МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук,
управління та адміністрування
Кафедра інформаційних технологій

Кваліфікаційна робота бакалавра

на тему: Розробка імітаційної моделі позиційної системи рухів NAO-робота

Виконав студент групи К-41
спеціальності 122 Комп'ютерні науки
Холлац Данік Валерійович

Керівник к.т.н., доцент
Гнатовська Ганна Арнольдівна

Рецензент к.геогр.н, доцент
Кузніченко Світлана Дмитрівна

Одеса 2021

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

Скорочення

API – Application Programming Interface – програмний інтерфейс.

ODE – Open Dynamics Enginge – високопродуктивна бібліотека з відкритим вихідним кодом для моделювання динаміки твердого тіла.

UML – Unified Modelling Language – уніфікована мова моделювання.

Терміни

DrawStuff – бібліотека для рендеринга простих 3D-об'єктів у віртуальному середовищі з метою демонстрації можливостей ODE.

Eclipse – вільне модульне інтегроване середовище розробки програмного забезпечення.

make – утиліта, яка автоматизує процес перетворення файлів з однієї форми в іншу. Найчастіше це компіляція вихідного коду в об'єктні файли і подальша компонування у виконувані файли або бібліотеки.

Makefile – файл, у якому зазначені команди, які виконує утиліта make.

NAO – автономний програмований людиноподібний робот-гуманоїд, розроблений компанією Aldebaran Robotics.

Papyrus – інструмент UML 2 з відкритим кодом, заснований на Eclipse.

Use Case – варіант використання, у розробці програмного забезпечення та системному проектуванні це опис поведінки системи.

ВСТУП

Сьогодні людина все активніше спрямовує роботів на обслуговування своїх потреб. Особливо це стосується тих сфер, де потрібно небезпечна або рутинна робота.

Розробка і розвиток антропоморфних роботів – одне з ключових напрямків робототехніки. Особливо це актуально для універсальних роботів-помічників, які з часом можуть стати звичним атрибутом в домашньому побуті. NAO-робота є автономним, програмованим, який розроблений французькою компанією Aldebaran Robotics.

Французька компанія з робототехники Aldebaran Robotics є власником та розробником цього роботу. Заснована в 2005 році Бруно Мезоном компанія Aldebaran Robotics має офіси у Франції, Китаї, Японії і США, розробляє, виробляє і продає автономних роботів-гуманоїдів з метою поліпшення добробуту людей. ALDEBARAN Robotics об'єднує понад 150 співробітників, у тому числі 60 інженерів і докторів наук, які беруть участь в розробці і виробництві роботів. Починаючи з 2004 року NAO, який має висоту 58 см, був створений, щоб стати дружелюбними компаньйоном для будинку. З 2008 року випущено вже кілька версій робота [1].

Найвідомішим екземпляром NAO є Nao Academics Edition, який розроблений для університетів і лабораторій для допомоги в наукових дослідженнях та освіті. Він був випущений для установ в 2008 році і став доступний для покупців з 2011 року. Роботи Nao використовувались для дослідницьких та навчальних цілей у численних академічних закладах світу. Станом на 2015 рік понад 5000 роботів Nao використовуються в більш ніж 50 країнах. Пізніші поновлення для платформи Nao включають версії Nao Next Gen і з 2014 року Nao Evolution [1].

Aldebaran Robotics випустила останнє покоління своїх програмованих роботів-гуманоїдів, які призначені для досліджень, навчання і, в більш загальному плані, для вивчення нової сфері службової робототехніки.

Збільшення людяності і комерціалізація роботів – завдання компанії Aldebaran Robotics. Універсальна платформа NAO-роботу надає можливість здійснювати моделювання переміщення в просторі за допомогою програмних засобів: Choreography – діаграмна мова програмування для управління. Програма може взаємодіяти з мовами Urbi і Python і оперувати з окремими модулями мови C ++; Visual Studio із встановленим SDK for NAO; мова програмування Python з встановленим SDK for NAO [2].

