Міністерство освіти і науки України

ДЕРЖАВНИЙ БІОТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет енергетики, робототехніки та комп'ютерних технологій

Кафедра електромеханіки, робототехніки, біомедичної інженерії та електротехніки

ПРОГРАМУВАННЯ МОБІЛЬНИХ РОБОТІВ.

Методичні вказівки із дисципліни «Основи робототехніки»

Для студентів першого (бакалаврського) рівня вищої освіти денної та заочної форми навчання, спеціальностей:
141 Електроенергетика, електротехніка та електромеханіка

Харків
2024

УДК 621.865.8
С69

Схвалено
на засіданні кафедри
електромеханіки, робототехніки, біомедичної інженерії та електротехніки
Протокол № 2 від 26.09.2024

Програмування мобільних роботів: методичні вказівки 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. – 32 с.

Методичні вказівки з дисципліни “Програмування мобільних роботів” складені у відповідності до навчально плану.

Видання призначене для студентів першого (бакалаврського) рівня вищої освіти денної та заочної форми навчання, спеціальностей: 141 Електроенергетика, електротехніка та електромеханіка,

Рецензенти:

К.В. Демченко, канд техн. наук, доцент, завідувачка кафедри автоматизації та комп’ютерно-інтегрованих технологій, Державного біотехнологічного університету;

Відповідальний за випуск : М.С. Сорокін, канд. техн. наук, доц.

© Сорокін М.С упорядкування, 2024
© ДБТУ, 2024

ЗМІСТ

Вступ

Webots – це тривимірний симулятор мобільних роботів з відкритим вихідним кодом. Спочатку він був розроблений як дослідницький інструмент для дослідження різних алгоритмів управління в мобільній робототехніці. З грудня 2018 року Webots випускається як програмне забезпечення з відкритим вихідним кодом під ліцензією Apache 2.0. Будь ласка, зверніть увагу, що ви повинні мати мінімальні знання з мобільної робототехніки, програмування на C, C++, Java, Python або MATLAB, а також на VRML97 (Virtual Reality Modeling Language).

Webots — це професійний пакет програмного забезпечення для моделювання мобільних роботів. Він пропонує середовище швидкого моделювання, яке дозволяє користувачеві створювати 3D-віртуальні світи з такими фізичними властивостями, як маса, шарнірні з'єднання, коефіцієнти тертя тощо. Користувач може додавати прості пасивні об'єкти або активні об'єкти, які називаються мобільними роботами. Ці роботи можуть мати різні схеми пересування (колісні роботи, роботи на ногах або літаючі роботи). Крім того, вони можуть бути оснащені рядом датчиків і виконавчих пристроїв, таких як датчики відстані, моторколеса, камери, двигуни, сенсорні датчики, випромінювачі, приймачі тощо. Нарешті, користувач може запрограмувати кожного робота окремо на демонстрацію бажаної поведінки. Webots містить велику кількість моделей роботів і прикладів програм-контролерів, щоб допомогти користувачам розпочати роботу.

Webots також містить ряд інтерфейсів до реальних мобільних роботів, так що, як тільки ваш змодельований робот поведе себе так, як очікувалося, ви можете перенести його програму управління на реального робота, такого як e-puck, DARwIn-OP, Nao тощо. Додавання нових інтерфейсів можливе через відповідну систему.

Системні вимоги

Для запуску Webots потрібне наступне обладнання:

1. Досить свіжий ПК або комп'ютер Mac з тактовою частотою двоядерного процесора не менше 2 ГГц і 2 ГБ оперативної пам'яті є мінімальною вимогою. Однак рекомендується використовувати чотирьохядерний процесор.

1. Потрібен графічний адаптер із підтримкою NVIDIA або AMD OpenGL (мінімальна версія 3.3) із принаймні 512 МБ оперативної пам'яті. Ми не рекомендуємо використовувати будь-які інші графічні адаптери, включаючи графічні адаптери Intel, оскільки вони часто не підтримують OpenGL, що може спричинити проблеми з 3D-рендерингом та збої програм. Тим не

менш, у деяких випадках встановлення найновішого графічного драйвера Intel може виправити такі проблеми та дозволити використовувати Webots. Однак немає жодних гарантій щодо цього. Для систем Linux ми рекомендуємо лише відеокарти NVIDIA. Webots добре працює на всіх відеокартах, що входять до складу досить свіжих комп'ютерів Apple.

Підтримуються такі операційні системи:

1. Linux: Webots гарантовано працює в останніх випусках Ubuntu з довготривалою підтримкою (LTS), включаючи версії 22.04 і 20.04. Але також відомо, що він працює на більшості основних дистрибутивів Linux, включаючи RedHat, Mandrake, Debian, Gentoo, Arch, SuSE та Slackware. Ми рекомендуємо використовувати останню версію Linux. Webots доступний лише для систем Linux 64 (x86-64). Webots не працює на Ubuntu версій до 20.04.

2. Windows: Webots працює на Windows 11 і Windows 10 (лише 64-розрядні версії).

3. Mac: Webots працює на macOS 11 «Big Sur», macOS 12 «Monterey» і macOS 13 «Ventura».

Webots може працювати, але офіційно не підтримується на більш ранніх версіях вищезгаданих операційних систем.

Інші версії Webots для інших систем UNIX можуть бути доступні за запитом.

Зазвичай для встановлення Webots потрібно мати права системного адміністратора. Після встановлення Webots може використовуватися звичайним користувачем.

Початок роботи з Webots

Webots добре підходить для дослідницьких та освітніх проектів, пов'язаних з мобільною робототехнікою. Багато проєктів мобільної робототехніки роками покладалися на Webots у таких сферах:

1. Створення прототипів мобільних роботів (академічні дослідження, автомобільна промисловість, аеронавтика, індустрія пилососів, індустрія іграшок, любителі тощо)
2. Дослідження пересування роботів (ногі, гуманоїди, чотириногі роботи тощо)
3. Мультискладові дослідження (ройовий інтелект, колаборативні групи мобільних роботів тощо)
4. Дослідження адаптивної поведінки (генетичний алгоритм, нейронні мережі, ШІ тощо).

5. Викладання робототехніки (лекції з робототехніки, лекції з програмування на C/C++/Java/Python тощо)
6. Змагання роботів (наприклад, Robotstadium або Rat's Life)

Вам знадобиться мінімальна кількість технічних знань для розробки власних симуляцій:

1. Базові знання мови програмування C, C++, Java, Python або MATLAB необхідні для програмування власних контролерів роботів. Однак, навіть якщо ви не знаєте цих мов, ви все одно можете запрограмувати роботів e-puck і Hemisson за допомогою простої графічної мови програмування під назвою BotStudio.
2. Якщо ви не хочете використовувати існуючі моделі роботів, надані в Webots, і хотіли б створювати власні моделі роботів або додавати спеціальні об'єкти в змодельованих середовищах, вам знадобляться базові знання 3D комп'ютерної графіки та мови опису VRML97. Це дозволить вам створювати 3D-моделі у Webots або імпортувати їх із програмного забезпечення для 3D-моделювання.

Симуляція Webots складається з наступних елементів:

1. Файл Світу *Webots* (.wbt), який визначає одного або кількох роботів та їхнє середовище. Файл *.wbt іноді залежить від зовнішніх файлів PROTO (.proto) і текстур.
2. Одна або декілька програм-контролерів для вищевказаних роботів (на C/C++/Java/Python/MATLAB).
3. Необов'язковий фізичний додаток, яким можна скористатися для зміни звичайної фізичної поведінки Webots (у C/C++).

Віртуальне середовище (Світ, World).

Світ у Webots — це 3D-опис властивостей роботів та їхнього оточення. Він містить опис кожного об'єкта: положення, орієнтацію, геометрію, зовнішній вигляд (наприклад, колір або яскравість), фізичні властивості, тип об'єкта тощо. Світи організовані як ієрархічні структури, де об'єкти можуть містити інші об'єкти (як у VRML97). Наприклад, робот може містити два колеса, датчик відстані та шарнір, який сам містить камеру тощо. Файл світу не містить коду контролера роботів; У ньому вказується лише ім'я контролера, яке необхідне для кожного робота. Світи зберігаються у файлах ".wbt". Файли ".wbt" зберігаються у підкаталозі "worlds" кожного проекту Webots.

Що таке контролер?

