.4).

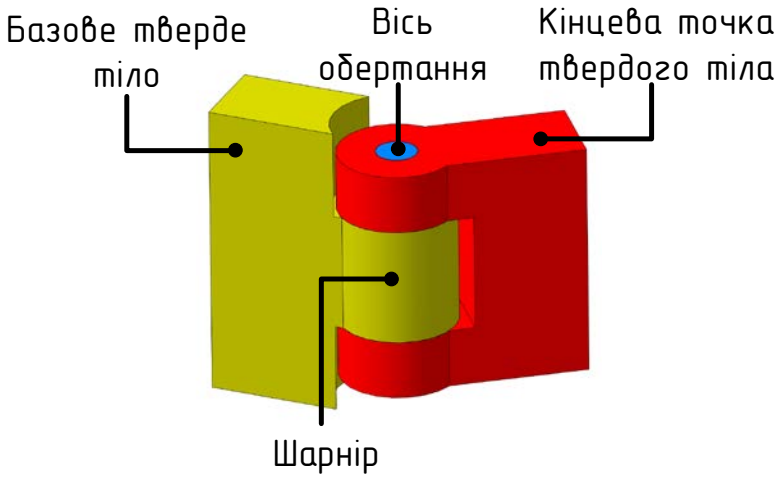

Рисунок 3.4 – Графічне представлення шарнірного з'єднання (*Hinge Joint*)

Для першого колеса просторове положення твердого об'єкта (*Solid translation*) має бути визначений до (0.045, 0.025, 0), щоб визначити відносний проміжок між корпусом і колесом, і обертання до (0 0 1 1.57), щоб циліндр колеса був правильно орієнтований. Шарнір (*Anchor*) у «*Параметри шарнірного з'єднання*» (*Hinge Joint Parameters*) також має бути визначений як (0.045, 0.025, 0), щоб визначити початок повороту (відносно тіла робота). Параметри шарнірного з'єднання (*Hinge Joint Parameters*) повинні визначати вісь обертання (вздовж осі *x*, (1, 0, 0)).

Для кожного з коліс мобільного робота необхідно створити повноцінну кінематичну пару «*Шарнірного з'єднання*» (*Hinge Joint*), яка складається з трьох основних частин:

Поле «*Параметри з'єднання*» (*joint Parameters*). До даного поля додається вузол «*Параметри шарнірного з'єднання*» (*Hinge Joint Parameters*), у якому встановлюються параметри з'єднання.

Основні параметри:

Шарнір (*anchor*): визначає точку з'єднання між тілом робота та колесом, наприклад «*Шарнір*» (*anchor*) -0.045 0.025 0 для лівого колеса. Для правого значення координати *x* буде зі знаком «+».

Вісь (*axis*): задає вісь обертання, наприклад *axis* 1 0 0 – обертання навколо осі *X*.

Поле «*Пристрій*» (*device*). До цього поля додається вузол «*Обертальний двигун*» (*Rotational Motor*), що дозволяє керувати обертанням відповідного колеса.

Необхідно задати параметр *name*, який буде ідентифікувати двигун в контролері.

Наприклад:

name «*wheel1*» – для лівого колеса,

name «*wheel2*» – для правого колеса.

Поле «*Кінцева точка*» (*end Point*). Це кінцева точка з'єднання – фактично колесо. Реалізується за допомогою вкладених вузлів:

«*Тверде тіло*» (*Solid*): задає фізичну суть колеса.

«*Дочірній*» (*children*) → «*Форма*» (*Shape*): визначає зовнішній вигляд.

«*Геометрія*» (*geometry*) → «*Циліндр*» (*Cylinder*): описує геометрію колеса (радіус 0.025, висота 0.01).

«*Вигляд*» (*appearance*) → «*Матеріал*» (*Material*): встановлює колір об'єкта; у даному випадку – блакитний (*diffuse Color* 0.0 0.5 1.0).

Також до вузла «*Тверде тіло*» (*Solid*) додається «*Обмежувальний об'єкт*» (*bounding Object*), що містить такий самий «*Циліндр*» (*Cylinder*), і вузол «*Фізика*» (*Physics*), який надає фізичні властивості (маса, інерція тощо).

4) Сенсори

Сенсори, використані у цій лабораторній роботі, відрізняються від тих, які були у попередній. Інтервали вимірювання сенсорів в межах від 0 см, що відповідає – 0, до 10 см, що відповідає – 1000 відносних одиниць.