Для навчання роботів необов'язково будувати фізичну модель робота – іноді це може бути економічно невигідно, іноді навіть небезпечно для людини, тому створення комп'ютерних стимуляторів NAO-робота, засобами створення імітаційних моделей є актуальним завданням.

Метою дипломного проектування є створення імітаційної моделі NAO-робота, яка реалізує можливість симуляції рухів у просторі, що дозволить проводити модельні експерименти з NAO-роботом щодо збільшення можливості маніпуляцій та сприяє розвиненню функціональних можливостей NAO-роботів для подальшого впровадження.

Застосування розробленої імітаційної моделі позиційної системи рухів NAO-робота дозволяє отримувати безкоштовне програмне забезпечення і розвивати функціональні можливості роботів засобами комп’ютерної симуляції для подальшого розвитку універсальних роботів-помічників. Використання розробниками такої моделі забезпечить можливість вирішення завдань моделювання зручними засобами та інструментами розробки.

Дипломна робота містить в собі 60 сторінок, 26 рисунків, 16 посилань.

1 ХАРАКТЕРИСТИКА ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАВДАННЯ

1.1 Опис предметної області

Підвищити ефективність виробництва на початку XIX століття мали машини, в середині XX століття – роботи, а сьогодні – роботи зі штучним інтелектом. Розробники та дослідники прагнуть зробити роботів більш ефективними, але також ще і зручними, інтерактивними, безпечними, що співпрацюють.

З розвитком автоматизованих систем управління, розширенням сфер застосування засобів обчислювальної техніки значно різноманітніше стає коло завдань, які необхідно вирішувати. Практика вимагає постановки і вирішення все більш складних завдань. У цих умовах побудова адекватних моделей і розробка методів їх рішення стають все більш насущними проблемами. Особливо це стосується таких завдань, в яких необхідно одночасно враховувати фактори невизначеності, динамічну взаємну обумовленість поточних рішень, наступних подій, комплексну взаємозалежність між досліджуваними факторами [3].

Область застосування автономних програмованих людиноподібних NAO-роботів вельми велика: від позитивного впливу на повсякденне життя людини, починаючи від допомоги по дому і розваги до терапії аутизму. Тому розробники сконцентровані на поліпшенні взаємодії робота і людини в двосторонньому порядку з допомогою мови тіла.

Основною проблемою є програмне забезпечення (ПЗ) під роботів, а точніше його бракує. Вже написано порівняно багато ПО, але цього критично недостатньо для перетворення роботів в самостійно приймаючих рішення гуманоїдів. Досить серйозна прогалина в ПЗ на сьогоднішній день – відсутність повноцінного хмарного рішення під NAO. Але є і позитивні моменти для розробників: інструменти програмування розвинені досить

добре. Є візуальний конструктор, який дозволяє побудувати логіку руху робота за допомогою базових визначених функцій. При цьому розробник має можливість дописати більш складну логіку при необхідності.

Роботи NAO – великорозмірні антропоморфні роботи, які мають високоякісну систему з безліччю ступенів свободи, вбудованою системою датчиків і взаємодії з навколишнім середовищем, а також потужне керуюче ядро. Для здійснення розробки імітаційної моделі позиційної системи рухів необхідно вивчити роботів, програмне забезпечення для роботи з ними, а також їх SDK. Розробити бібліотеки руху і створити послідовності рухів та розробити відповідні імітаційні демонстраційні моделі [4].

RPA (роботизація бізнес-процесів) – явище не нове. Роботизація є логічним продовженням використання макросів, які тепер діють в будь-яких системах і роблять це набагато швидше людини. Багато фахівців У більшості випадків розробити і впровадити робота виходить дешевше і швидше, ніж провести повноцінну інтеграцію програмного забезпечення, особливо якщо воно працює на різних платформах, є власною розробкою.