Контролер - це комп'ютерна програма, яка керує роботом, заданим у файлі світу. Контролери можуть бути написані на будь-якій з мов програмування, що підтримуються Webots: C, C++, Java, Python або MATLAB. Коли

починається симуляція, Webots запускає вказані контролери, кожен як окремий процес, і пов'язує процеси контролера з змодельованими роботами. Зверніть увагу, що кілька роботів можуть використовувати один і той же код контролера, однак для кожного робота буде запущено окремий процес.

Деякі мови програмування потрібно компілювати (C і C++), інші мови потрібно інтерпретувати (Python і MATLAB), а деякі потрібно як компілювати, так і інтерпретувати (Java). Наприклад, контролери C і C++ компілюються в залежні від платформи двійкові виконувані файли (наприклад, ".exe" у Windows). Контролери Python та MATLAB інтерпретуються відповідними системами виконання (які мають бути встановлені). Контролер Java повинен бути скомпільований у байт-код (".class" файли або ".jar"), а потім інтерпретований віртуальною машиною Java.

Вихідні файли та двійкові файли кожного контролера зберігаються разом у каталозі контролера. Каталог контролерів розміщується у підкаталозі "controllers" кожного проекту Webots.

Що таке контролер-супервізор (supervisor)?

Контролер супервізора — це контролер робота, поле супервізора якого встановлено на TRUE, він може виконувати операції, які зазвичай може виконувати лише людина-оператор, а не реальний робот. Контролер Supervisor може бути написаний на будь-якій з перерахованих вище мов програмування. Однак, на відміну від звичайного контролера робота, контролер Supervisor матиме доступ до привілейованих операцій. До привілейованих операцій можна віднести імітаційне управління, наприклад, переміщення роботів у випадкове положення, відеофіксацію симуляції тощо.

Практикум № 1. Перший проект Webots

Створення нового оточення(новий Світ).

Світ (**World**) — це файл, що містить інформацію про те, де знаходяться об'єкти, як вони виглядають, як вони взаємодіють один з одним, який колір неба, а також визначення гравітації, тертя, маси об'єктів тощо. Він визначає початковий стан симуляції. Різні об'єкти називаються **вузлами (Nodes)** і організовані ієрархічно в **дерево сцени (Scene Tree)**. Тому вузол може містити підвузли. Світ зберігається у файлі з розширенням **.wbt**. Формат файлу походить від мови **VRML97** і є зручним для читання людиною. Файли світу повинні зберігатися безпосередньо в каталозі під назвою **worlds**.

Завдання 1. Призупиніть поточну симуляцію, натиснувши кнопку **Pause**  на 3D-панелі. Імітацію буде призупинено, якщо віртуальний лічильник часу на головній панелі інструментів зупинено. Створіть новий проект з каталогу **File / New / New Project Directory** пункту меню і дотримуйтесь інструкцій:

1. Назвіть каталог проекту **my_first_simulation** замість запропонованого **my_project**.
2. Назвіть файл світу **my_first_simulation.wbt** замість запропонованого **empty.wbt**.
3. Поставте всі прапорці, включно з пунктом «Додати прямокутну арену», який не встановлено за замовчуванням.

Webots відображає список каталогів і файлів, які він щойно створив. Це відповідає стандартній ієрархії файлів проекту Webots. Натисніть кнопку «Готово» (Windows, Linux) або «Готово» (macOS), щоб закрити це вікно.

Вітаємо, ви щойно створили свій перший світ Webots! На 3D-зображенні має відображатися квадратна арена з підлогою у вигляді шахівниці. Ви можете переміщати точку огляду в 3D-режимі за допомогою миші: ліву кнопку, праву кнопку та коліщатко.

Вузли **(Nodes)** Webots, що зберігаються у файлах світу, організовані в деревоподібну структуру, яка називається **деревом сцени**. Дерево сцени можна переглядати у двох підвікнах головного вікна: 3D огляд (у центрі головного вікна) є 3D-представленням дерева сцени, а вигляд дерева сцени (ліворуч) є ієрархічним представленням дерева сцени. Перегляд дерева сцени – це місце, де можна змінювати вузли та поля. В даний час в ньому повинні бути перераховані наступні вузли (рис. 1):

1. WorldInfo: містить глобальні параметри моделювання.
2. Viewpoint: визначає основні параметри камери огляду.
3. TexturedBackground: визначає фон сцени (ви повинні бачити гори вдалині, якщо трохи обернете точку огляду).

4. TexturedBackgroundLight: визначає світло, пов'язане з вищевказаним фоном.
5. RectangleArena: визначте єдиний об'єкт, який ви бачите в цій сцені.

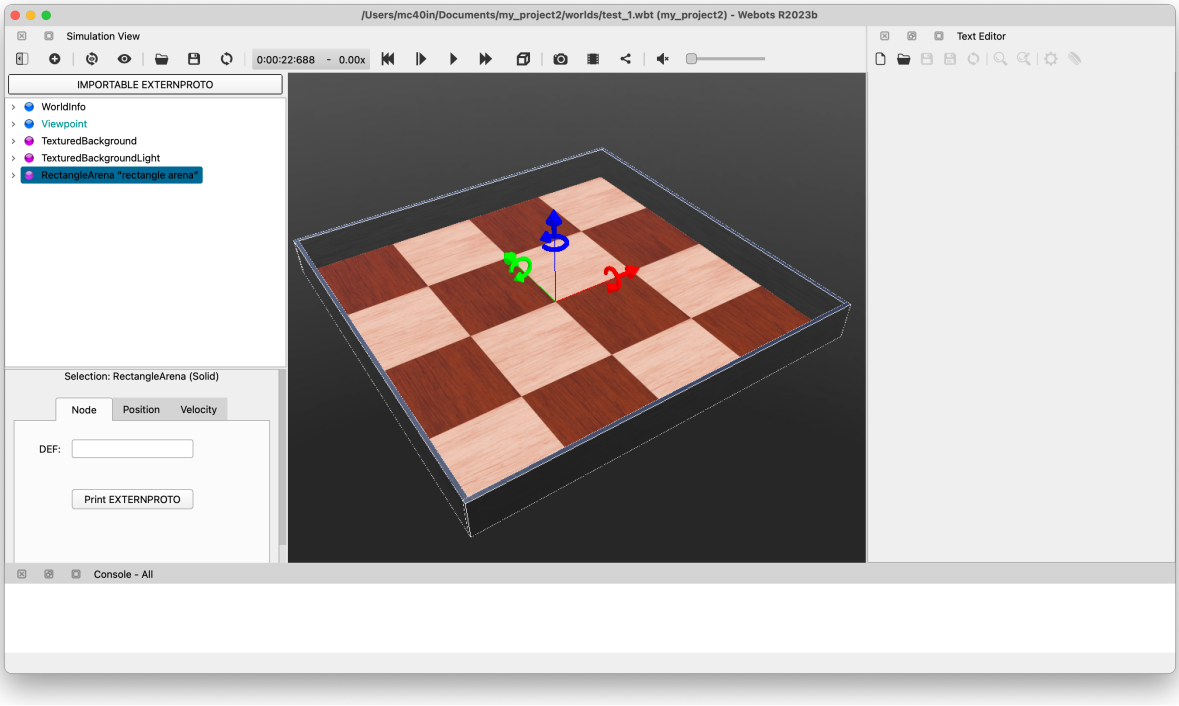


Рисунок 1. – Екран нового проекту Webots.

Завдання №2: Двічі клацніть вузол RectangleArena у дереві сцени. Це повинно відкрити вузол і відобразити його поля.

1. Виберіть поле **floorTileSize** і встановіть його значення **0.25 0.25** замість **0.5 0.5**. Ви повинні побачити ефект відразу в 3D-вигляді.
2. Виберіть поле **wallHeight** і змініть його значення на **0,05** замість **0,1**. Стіна арени тепер повинна бути нижче**.**

У перегляді дерева сцен поля відображаються іншим кольором (залежно від теми), якщо вони відрізняються від значень за замовчуванням. Зараз пропоную вам зупинитися і провести декілька практичних завдань для більш детального ознайомлення із налаштуваннями та властивостями вашого першого віртуального середовища.