Фінальна частина моделювання робота полягає в додаванні до робота двох сенсорів відстані. Додається два вузли *Distance Sensor* як прямі дочірні елементи вузла *Robot*. Необхідно звернути увагу, що сенсор відстані отримує дані вздовж додатної осі *x*. Отже, необхідно повернути сенсор відстані, щоб вказати їх вісь *x* поза роботом (див. рис. 3.5).

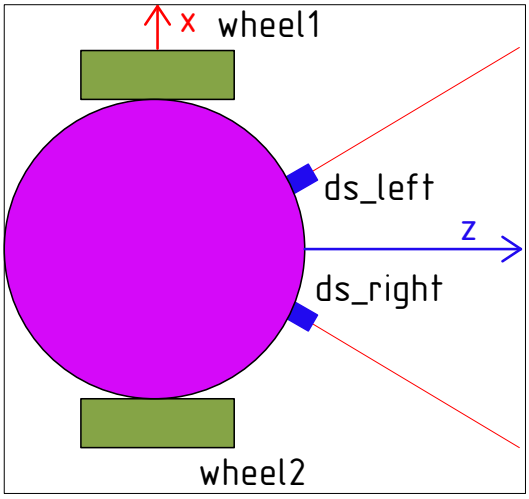

Рисунок 3.5 – Графічне представлення розміщення сенсорів вздовж осі *x*

Налаштування сенсора відстані можна переглядати за допомогою комбінації клавіш *ctrl-F10* або *View / Optional Rendering / Show Distance Sensor Rays*.

До моделі робота додаються два вузли типу *Distance Sensor*. Їх необхідно правильно розмістити, задати орієнтацію, ім'я, фізичні характеристики та графічне представлення.

Основні параметри:

Кут нахилу: кожен сенсор розміщується під кутом 0,3 рад. відносно напрямку руху робота (вектора вперед).

Форма сенсорів: геометрично сенсор моделюється у вигляді неперетвореного куба з довжиною ребра 0,01 м.

Колір: встановлюється синій колір (diffuse Color 0.0 0.0 1.0).

Ідентифікація: поле *name* визначається згідно з мітками на рисунку 3.2, наприклад «*ds_left*» та «*ds_right*».

У вузлі *Distance Sensor* поле обертання має чотири параметри. Якщо для параметра кута встановлено значення 0, то можна використовувати налаштування колеса для збільшення/зменшення на 0,1309 рад (що рівне 7,5 градусам).

5) Контролер

У попередніх лабораторних роботах розглянуто налаштування зворотного зв'язку та зчитування значення сенсора відстані.

Для програмування двигунів першим кроком є під'єднання модуля API, відповідного вузлу Rotational Motor:

```
#include <webots/motor.h>
```

Потім, щоб отримати посилання на вузли RotationalMotor:

```
// Ініціалізація моторів (wheel1 – лівий, wheel2 – правий)
WbDeviceTag left_motor = wb_robot_get_device("wheel1");
WbDeviceTag right_motor = wb_robot_get_device("wheel2");
```

Двигун (*Motor*) можна привести в дію, встановивши його положення, швидкість, прискорення або силу. В даному завданні необхідно встановити його швидкості. Для цього необхідно встановити його значення на нескінченність (*INFINITY*) і задавши його швидкість:

```
double speed = -1.5; // [rad/s]
wb_motor_set_position(wheels[0], INFINITY);
wb_motor_set_velocity(wheels[0], speed);
```

Для забезпечення автономного руху мобільного робота та реалізації функції огинання перешкод, необхідно застосувати контролер з назвою *two_wheeled_collision_avoidance*. Даний контролер керує рухом робота, використовуючи значення, отримані з сенсорів відстані вузол (*Distance Sensor*).

Контролер аналізує значення, що надходять з сенсорів відстані, та приймає відповідні рішення щодо зміни напрямку обертання коліс вузол (*Rotational Motor*) у випадку наближення до перешкоди. Таким чином забезпечується динамічне запобігання зіткненню з перешкодою у середовищі симуляції.

Технічні аспекти. Поле *lookup Table* вузлів *Distance Sens*or визначає відповідність між аналоговим сигналом та реальною відстанню до об’єкта.

Для полегшення процесу налагодження роботи сенсорів рекомендовано використовувати інтерфейс Webots. Для цього необхідно двічі натиснути клавішу курсору на корпусу робота, у лівій панелі відкриється меню з графіками активності сенсорів (*Distance Sensor*) та виконавчих пристроїв (*Rotational Motor*). Запуск симуляції дозволить показати зміну поведінки робота в режимі реального часу.