1.2 Області застосування Nao-роботів

Потенціал використання роботів для допомоги людям величезний. Роботи можуть служити помічниками, товаришами і друзями. Вони можуть виконувати унікальні функції в сфері навчання. Але щоб люди могли подолати створене індустрією розваг почуття страху, роботи повинні бути милими і веселими. А ще вони повинні імітувати зовнішній вигляд і поведінку людей.

Nao доступний як дослідницький робот для шкіл, коледжів та університетів для навчання програмуванню та проведення досліджень взаємодії людина-робот.

Починаючи з 2011 року, понад 200 академічних установ у всьому світі використовували робота, включаючи Університет Хартфордширу, Індійський

інститут інформаційних технологій, Токійський університет, Індійський технологічний інститут, Саудівська Аравія, Університет ім. Короля Фахда з нафти та мінералів, Університет Південного Уельсу і Університет штату Монтана. У 2012 році для навчання дітей з діагнозом аутизм використовували Nao-роботів у школі Великобританії. У більш широкому контексті Nao-роботи використовували численні британські школи для ознайомлення дітей з роботами та робототехнічною галуззю [1].

У серпні 2018 року RobotLAB випустила Engage! K12. Це онлайн-навчальна платформа для шкіл, яка покращує використання NAO для STEM, кодування та інженерії. У лютому 2018 року фінська компанія Utelias Technologies випустила Elias Robot, навчальний додаток, який допомагає вивчати мови за допомогою NAO-роботів.

У вересні 2015 року Французький інститут здоров'я та медичних досліджень використовував Nao-роботів для тестування системи роботизованої «автобіографічної пам'яті», призначеної для допомоги у тренуванні та наданню допомоги літнім пацієнтам.

Роботи-гуманоїди надали значну допомогу літнім людям завдяки своєму інтерактивному і інтелектуальному інтерфейсу, що дозволяє задовольнити індивідуальні потреби всіх. Робот може отримувати прості і індивідуальні інструкції для таких дій, як вправи, а також дозволяє обробляти медичні результати і передавати їх медичним працівникам, забезпечуючи ефективний моніторинг здоров'я літніх людей. Функція телеприсутності дозволяє літнім людям підтримувати постійний контакт зі своїми родинами і друзями і виступає як допоміжна технологія, що робить Nao-роботів зручними і корисними помічниками з догляду за літніми людьми [2].

1.3 Технічні характеристики

Nao – найрозвиненіша гуманоїдна платформа, наявна на ринку сьогодні. Робот Nao це ідеальний партнер для дослідження і викладання в

області робототехніки і штучного інтелекту. Його різні датчики, 25 ступенів свободи, вбудований комп'ютер і графічно оформлене програмне забезпечення роблять робота унікальним інструментом для досліджень. На додаток до 25 ступенями свободи у Nao є інерційний датчик і багато інших сенсорів, включаючи сонар високої точності. Голосові динаміки і пристрій розпізнавання, так само як і звукова орієнтованість, роблять робота по-справжньому інтерактивним інструментом. Інфрачервоний порт і Wi-fi дозволяє легко спілкуватися з іншими пристроями [1].

Роботом Nao можливо здійснювати керування через призначену для користувача програму Choregraphe®, з програмованими модулями C ++, можна використовувати мови програмування Python або Urbi. Nao поставляється з повною документацією і безліччю прикладів коду програмування. Розробники можуть вносити свої доробки і нові програмні модулі, вносити зміни. Завдяки міжплатформеній розробці (Linux, Mac OS і Windows), розробники можуть створювати нові поведінки для Nao-роботів, через що збільшується масив мов включаючи C, C ++, Urbi, і Python.

В процесі розробки робота NAO фахівці Aldebaran використовували рішення SOLIDWORKS Premium для проектування, SOLIDWORKS Simulation Premium для аналізу, SOLIDWORKS Plastics для аналізу прес-форм і SOLIDWORKS Enterprise PDM для управління даними про вироби. NAO складається з 1400 деталей, це стільки ж, скільки в невеликому автомобілі [2].