Завдання №3: Двічі клацніть на **RectangleArena** в дереві сцени, щоб закрити її та виділити. Натисніть кнопку Додати  у верхній частині дерева сцени. У діалоговому вікні, що відкриється, виберіть **PROTO nodes (Webots Projects) / objects / factory / containers / WoodenBox (Solid).** Посеред арени має з'явитися велика коробка. Двічі клацніть на ньому в дереві сцени, щоб відкрити його поля.

1. Змініть його **size** на **0,1, 0,1, 0,1** замість **0,6, 0,6, 0,6**.
2. Змініть його **translation** на **0 0 0,05** замість **0 0 0,3**. Крім того, ви можете скористатися синьою стрілкою, яка з'являється на 3D-екрані, щоб налаштувати поле **translation.z**.
3. Тепер клацніть, утримуючи клавішу **Shift**, і перетягніть поле в 3D-режимі та перемістіть його в якомусь кутку арени.

4. Тепер клацніть, утримуючи клавішу **Shift**, і перетягніть поле в 3D-режимі та перемістіть його в якомусь кутку арени.
5. Виділіть поле та натисніть ctrl-C, ctrl-V (Windows, Linux) або ⌘ command-C, ⌘ command-V (macOS), щоб скопіювати та вставити його. Клацніть, утримуючи клавішу Shift, і перетягніть нове поле, щоб перемістити його в інше місце. Таким чином створіть третю коробку.
6. Переміщайте ящики так, щоб жодна коробка не опинилася в центрі арени. Ви також можете використовувати сині стрілки обертання, щоб обертати поля вздовж вертикальної осі. Зробити це також можна клацанням лівою кнопкою миші і перетягуванням правою кнопкою миші. Крім того, ви можете змінити кут **rotation** обертання вузлів **WoodenBox** у дереві сцени.
7. Після того, як ви будете задоволені результатом, збережіть світ за допомогою кнопки «Зберегти».

Додавання e-puck робота.

E-puck — це невеликий робот із колесами поєднаними диференціалом, 10 світлодіодами та кількома датчиками, включаючи 8 датчиків відстані та камеру. Зараз нас цікавить лише використання його коліс. Тепер ми збираємося додати у світ модель електронної робота. Переконайтеся, що симуляцію призупинено, а віртуальний час, що минув, дорівнює 0. Якщо це не так, скиньте симуляцію за допомогою кнопки **Reset**  .

Перед тим к вносити різні зміни у Світ Webots, принципово, щоб симуляція спочатку була поставлена на паузу і перезавантажена до початкового стану, тобто віртуальний лічильник часу на головній панелі інструментів повинен показувати **0:00:00:000**. В іншому випадку при кожному збереженні положення кожного 3D-об'єкта може накопичувати помилки. Тому будь-яка модифікація світу повинна виконуватися саме в такому порядку: **поставити на паузу, скинути, змінити і зберегти симуляцію**.

Нам не потрібно створювати робота з електронною шайбою з нуля, нам просто потрібно буде імпортувати **Nodes E-puck**. Цей вузол насправді є вузлом PROTO, як **RectangleArena** або **WoodenBox**, який ми представили раніше. Прототипування дозволяє створювати власні об'єкти та використовувати їх повторно.

Завдання №4: Виберіть останній вузол **WoodenBox** у вигляді дерева сцени. Натисніть кнопку Додати  у верхній частині ієрархії сцени. У діалоговому вікні виберіть вузли **PROTO (Webots Projects) / robots / gctronic / e-puck / E-puck (Robot)**. Посеред арени має з'явитися **E-puck**. Переміщайте і обертайте цього робота, так само, як ви це робили з коробками. Збережіть симуляцію та натисніть кнопку Запустити в **run real-time**  .

Робот повинен рухатися, блимати світлодіодами і об'їжджати перешкоди. Це тому, що він має контролер за замовчуванням з такою поведінкою. Можливо, ви помітили маленьке чорне вікно, що з'явилося у верхньому лівому куті 3D-огляду. Він показує зображення, зроблене камерою робота e-puck. Це зображення залишатиметься чорним, доки контролер робота не ввімкне камеру. Це маленьке вікно зображення можна переміщати, перетягуючи його. Його розмір також можна змінити, перетягнувши нижній правий кут. Нарешті, його можна закрити, натиснувши «x» у верхньому правому куті. Ви можете знову зробити його видимим у меню **Overlays** (Накладання), вибравши його в підменю **Camera Devices** (Пристрої камери). Тому що він нам не знадобиться, ви можете його фактично закрити. Тепер, поки симуляція запущена, давайте пограємо з фізикою:

Завдання №5: Застосуйте силу до робота, натиснувши alt ***+ клік лівою кнопкою миші + перетягування*** (використовуйте клавішу ⌥ option на деяких клавіатурах Mac). У Linux вам також слід натиснути клавішу ctrl на додаток до alt ***+ клік лівою кнопкою миші + перетягування.*** Прикласти зусилля до вузла **WoodenBox** неможливо, тому що за замовчуванням вони не мають маси і вважаються приклеєними на підлогу. Для включення фізики на вузлах **WoodenBox** слід встановити їх поле маси на певне значення (наприклад, 0,2 кг). Як тільки це буде зроблено, слід мати можливість прикладати силу і до них.

В результаті ви маєте отримати приблизно такий вигляд як показано на рис. 2.

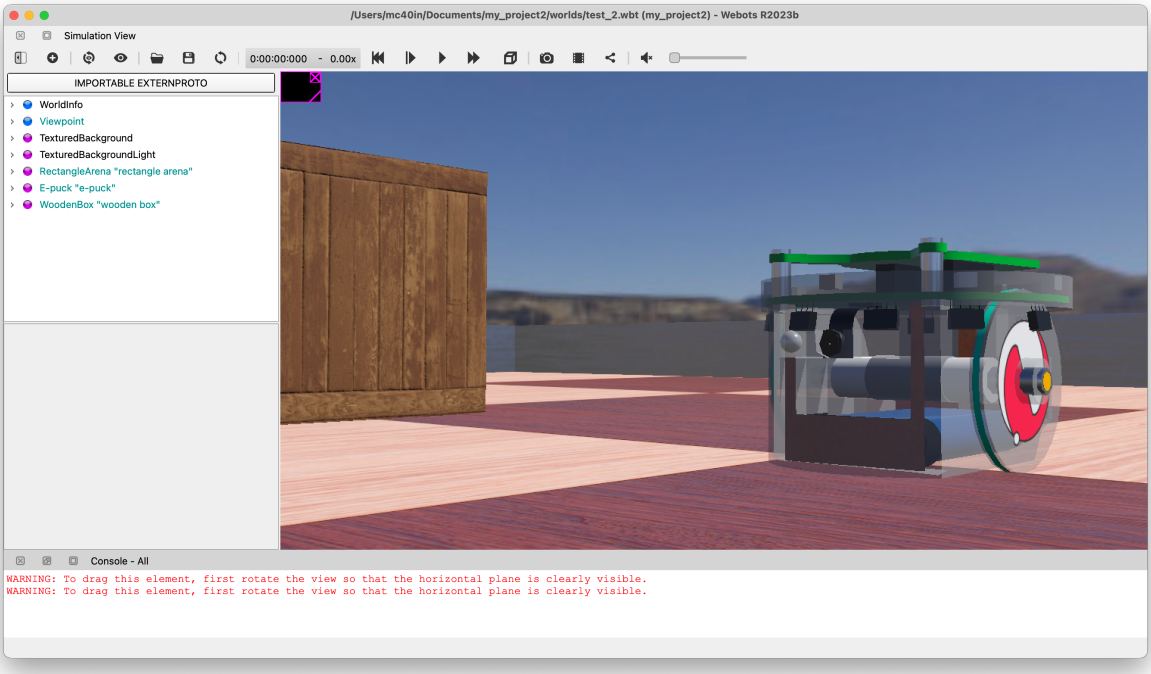


Рисунок 2. Віртуальний Світ із роботом E-puck.

Симуляцію можна поставити на паузу **(pause)**  , запускати крок за кроком **(run step-by-step)**  , в режимі реального часу **(real time)**  або в швидкиий режим **(fast)**  .

Тепер ми змінимо Світ і зменшимо крок симуляції фізики: це підвищить точність і стабільність симуляції (але зменшить максимальну швидкість симуляції).

Create a New Controller

Зараз ми запрограмуємо простий контролер, який просто змусить робота рухатися вперед.