Інтеграція контролера. Необхідно призначити контролер *two_wheeled_collision_avoidance* у полі *controller* вузла *Robot*. Це забезпечить його активацію при запуску симуляції, та дозволить програмному забезпеченню взаємодіяти з віртуальними пристроями, інтегрованими в модель.

Як зазвичай, можливе рішення цієї вправи знаходиться в каталозі туторіалів до webots.

Код контролера, розроблений вище:

```
#include <webots/distance_sensor.h>
#include <webots/motor.h>
#include <webots/robot.h>
#define TIME_STEP 64

int main(int argc, char **argv) {
  wb_robot_init();

  int i;
  int avoid_obstacle_counter = 0;
```

```
// Ініціалізація сенсорів (лівий і правий)
     WbDeviceTag ds[2];
     const char *ds_names[2] = {"ds_left", "ds_right"};
     for (i = 0; i < 2; i++) {
       ds[i] = wb_robot_get_device(ds_names[i]);
       wb_distance_sensor_enable(ds[i], TIME_STEP);
     }

// Ініціалізація моторів (wheel1 - лівий, wheel2 - правий)
  WbDeviceTag left_motor = wb_robot_get_device("wheel1");
  WbDeviceTag right_motor = wb_robot_get_device("wheel2");

     wb_motor_set_position(left_motor, INFINITY);
     wb_motor_set_position(right_motor, INFINITY);

     while (wb_robot_step(TIME_STEP) != -1) {
       double left_speed = 1.0;
       double right_speed = 1.0;
      if (avoid_obstacle_counter > 0) {
        avoid_obstacle_counter--;
        left_speed = 1.0;
        right_speed = -1.0;
  // Поворот на місці для уникнення перешкоди
      } else {
  // Читання значень сенсорів
        double ds_values[2];
        for (i = 0; i < 2; i++) {
       ds_values[i] =
wb_distance_sensor_get_value(ds[i]);
        }

        // Перевірка наявності перешкоди
        if (ds_values[0] < 950.0 || ds_values[1] < 950.0)
{
          avoid_obstacle_counter = 100;
        }
      }

      wb_motor_set_velocity(left_motor, left_speed);
      wb_motor_set_velocity(right_motor, right_speed);
    }
      wb_robot_cleanup();
      return 0;  // EXIT_SUCCESS
    }
```

Варіанти завдань

№	К-ть коліс	Колір корпусу	Форма корпусу	Колір коліс	Контролер руху
1	2	Синій	Циліндр	Чорний	Простий прямий рух
2	4	Червоний	Прямокутник	Сірий	Оминання перешкод
3	2	Зелений	Циліндр	Чорний	Рух задом
4	4	Помаранчевий	Прямокутник	Жовтий	Оминання перешкод заднім ходом
5	2	Чорний	Циліндр	Червоний	Рух по колу
6	4	Білий	Прямокутник	Чорний	Рух вздовж стіни
7	2	Жовтий	Циліндр	Синій	Рух квадратною траєкторією
8	4	Сірий	Прямокутник	Білий	Рух траєкторією цифри 8
9	2	Блакитний	Циліндр	Чорний	Рух трикутною траєкторією
10	4	Зелений	Прямокутник	Синій	Рух вздовж стіни заднім ходом
11	2	Червоний	Циліндр	Сірий	Простий прямий рух
12	4	Чорний	Прямокутник	Помаранчевий	Оминання перешкод
13	2	Білий	Циліндр	Блакитний	Рух заднім ходом
14	4	Помаранчевий	Прямокутник	Чорний	Оминання перешкод заднім ходом
15	2	Сірий	Циліндр	Чорний	Рух по колу
16	4	Зелений	Прямокутник	Сірий	Рух вздовж стіни
17	2	Червоний	Циліндр	Чорний	Рух квадратною траєкторією
18	4	Жовтий	Прямокутник	Чорний	Рух траєкторією цифри 8
19	2	Синій	Циліндр	Сірий	Рух трикутною траєкторією
20	4	Сірий	Прямокутник	Червоний	Рух вздовж стіни заднім ходом
21	2	Помаранчевий	Циліндр	Чорний	Простий прямий рух
22	4	Чорний	Прямокутник	Синій	Оминання перешкод
23	2	Білий	Циліндр	Чорний	Рух заднім ходом
24	4	Сірий	Прямокутник	Блакитний	Оминання перешкод заднім ходом
25	2	Зелений	Циліндр	Чорний	Рух по колу
26	4	Червоний	Прямокутник	Помаранчевий	Рух вздовж стіни
27	2	Чорний	Циліндр	Сірий	Рух квадратною траєкторією
28	4	Білий	Прямокутник	Чорний	Рух траєкторією цифри 8