Контролер – це програма, яка визначає поведінку робота. Контролери Webots можуть бути написані на наступних мовах програмування: C, C++, Java, Python, MATLAB, ROS тощо. Контролери C, C++ та Java повинні бути скомпільовані, перш ніж їх можна буде запускати як контролери роботів. Контролери Python і MATLAB є інтерпретованими мовами, тому вони працюватимуть без компіляції. У цьому посібнику ми будемо використовувати C як довідкову мову, але всі фрагменти коду також доступні на C++, Java, Python та MATLAB. Зверніться до розділу мови, щоб налаштувати контролер за допомогою іншої мови програмування.

Поле **Controller** вузла **Robot** визначає, який контролер в даний момент пов'язаний з роботом. Зверніть 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, який зазвичай породжується Webots. Оскільки контролери є незалежними процесами, вони не мають спільного адресного простору і можуть працювати на різних ядрах процесора.

Новий вихідний файл відображається у вікні текстового редактора Webots. Цей вихідний файл може бути скомпільований (якщо він написаний на C, C++ або Java) без будь-яких змін, однак поточний код не має реального ефекту. Тепер ми пов'яжемо новий контролер **epuck_go_forward** (або **EPuckGoForward**) з вузлом **E-puck**

Завдання №6: Створіть новий контролер C (або будь-якої іншої мови. В наших прикладах я буду використовувати мову **Python**) під назвою **epuck_go_forward** (для C++ та Java назвіть його **EPuckGoForward**) за допомогою **контролера File / New / New Robot Controller...** пункт меню. Це створить новий каталог epuck_go_forward (або EPuckGoForward) у my_first_simulation/controllers. Виберіть опцію, що пропонує вам відкрити вихідний файл у текстовому редакторі.

Якщо все гаразд, ваш робот повинен рухатися вперед. Робот буде рухатися з максимальною швидкістю протягом деякого часу, а потім зупиниться, коли колеса повернуться на 10 радіан.

Завдання №7: У режимі дерева сцени виберіть поле **controller** вузла **E-puck,** а потім скористайтеся редактором полів у нижній частині панелі Дерева сцен: натисніть кнопку **Select...** , а потім виберіть **epuck_go_forward** у списку. Як тільки контролер буде пов'язаний з роботом, врятуйте світ. Змініть програму, отримавши моторні пристрої **(leftMotor = robot.getDevice('left wheel motor'))** і застосувавши команду двигуна **(leftMotor.setPosition(10.0)):**

```
from controller import Robot, Motor

TIME_STEP = 64

# створити екземпляр робота.
robot = Robot()

# доступ до моторів
leftMotor = robot.getDevice('left wheel motor')
rightMotor = robot.getDevice('right wheel motor')
# установити цільову позицію мотора
leftMotor.setPosition(10.0)
rightMotor.setPosition(10.0)

while robot.step(TIME_STEP) != -1:
   pass
```

Збережіть змінений вихідний код **(File / Save Text File)**, скиньте налаштування та запустіть симуляцію.

У каталозі **controllers** вашого проекту створено каталог, що містить контролер **epuck_go_forward** (або **EPuckGoForward**). Каталог **epuck_go_forward** (або **EPuckGoForward**) містить **epuck_go_forward** (або **EPuckGoForward**) двійковий файл, згенерований після компіляції контролера (у Windows цей файл має розширення **.exe**). Ім'я директорії контролера має збігатися з ім'ям бінарного файлу.

Регулювання швидкості обертання робота.

Колеса роботів часто керуються за допомогою швидкості, а не позиціонування, як ми це робили в попередньому прикладі. Для того, щоб управляти моторами коліс по швидкості, потрібно виставити цільове положення на нескінченність і виставити потрібну швидкість:

Завдання №8: Змініть програму контролера, як показано нижче, перекомпілюйте її та запустіть:

```
from controller import Robot, Motor
TIME_STEP = 64
MAX_SPEED = 6.28

# створити екземпляр Robot.
robot = Robot()

# отримати обробник до двигунів і встановити цільове положення на нескінченність (регулювання швидкості)
leftMotor = robot.getDevice('left wheel motor')
rightMotor = robot.getDevice('right wheel motor')
leftMotor.setPosition(float('inf'))
rightMotor.setPosition(float('inf'))

# Встановіть частоту обертання двигуна на рівні 10% від MAX_SPEED.
leftMotor.setVelocity(0.1 * MAX_SPEED)
rightMotor.setVelocity(0.1 * MAX_SPEED)
while robot.step(TIME_STEP) != -1:
  pass
```

Тепер робот буде рухатися (колеса будуть обертатися зі швидкістю 0,2 радіана в секунду) і ніколи не зупинятися. Якщо нічого не відбувається, не забудьте скомпілювати свій код, вибравши пункт меню **Build / Build** або натиснувши на значок шестерні  над кодовою областю. Помилки компіляції відображаються в консолі червоним кольором. Якщо такі є, виправте їх і повторіть спробу компіляції. Потім перезавантажте світ.

Приклади програм керування різними мовами.

До завдання 7.

C

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

// Added a new include file
#include <webots/motor.h>

#define TIME_STEP 64

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

 // get the motor devices
 WbDeviceTag left_motor = wb_robot_get_device("left wheel motor");
 WbDeviceTag right_motor = wb_robot_get_device("right wheel motor");
 // set the target position of the motors
 wb_motor_set_position(left_motor, 10.0);
 wb_motor_set_position(right_motor, 10.0);

 while (wb_robot_step(TIME_STEP) != -1);

 wb_robot_cleanup();

 return 0;
}
```

C++

```
#include <webots/Robot.hpp>

// Added a new include file
#include <webots/Motor.hpp>

#define TIME_STEP 64

// All the webots classes are defined in the "webots" namespace
using namespace webots;

int main(int argc, char **argv) {
 Robot *robot = new Robot();

 // get the motor devices
 Motor *leftMotor = robot->getMotor("left wheel motor");
 Motor *rightMotor = robot->getMotor("right wheel motor");
 // set the target position of the motors
 leftMotor->setPosition(10.0);
 rightMotor->setPosition(10.0);

 while (robot->step(TIME_STEP) != -1);

 delete robot;

 return 0;
}
```

Java

```
import com.cyberbotics.webots.controller.Robot;

// Added a new include file
import com.cyberbotics.webots.controller.Motor;

public class EPuckGoForward {

 public static void main(String[] args) {

  int TIME_STEP = 64;

  Robot robot = new Robot();

  // get the motor devices
  Motor leftMotor = robot.getMotor("left wheel motor");
  Motor rightMotor = robot.getMotor("right wheel motor");
  // set the target position of the motors
  leftMotor.setPosition(10.0);
  rightMotor.setPosition(10.0);

  while (robot.step(TIME_STEP) != -1);
 }
}
```

MATLAB

```
function epuck_go_forward

TIME_STEP = 64;

% get the motor devices
left_motor = wb_robot_get_device('left wheel motor');
right_motor = wb_robot_get_device('right wheel motor');
% set the target position of the motors
wb_motor_set_position(left_motor, 10.0);
wb_motor_set_position(right_motor, 10.0);

while wb_robot_step(TIME_STEP) ~= -1
end
```

До завдання 8.

C

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

// Added a new include file
#include <webots/motor.h>

#define TIME_STEP 64

#define MAX_SPEED 6.28

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

  // get a handler to the motors and set target position to infinity (speed control)
  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);

  // set up the motor speeds at 10% of the MAX_SPEED.
  wb_motor_set_velocity(left_motor, 0.1 * MAX_SPEED);
  wb_motor_set_velocity(right_motor, 0.1 * MAX_SPEED);

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

  wb_robot_cleanup();

  return 0;
}
```

MATLAB

```
function epuck_go_forward

TIME_STEP = 64;

MAX_SPEED = 6.28;

% get a handler to the motors and set target position to infinity (speed control)
left_motor = wb_robot_get_device('left wheel motor');
right_motor = wb_robot_get_device('right wheel motor');
wb_motor_set_position(left_motor, inf);
wb_motor_set_position(right_motor, inf);

% set up the motor speeds at 10% of the MAX_SPEED.
wb_motor_set_velocity(left_motor, 0.1 * MAX_SPEED);
wb_motor_set_velocity(right_motor, 0.1 * MAX_SPEED);

while wb_robot_step(TIME_STEP) ~= -1
end
```

C++

```
#include <webots/Robot.hpp>

// Added a new include file
#include <webots/Motor.hpp>

#define TIME_STEP 64

#define MAX_SPEED 6.28

// All the webots classes are defined in the "webots" namespace
using namespace webots;

int main(int argc, char **argv) {
 Robot *robot = new Robot();

 // get a handler to the motors and set target position to infinity (speed control)
 Motor *leftMotor = robot->getMotor("left wheel motor");
 Motor *rightMotor = robot->getMotor("right wheel motor");
 leftMotor->setPosition(INFINITY);
 rightMotor->setPosition(INFINITY);

 // set up the motor speeds at 10% of the MAX_SPEED.
 leftMotor->setVelocity(0.1 * MAX_SPEED);
 rightMotor->setVelocity(0.1 * MAX_SPEED);

 while (robot->step(TIME_STEP) != -1);

 delete robot;

 return 0;
}
```

Java

```
import com.cyberbotics.webots.controller.Robot;

// Added a new include file
import com.cyberbotics.webots.controller.Motor;

public class EPuckGoForward {

 public static void main(String[] args) {

   int TIME_STEP = 64;

   double MAX_SPEED = 6.28;

   Robot robot = new Robot();

   // get a handler to the motors and set target position to infinity (speed control)
   Motor leftMotor = robot.getMotor("left wheel motor");
   Motor rightMotor = robot.getMotor("right wheel motor");
   leftMotor.setPosition(Double.POSITIVE_INFINITY);
   rightMotor.setPosition(Double.POSITIVE_INFINITY);

   // set up the motor speeds at 10% of the MAX_SPEED.
   leftMotor.setVelocity(0.1 * MAX_SPEED);
   rightMotor.setVelocity(0.1 * MAX_SPEED);

   while (robot.step(TIME_STEP) != -1);
 }
}
```

Практикум №2. Модифікація зовнішнього середовища.

У цьому уроці ми навчимось створювати прості об'єкти в навколишньому середовищі. Першим кроком буде створення кулі, яка буде взаємодіяти з навколишнім середовищем. Ми розглянемо кілька понять, пов'язаних з вузлами: яке їх значення, як їх створювати, як вони повинні бути пов'язані і т.д. Крім того, ми подивимося, як налаштувати фізику.

Буде представлено кілька видів вузлів. З їх детальним визначенням можна ознайомитися в Довідковому посібнику. Наявність діаграми вузлів перед вами також допоможе зрозуміти зв'язок успадкування вузлів.

Створіть новий проект так як було описано у практикумі №1.

Завдання No1: Переконайтеся, що файл світу **my_first_simulation.wbt** відкритий, а симуляція призупинена і знаходиться у віртуальному часі **0**. Використання файлу **File / Save World As...**, збережіть симуляцію як **obstacles.wbt**.

Модифікація підлоги.

Стандартний **RectangleArena** PROTO визначає просту підлогу, закріплену на статичному середовищі, тобто без вузла фізики, і оточену стінами. Інші готові поверхи доступні в бібліотеці об'єктів Webots. Тепер ми видалимо вузол **RectangleArena** і додамо просту підлогу, яку ми вручну оточимо стінами пізніше в цьому уроці.

Завдання №2: Щоб видалити **RectangleArena**, виділіть її або в 3D-перегляді, або в режимі дерева сцен клацанням лівої кнопки миші та натисніть клавішу `del` на клавіатурі. Крім того, ви можете клацнути правою кнопкою миші на панелі перегляду 3D і вибрати пункт **Delete** у контекстному меню (ви також можете скористатися контекстним меню безпосередньо на панелі дерева сцен). Вибираємо вузол **TexturedBackgroundLight** і натискаємо на кнопку **Add**. У діалоговому вікні, що відкриється, і виберіть вузли **PROTO nodes (Webots Projects) / objects / floors / Floor (Solid)**.

Нещодавно доданий **Floor** PROTO має розмір за замовчуванням 10m x 10m, але є можливість налаштувати його розмір, положення та текстуру, змінивши відповідні поля.

Завдання №3: У перегляді дерева сцени виберіть і розгорніть **Floor**. Змініть поле **Size** і встановіть для нього значення {1, 1}, щоб змінити його розмір на 1m x 1m.

Суцільний вузол (Solid Node)

У цьому підрозділі представлено найважливіший базовий вузол у Webots: вузол **Solid**, від якого походять багато інших вузлів.

Solid Node являє собою **тверде тіло (rigid body)**, тобто тіло, в якому деформацією можна знехтувати. Відстань між будь-якими двома заданими точками твердого тіла залишається постійним у часі незалежно від зовнішніх сил, що діють на нього. Наприклад, стіл, фаланга пальця робота або колесо є твердими тілами. М'які тіла (Soft bodies) і шарнірні об'єкти не є твердими тілами. Наприклад, мотузка, шина, губка або шарнірна рука робота не є твердими тілами. Однак поєднану систему можна розбити на кілька жорстких тіл.

Фізичний движок Webots призначений лише для моделювання твердих тіл. Важливим кроком при проектуванні симуляції є розбиття різних сутностей на окремі тверді тіла.

Щоб визначити жорстке тіло, вам доведеться створити вузол **Solid**. Усередині цього вузла ви встановите різні підвузли, що відповідають характеристикам жорсткого тіла. На наступному на рисунку 1 зображено тверде тіло і його підвузли. Графічне представлення вузла **Solid** визначається вузлами **Shape**, що заповнюють його **дочірній** список. Межі колізії визначаються в його полі границі об’єкту **(boundingObject)**. Графічне зображення і форма зіткнення часто, але не обов'язково ідентичні. Нарешті, поле фізики **(physics)** визначає, чи належить об'єкт до динамічного або до статичного середовища. Всі ці підвузли є необов'язковими, але поле **physics** потребує визначення **boundingObject**.

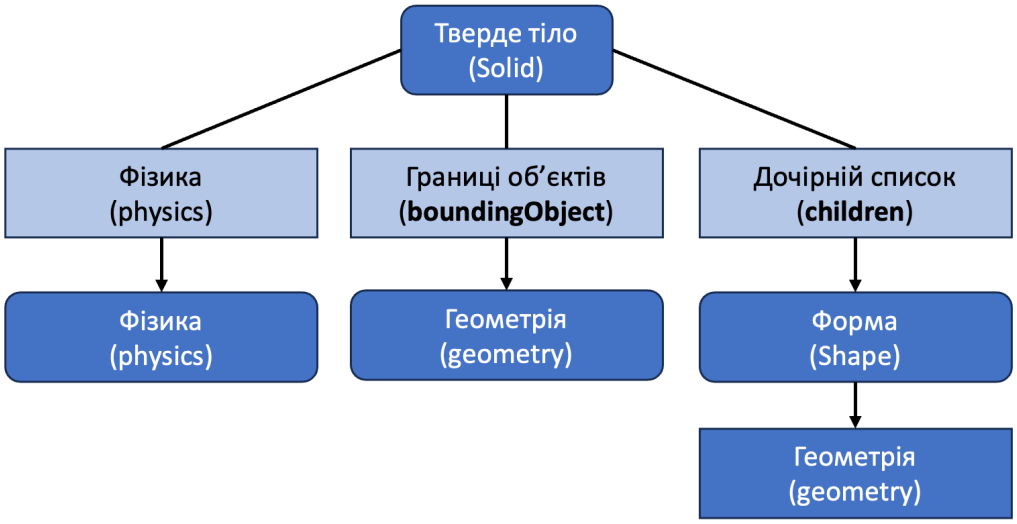


Рисунок 1. Найпростіша модель твердого тіла у Webots, що має графічне зображення, фізичне обмеження об'єкту і знаходиться в динамічному середовищі.

Поле «Геометрія» на рис 1 означає будь-який геометричний примітив. Фактично, його можна представити у вигляді сфери, коробки, циліндра тощо.

Створення круглого тіла.

Додаємо кулю до симуляції. Ця куля буде змодельована як тверде тіло, як показано на рис 2. Вузол **Sphere** буде використовуватися для визначення геометрії нашої кулі.

Завдання №4: У перегляді дерева сцени виберіть останній вузол і натисніть кнопку **Add**. У діалоговому вікні відкрийте розділ **Bases Nodes** та виберіть вузол **Solid**. У вигляді дерева сцени розгорніть вузол **Solid** та виберіть його **children** поле. Додайте до нього вузол Shape за допомогою кнопки **Add**. Виділіть поле **appearance** вузла **Shape** і за допомогою кнопки **Add** додайте вузол **PBRAppearance**.