Зміст звіту

1. Звіт з лабораторної роботи має бути виконаний на аркушах формату А4.
2. Звіт повинен містити: назву лабораторної роботи, її мету і короткі теоретичні відомості.
3. У розділі «Результати виконання лабораторної роботи» студент має представити результати покрокового виконання лабораторної роботи.
4. Написати висновки до даної лабораторної роботи.
5. Оформити звіт та подати його викладачу на захист.

Контрольні запитання

1. Які основні вузли використовуються для створення мобільного робота у Webots?
2. Що таке вузол *Robot* і які основні поля він містить?
3. Для чого використовується вузол *Hinge Joint* у моделі робота?
4. Яке призначення вузлів *Hinge Joint* та *Hinge Joint Parameters* у побудові робота?
5. Для чого використовується вузол Rotational Motor?
6. Яка роль вузла Solid у моделюванні робота?
7. Що таке поле *bounding Object* і чому воно є обов’язковим для тіл, що беруть участь у фізичній симуляції?
8. Які властивості визначають геометрію вузла *Cylinder*?
9. Для чого потрібен вузол *Distance Sensor* і як здійснюється його налаштування?
10. У чому полягає принцип роботи алгоритму огинання перешкод, реалізованого в контролері *two_wheeled_collision_avoidance*?

ПЕРЕЛІК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

Основна

1.Барабан С. В. Комп'ютерна інженерія та основи робототехніки : електронний навчальний посібник комбінованого (локального та мережного) використання [Електронний ресурс] / С. В. Барабан, Р. С. Белзецький, І. Р. Арсенюк. – Вінниця : ВНТУ, 2024. – 155 с. – Режим доступу: https://pdf.lib.vntu.edu.ua/books/2024/Baraban_2024_155.pdf (дата звернення 8.05.2025).

2. Белзецький Р. С. Комп'ютерна інженерія та основи робототехніки : електронний лабораторний практикум комбінованого (локального та мережного) використання [Електронний ресурс] / Р. С. Белзецький, Я. В. Іванчук. – Вінниця : ВНТУ, 2024. – 78 с. – Режим доступу: https://pdf.lib.vntu.edu.ua/books/2024/Belzetskii_2024_78.pdf (дата звернення 3.05.2025).

3. Белзецький Р. С. Робоча програма навчальної дисципліни «Комп'ютерна інженерія та основи робототехніки». Рівень вищої освіти перший (бакалаврський). Спеціальність 122 «Комп'ютерні науки» [Електронний ресурс] / Р. С. Белзецький. – Вінниця : ВНТУ, 2024. – 15 с. – Режим доступу: https://iq.vntu.edu.ua/method/getfile.php?fname=161144.pdf&x=1&card_id=48543&id=161144) (дата звернення 12.05.2025).

4. Ковальов Ю. А. Проектування промислових роботів та маніпуляторів / С. О. Кошель, Ю. А. Ковальов, О. П. Манойленко. – Київ : Центр навчальної літератури, 2023. – 256 с.

5. Комп'ютерна схемотехніка : підручник / О. Д. Азаров, В. А. Гарнага, Я. М. Клятченко, В. П. Тарасенко ; ВНТУ, НТУ України «КПІ ім. І. Сікорського». – Вінниця : ВНТУ, 2018. – 230 с.

6. Комп'ютерна схемотехніка та логіка : [навчальний посібник] / В. В. Лапко, Б. С. Гусєв, Д. Ю. Касаткін [та ін.]. – Київ : НУБіП України, 2017. – 291 с.

7. Комп'ютерна техніка : електронний навчальний посібник комбінованого використання / Н. О. Біліченко, Д. О. Галущак, В. Л. Крещенецький, С. В. Цимбал. – Вінниця : ВНТУ, 2021.

8. Цвіркун Л. І. Робототехніка та мехатроніка : навчальний посібник / Л. І. Цвіркун, Г. Грулер ; під заг. ред. Л. І. Цвіркун. – Вид. третє, перероб. і допов. – Дніпро : НГУ, 2017.