1. Додайте вузол **Sphere** як поле **geometry** новоствореного вузла **Shape.**
2. Розгорніть вузол **PBRAppearance** і змініть його поле металевості **(metalness)** на 0, а поле шорсткості **(roughness)** на 1.
3. Додайте ще один вузол **Sphere** до поля межі об’єкта **boundingObject** об'єкта **Solid**.
4. Нарешті, додайте вузол **Physics** до поля фізики **Solid**.
5. Змінюючи поле **translation** вузла **Solid**, розташуйте м'яч перед роботом (наприклад, на {0.2, 0, 0.2}).
6. Збережіть симуляцію.
7. Результат зображений на цьому рисунку 2.

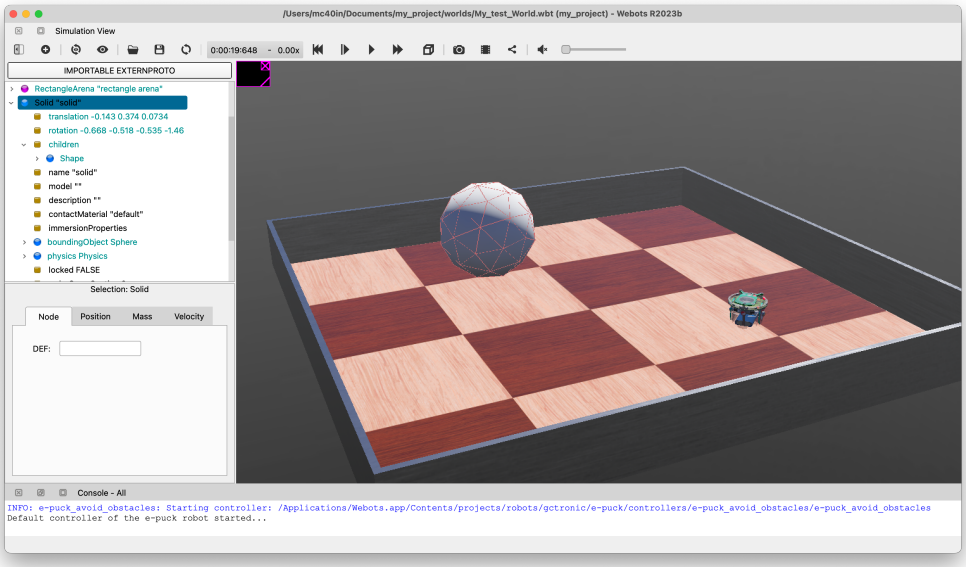


Рисунок 2. – Наближений вигляд вікна симуляції після додавання твердого тіла Webots.

Коли починається імітація, м'яч вдаряється об підлогу. Ви можете переміщати м'яч, прикладаючи до нього силу (ctrl + alt + лівий клік + перетягування). Точки контакту між м'ячем і підлогою можна відобразити у вигляді блакитних

ліній, увімкнувши пункт меню **View / Optional Rendering / Show Contact Points** який покаже точки контакту.

Геометрія (Geometries)

Для визначення кулі ми використовували вузол **Sphere** у двох різних контекстах: для графічного представлення (дочірні елементи (**children**)) та для визначення фізичних меж (межа об'єкта (boundingObject)). Всі вузли Geometry (такі як вузол Sphere) можуть бути використані в графічному контексті. Однак лише підмножина з них може бути використана у фізичному контексті. Діаграма вузлів показує, які вузли підтримуються в кожному контексті.
Тепер ми зспробуємо зменшити розмір **Sphere** та підвищити її графічну якість за рахунок збільшення кількості трикутників, які використовуються для її представлення.

Завдання №5: Для кожного вузла Сфери, що визначає кулю, встановіть його поле радіуса **`radius`** на 0,05, а поле поділу **`subdivision`** на 2.

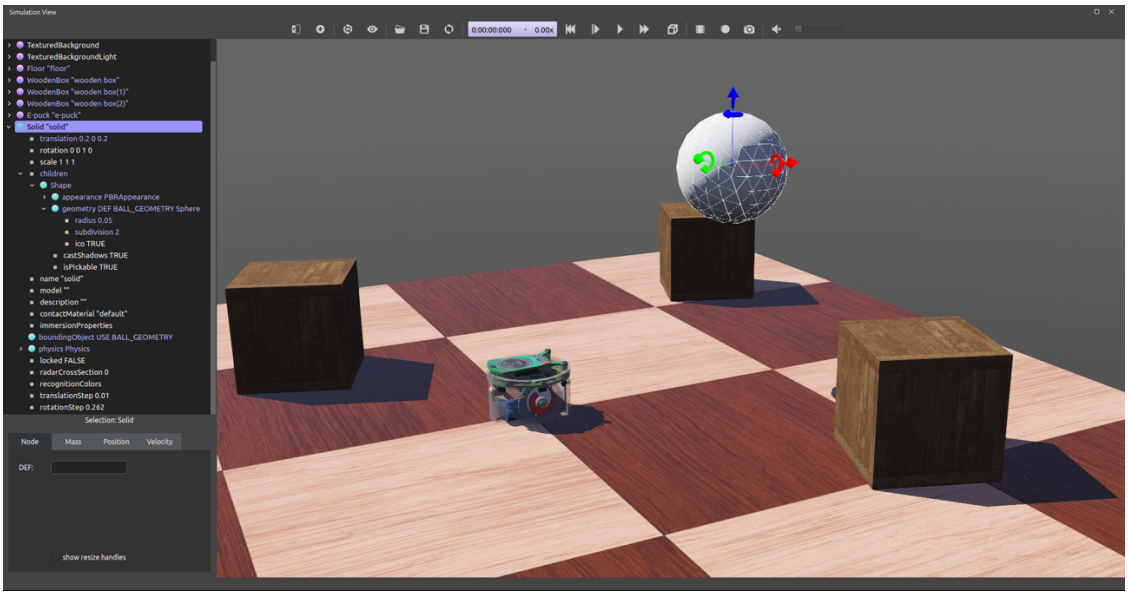
Система наслідування (DEF-USE Mechanism)

Механізм **DEF-USE** дозволяє визначити вузол в одному місці та повторно використовувати це визначення в іншому місці дерева сцени. Це корисно для уникнення дублювання однакових вузлів у файлах **Світу**. Крім того, це також дозволяє користувачам змінювати кілька об'єктів одночасно. Ось як це працює: спочатку вузол позначається рядком **DEF**. Потім копії цього вузла можуть бути повторно використані в іншому місці з ключовим словом **USE**. Редагувати можна тільки поля вузла **DEF**, поля вузла **USE** успадковуються від вузла **DEF** і не можуть бути змінені. Цей механізм залежить від порядку вузлів у файлі **World**. Вузол **DEF** повинен бути визначений перед будь-яким відповідним вузлом **USE**.
Два визначення **Sphere**, які ми використовували раніше для визначення кулі, є зайвими. Тепер ми об'єднаємо ці дві сфери лише один раз за допомогою механізму **DEF-USE**.

Завдання №6: Виберіть перший вузол **Sphere** (дочірній вузол **Shape**) у вигляді дерева сцени. Редактор полів перегляду дерева сцени дозволяє ввести рядок **DEF**.
1. Введіть **BALL_GEOMETRY** у це поле.
2. Виділіть поле **boundingObject** (що містить другий вузол **Sphere**) і очистіть його, клацнувши правою кнопкою миші поле в дереві сцени і вибравши пункт **Delete** у контекстному меню, що з'явиться.
3. Потім вибираємо поле **boundingObject** і натискаємо на кнопку **Add**, а в діалоговому вікні вибираємо **USE / BALL_GEOMETRY**.

Тепер зміна **radius** поля першого вузла **Sphere** також змінює його **boundingObject**.
Для зручності поле **boundingObject** також приймає вузол **Shape** (а не вузол **Sphere** безпосередньо). Також можна було б використовувати той самий механізм **DEF-USE** на рівні **Shape**, як показано на цьому рисунку. На даний момент найбільшою перевагою є можливість також використовувати цю фігуру безпосередньо для графічних цілей. Пізніше цей механізм виявиться дуже корисним для деяких датчиків.

Завдання №7: Створіть другу кулю з тими ж параметрами, але використовуючи вузол **Shape** (а не вузол **Sphere** безпосередньо) для механізму **DEF-USE**.



Додавання стін (Add Walls)

Для того, щоб перевірити свій прогрес, встановлюйте самостійно чотири стіни, щоб оточувати навколишнє середовище. Стіни повинні бути статично визначені навколишньому середовищу. Щоб зрозуміти різницю між статичним і динамічним, візьмемо визначений об'єкт (куля) над землею. Якщо вузол фізики (**physics**) має значення **NULL**, він залишатиметься замороженим у повітрі під час симуляції (статичний випадок). Якщо поле **physics** містить вузол **physics**, то він потрапить під дію гравітації (динамічний випадок).
Використовуйте механізм **DEF-USE** якомога частіше на рівні **Shape**, а не на рівні Геометрії (**Geometry**). Дійсно, зручніше додавати проміжний вузол **Shape** в поле **boundingObject** вузла **Solid**. Кращим примітивом геометрії для реалізації стін є вузол **Box**. Для всіх стін повинна бути визначена тільки одна фігура. Очікуваний результат показаний на рисунку 2.6.

Завдання №8: Додайте чотири стіни без фізики та використовуючи лише одне визначення вузла **Shape.**

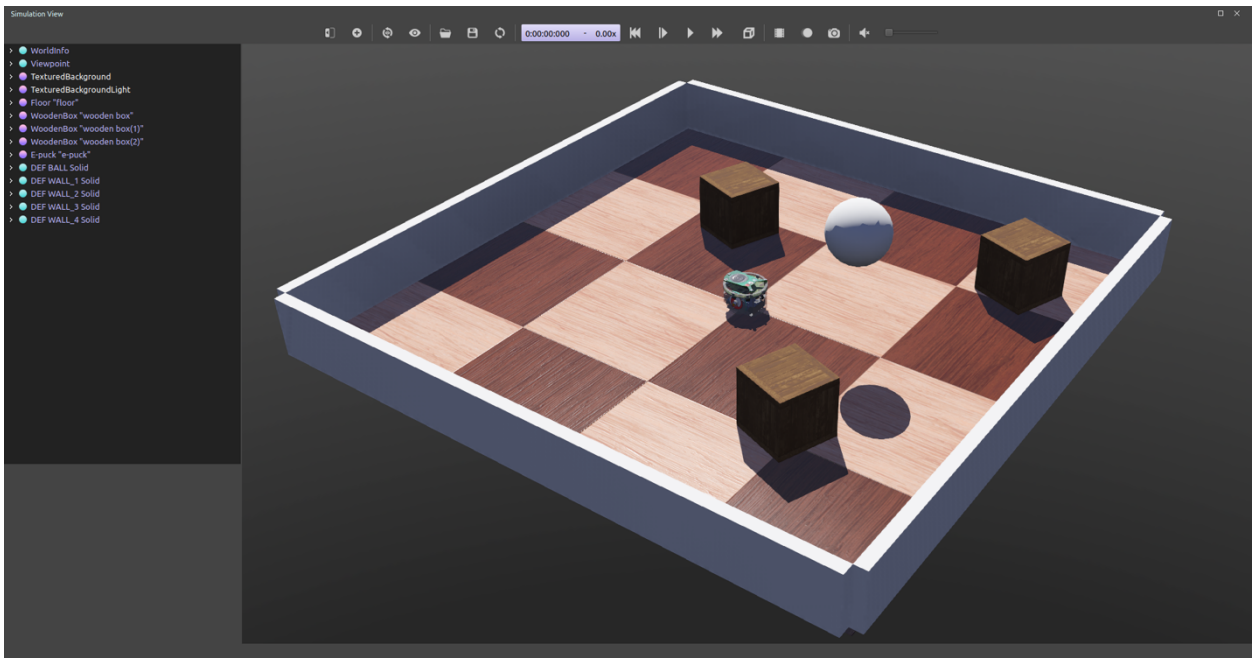


Рисунок 2.6 Результати створення Світу.

Ефективність (Efficiency)

Моделювання твердих тіл є із апаратної точки зору вимагає достатньо високих ресурсів. Швидкість моделювання може бути збільшена за рахунок мінімізації кількості обмежувальних об'єктів, мінімізації обмежень між ними та максимізації поля **WorldInfo.basicTimeStep**. У кожній симуляції потрібно знайти компроміс між швидкістю симуляції та реалістичністю.

Висновок (Conclusion)

В кінці цього практикуму ви зможете створювати прості середовища на основі твердих тіл. Ви можете додавати вузли з перегляду дерева сцени та змінювати їхні поля. Ви ознайомилися з вузлами **Solid, Physics, Shape, Sphere** та **Box.** Ви також бачили механізм **DEF-USE**, який дозволяє зменшити кількість вузлів дерева сцени.

Практикум №3. Особливості програмування контролерів

В цій роботі ми більш детально розглянемо програмуванням контролерів роботів. Ми розробимо простий контролер, який дозволить уникнути перешкод, створених у попередніх уроках.

Ця робота познайомить вас з основами програмування роботів у Webots. В кінці цієї роботи ви повинні зрозуміти, який зв'язок між вузлами дерева сцени і API (API – Application Programming Interface, Інтерфейс прикладного програмування.) контролера, як контролер робота повинен бути ініціалізований та підготовлений, як ініціалізувати роботизовані пристрої, як отримати значення датчиків, як дати команду виконавчим механізмам і як запрограмувати простий цикл зворотного зв'язку.

У цій роботі розглядається лише правильне використання функцій Webots. Вивчення алгоритмів робототехніки виходить за рамки цілей цього підручника, тому воно не буде розглядатися. Для опрацювання цього розділу потрібні деякі елементарні знання програмування.

Завдання #1: Збережіть попередній світ як **collision_avoidance.wbt.** Створіть новий контролер Python (або будь-якої іншої мови) під назвою **epuck_avoid_collision** (для C++ і Java назвіть його **EPuckAvoidCollision**) з **File / New / New Robot Controller...** пункт меню. Змініть поле контролера вузла E-puck, щоб пов'язати його з новим контролером.

Будова мобільного робота E-puck.

Програмування контролера вимагає деяких занань, пов'язаних з моделлю E-puck. Для того, щоб створити алгоритм запобігання зіткненням, нам потрібно прочитати значення його 8 інфрачервоних датчиків відстані, розташованих навколо, і нам потрібно привести в дію два його колеса. На рисунку наведено зображення та те, як датчики відстані розподіляються навколо башти та напрямок руху електронного робота.

Датчики відстані описуються 8 вузлами DistanceSensor в ієрархії робота. На ці вузли посилаються поля імен (від ps0 до ps7). Пізніше ми пояснимо, як визначаються ці вузли. Наразі просто зауважте, що доступ до вузла DistanceSensor можна отримати через відповідний модуль Webots API (через файл webots/distance_sensor.h include). Значення, що повертаються датчиками відстані, масштабуються від 0 до 4096 (лінійно-пропорційно відстані). Значення, що відповідає 4096 означає, що відбивається велика кількість світла (перешкода близько), а 0 означає, що світло не відбивається (немає перешкоди).

API **контролера** – це програмний інтерфейс, який дає вам доступ до змодельованих датчиків і виконавчих механізмів робота. Наприклад, включення файлу webots/distance_sensor.h дозволяє використовувати функції wb_distance_sensor_*, і за допомогою цих функцій ви можете запитувати значення вузлів DistanceSensor.