Додаткова

1. Белзецький Р. С. Чотирьохосьова рука-маніпулятор з тактильним зворотнім зв'язком [Електронний ресурс] / Р. С. Белзецький // Матеріали XLVIII Науково-технічної конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ–2019) (Вінниця, 13-

15 берез. 2019 р.) : зб. доп. – Вінниця : ВНТУ, 2019. – С. 561-564. – Режим доступу: https://conferences.vntu.edu.ua/public/files/1/ininv_2019_netpub.pdf (дата звернення 28.04.2025).

2. Белзецький Р. С. Чотирьохколісний мобільний робот на базі плати mCore [Електронний ресурс] / Р. С. Белзецький, Т. С. Мельник, А. П. Вітковська // Матеріали LIII науково-технічної конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ–2024) (Вінниця, 20-22 черв. 2024 р.) : зб. доп. – Вінниця : ВНТУ, 2024. – С. 991-994. – Режим доступу: https://press.vntu.edu.ua/index.php/vntu/catalog/view/832/1453/2726-1 (дата звернення 28.04.2025).

4. Освітня робототехніка : зб. наук. пр. за матеріалами II Всеукраїнської науково-практичної конференції «Освітня робототехніка» (Дніпро, 14 квіт. 2022 р.). – Дніпро, 2022. – 162 с.

5. Основи робототехніки : конспект лекцій для студентів першого (бакалаврського) рівня вищої освіти денної та заочної форми навч., спец. : 141 «Електроенергетика, електротехніка та електромеханіка» / Державний біотехнологічний університет ; упоряд. М. С. Сорокін. – Харків : [б. в.], 2024. – 94 с.

6. Belzetskyi R. S. Methods for solving the direct kinematics problem for determining the grip position of the robot-manipulator in space [Електронний ресурс] / R. S. Belzetskyi, M. O. Chernilevskyi, K. I. Yanchuk // Матеріали LIII науково-технічної конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ–2024) (Вінниця, 20-22 черв. 2024 р.) : зб. доп. – Вінниця : ВНТУ, 2024. – С. 989-990 – Режим доступу: URL: https://press.vntu.edu.ua/index.php/vntu/catalog/view/832/1453/2726-1 (дата звернення 28.04.2025).

7. Internet of Things : Novel Advances and Envisioned Applications / eds.: D. P. Acharya; M. K. Geetha. Springer, 2017. – P. 311.

Інформаційні ресурси

1. Національна бібліотека України імені В. І. Вернадського [Електронний ресурс] : [Вебсайт]. – Електронні дані. – Режим доступу: http://nbuv.gov.ua/ (дата звернення 1.05.2025).

2. Робототехніка [Електронний ресурс] : [Вебсайт]. – Електронні дані. – Режим доступу: https://sites.google.com/view/robot0technika/%D0%B3%D0%BE%D0%BB%D0%BE%D0%B2%D0%BD%D0%B0-%D1%81%D1%82%D0%BE%D1%80%D1%96%D0%BD%D0%BA%D0%B0 (дата звернення 1.05.2025).

Додаток А
Зразок титульної сторінки лабораторної роботи

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра КН

ЛАБОРАТОРНА РОБОТА № 1
з дисципліни
«Комп'ютерна інженерія та основи робототехніки»
на тему:
«Моделювання у Webots»

Виконала: ст. гр. 1КН-22б
Добровольська Є. Р.

Перевірив: к.т.н., доцент
Белзецький Р. С.

Вінниця
2025

Електронне навчальне видання

Руслан Станіславович Белзецький

Методичні вказівки до виконання лабораторних робіт з дисципліни «Комп'ютерна інженерія та основи робототехніки» зі спеціальності «Комп'ютерні науки»

Рукопис оформив *Р. Белзецький*

Редактор *Н. Слободянюк*

Оригінал-макет виготовлено в *РВВ ВНТУ*

Підписано до видання 26.09.2025 р.
Гарнітура Times New Roman.
Зам. № Р2025-136.

Видавець та виготовлювач
Вінницький національний технічний університет
редакційно-видавничий відділ.
ВНТУ, ГНК, к. 114.
Хмельницьке шосе, 95,
м. Вінниця, 21021.
press.vntu.edu.ua;
E-mail: rvv.vntu@gmail.com.
Свідоцтво суб'єкта видавничої справи
серія ДК № 3516 від 01.07.2009 р.