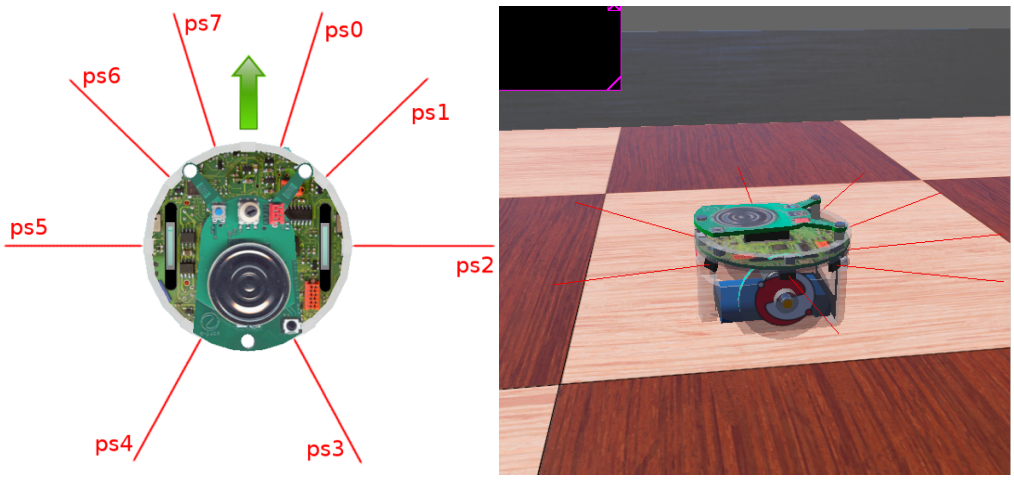


Рисунок – Розташування датчиків відстані. Вид зверху на модель e-puck. Зелена стрілка вказує на передню частину робота. Червоні лінії позначають напрямки інфрачервоних датчиків відстані.

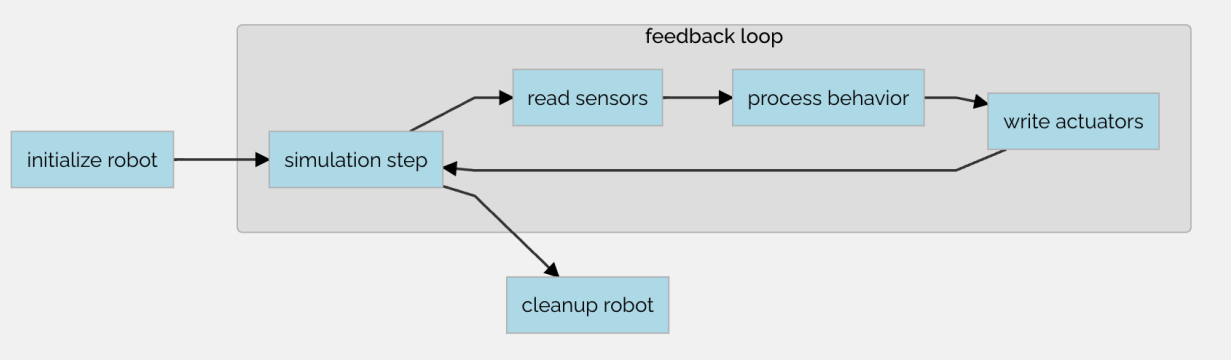


Рисунок - Графік станів UML простого циклу зворотного зв'язку.

Програмування контролеру.

Давайте спробуємо запрограмувати просту поведінку уникнення зіткнень. Ви запрограмуєте робота рухатися вперед до тих пір, поки перешкода не буде виявлена передніми датчиками відстані, а потім повертати в напрямку де перешкода відсутня. Для цього скористаємося простим циклом зворотного зв'язку, зображеним на автоматі станів UML на рисунку.
Повний код цього контролера наведено нижче.

Завдання #2: На початку файлу контролера додайте директиви імпорту, що відповідають вузлам Robot, DistanceSensor та Motor, щоб мати можливість використовувати відповідний API:

```
from controller import Robot, DistanceSensor, Motor
```

Відразу після операторів import визначають змінну, яка визначає тривалість кожного кроку фізики. Цей макрос буде використовуватися як аргумент функції **Robot::step**, а також для включення пристроїв. Ця тривалість вказується в мілісекундах і повинна бути кратною значенню в полі **basicTimeStep** вузла **WorldInfo**.

```
TIME_STEP = 64
```

Основна функція полягає в тому, де починається виконання програми контролера. Аргументи, що передаються функції **main**, задаються полем controllerArgs вузла Robot. API Webots має бути ініціалізований за допомогою функції **wb_robot_init** і очищений за допомогою функції **wb_robot_cleanup**.

Завдання #3: У оскільки Python це інтерпретуєма мова то в ній немає головної функції, програма починає своє виконання зі старту файлу:
 # create the Robot instance.

```
robot = Robot()

# initialize devices

# feedback loop: step simulation until receiving an exit event

while robot.step(TIME_STEP) != -1:

    # read sensors outputs

    # process behavior

    # write actuators inputs
```

На робот-пристрій посилається тег **WbDeviceTag**. WbDeviceTag витягується функцією **wb_robot_get_device**. Потім він використовується як перший аргумент у кожному виклику функції, що стосується цього пристрою. Такий датчик, як **DistanceSensor**, повинен бути ввімкнений перед використанням. Другий аргумент функції **enable** визначає, з якою швидкістю буде оновлюватися датчик.

Завдання #4: Відразу після коментаря **// initialize devices**, отримайте та увімкніть датчики відстані наступним чином:

```
# initialize devices

ps = []

psNames = [

   'ps0', 'ps1', 'ps2', 'ps3',

   'ps4', 'ps5', 'ps6', 'ps7'

]

for i in range(8):

   ps.append(robot.getDevice(psNames[i]))

   ps[i].enable(TIME_STEP)
```

Після ініціалізації пристроїв ініціалізують двигуни:

```
leftMotor = robot.getDevice('left wheel motor')

rightMotor = robot.getDevice('right wheel motor')

leftMotor.setPosition(float('inf'))

rightMotor.setPosition(float('inf'))

leftMotor.setVelocity(0.0)

rightMotor.setVelocity(0.0)
```

В основному циклі, відразу після коментаря **# read sensors outputs**, зчитуємо значення датчика відстані наступним чином:

```
# read sensors outputs

psValues = []

for i in range(8):

   psValues.append(ps[i].getValue())
```

В основному циклі, відразу після коментаря **# process behavior**, визначити, чи відбувається зіткнення (тобто значення, яке повертає датчик відстані, більше порогового значення) наступним чином:

```
# detect obstacles

right_obstacle = psValues[0] > 80.0 or psValues[1] > 80.0 or psValues[2] > 80.0

left_obstacle = psValues[5] > 80.0 or psValues[6] > 80.0 or psValues[7] > 80.0
```

Визначивши наявність перешкоди треба передати сигнали на колеса робота для об'їзду перешкоди наступним чином:

```
MAX_SPEED = 6.28

...

# initialize motor speeds at 50% of MAX_SPEED.

leftSpeed  = 0.5 * MAX_SPEED

rightSpeed = 0.5 * MAX_SPEED

# modify speeds according to obstacles

if left_obstacle:

   # turn right

   leftSpeed  = 0.5 * MAX_SPEED

   rightSpeed = -0.5 * MAX_SPEED

elif right_obstacle:

   # turn left

   leftSpeed  = -0.5 * MAX_SPEED

   rightSpeed = 0.5 * MAX_SPEED

# write actuators inputs

leftMotor.setVelocity(leftSpeed)

rightMotor.setVelocity(rightSpeed)
```

Збережіть код, вибравши піктограму **File/Save Text File** пункт меню. Перезавантажте світ.

Список використаних джерел.

1. Webots User Guide URL https://cyberbotics.com/doc/guide/index/

2. Сорокін М.С. Основи робототехніки: конспект лекцій для студентів першого (бакалавр.) рівня вищ. освіти ден. та заоч. форми навчання, спец.: 141 Електроенергетика, електротехніка та електромеханіка / М. С. Сорокін ; Держ. біотехнол. ун-т. - Харків : [б. в.], 2024. - 94 с. - Б. ц.
3. Сорокін М.С. Основи програмування Python: метод. вказівки для виконання практ. завдань студентів першого (бакалавр.) рівня вищ. освіти ден. та заоч. форми навчання, спец. 163 Біомедична інженерія / уклад. М. С. Сорокін. - Харків : ДБТУ, 2023. - 109 с. - Б. ц.

Міністерство освіти і науки України

ДЕРЖАВНИЙ БІОТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет енергетики, робототехніки та комп'ютерних технологій

Кафедра електромеханіки та робототехніки

ПРОГРАМУВАННЯ МОБІЛЬНИХ РОБОТІВ

Методичні вказівки

для виконання лабораторних робіт

з дисципліни "Основи робототехніки"

Для студентів першого (бакалаврського) рівня вищої освіти денної та заочної форми навчання, спеціальностей:

141 Електроенергетика, електротехніка та електромеханіка

Затверджено рішенням Науково-методичної ради

Факультету ЕРКТ

Протокол № 1

від 22.10.24р.

Харків

2024