Правильно розмістити всі ці деталі в обмеженому просторі (висота NAO становить всього 58 сантиметрів) – це непросте завдання. SOLIDWORKS допомагає нам моделювати продуктивність і перевіряти конфлікти компонентів, забезпечуючи їх розміщення та функціонування відповідно до проекту. Схема з зазначенням технічних характеристик NAO-робота наведена на рисунку 1.1.

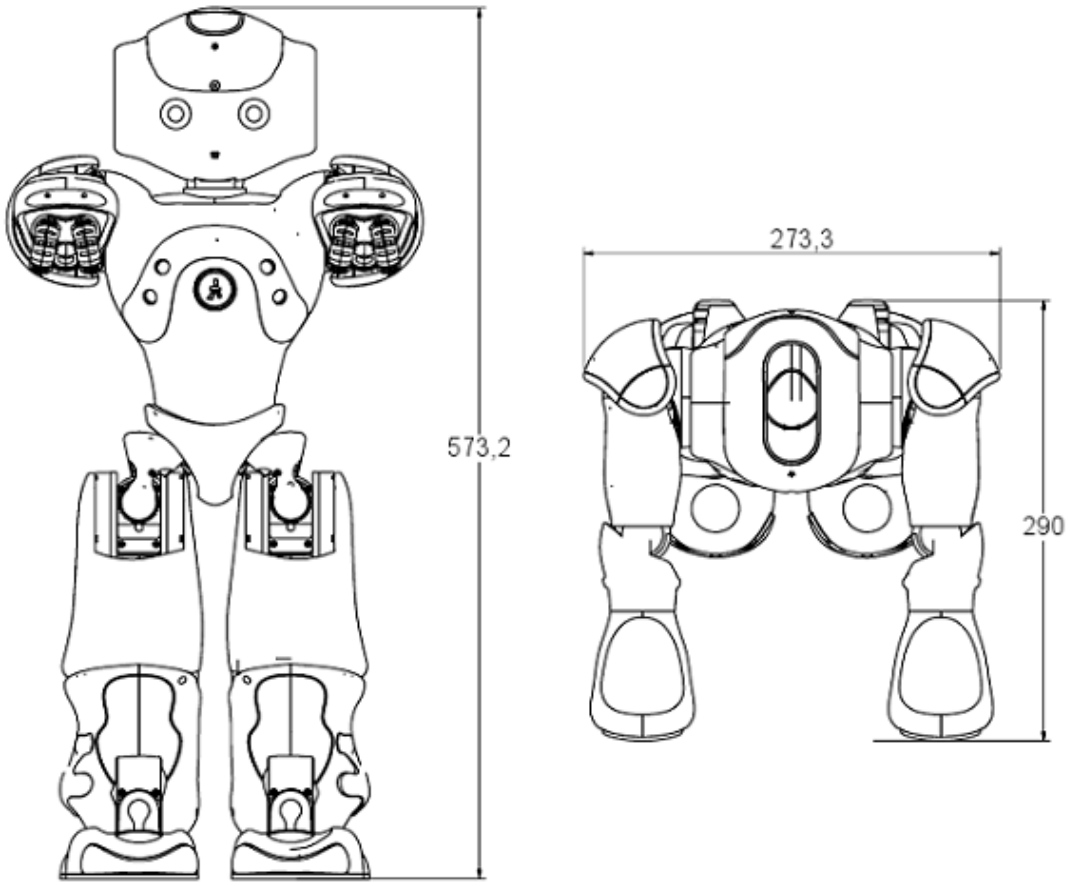


Рисунок 1.1 – Технічні характеристики NAO-робота

1.4 Визначення імітаційної моделі

Імітація – це чисельний метод виконання на обчислювальних машинах експериментів з математичними моделями, що описують поводження складних систем протягом тривалих проміжків часу. Принципова відмінність імітаційного експерименту від реального полягає в тому, що в процесі імітації експеримент проводиться не з самою системою, а з її моделлю. Доцільність застосування імітаційного моделювання роботів як в складі роботехнічних комплексів, так і окремо, визначається наступними причинами [5]:

– рішення задачі аналітичними методами або неможливо, або складно;

– крім отримання середніх значень вихідних змінних необхідно

спостереження за їх зміною в перебігу певного проміжку часу;

- за допомогою методу імітаційного моделювання можуть бути побудовані моделі, що відображають велику сукупність елементів даної системи;
- імітаційне моделювання вільно від обмежень, які мають аналітичні методи;
- на імітаційної моделі можна провести експерименти, які на реальному об'єкті по ряду причин провести неможливо;
- імітаційне моделювання дозволяє проводити довготривалі експерименти шляхом стиснення тимчасової шкали;
- результати імітаційного моделювання наочні і легко інтерпретовані;
- імітація поведінки об'єкта дає уявлення про те, які змінні системи найбільш істотні і як вони взаємодіють ще до створення самого об'єкта.

Завданням дипломної пороти є створення імітаційної моделі позиційної системи рухів Nao-робота, що дозволить здійснювати управління рухами робота.

1.5 Постановка завдання

NAO-роботи знаходяться в повсякденній розробці і поліпшуються з кожним днем, отримуючи все новий функціонал. Незважаючи на постійну розробку, звичайний користувач не має відкритого доступу до бібліотеки IDE, яка дозволяє симулювати робота. У зв'язку з цим актуальним завданням є розробка своєї імітаційної моделі позиційної системи рухів NAO-робота, яка б могла імітувати рухи робота. А відкритість коду і можливість безкоштовного доступу дозволить розробникам привносити внесок в розвиток функціональних можливостей NAO-роботів і як наслідок, впровадження отриманих результатів на фізичних моделях NAO-роботів. Розробка імітаційної моделі позиційної системі рухів NAO-робота дозволить

створити каркас і базовий функціонал робота, який з часом може доопрацьовуватися. Так, для здійснення розробки імітаційної моделі позиційної системи рухів NAO-робота необхідним є наявність наступного програмного забезпечення:

- безкоштовної операційної системи Linux;
- безкоштовні бібліотеки загального доступу Open Dynamics Engine для реалізації фізики NAO-робота;
- бібліотеки Drawstuff для реалізації графіки і відображення позиціонування NAO-робота;
- мова програмування C ++.

В Papyrus повинен бути реалізований проект, що складається з UML-діаграм, які описують взаємозв'язок класів частин робота. Використовуючи готові бібліотеки для фізики і відображення робота в просторі, необхідно реалізувати наступні функції позиційної системи рухів NAO-робота реалізований повинен буде наступний функціонал:

- нахиляти ліве коліно;
- нахиляти праве коліно;
- нахиляти лівий гомілковостопній суглоб (щиколотку);
- нахиляти правий гомілковостопній суглоб;
- підіймати лівий гомілковостопній суглоб;
- підіймати правий гомілковостопній суглоб;
- згинати лівий / правий лікоть;
- підіймати лівий лікоть;
- підіймати правий лікоть;
- згинати ліве / праве зап'ястя;
- нахиляти, повертати голову.

При виконанні того чи іншого руху роботом накладається певна система обмежень, описана в технічній документації для розробників NAO-роботів [6].

2 ВИБІР ПРОГРАМНИХ ЗАСОБІВ РОЗРОБКИ ІМІТАЦІЙНОЇ МОДЕЛІ

2.1 Загальні вимоги до реалізації моделі

Основною функцією робототехнічного стимулятора є надання різноманітних засобів і широких можливостей для моделювання, але користувачам важливо отримувати гнучкий підхід, який надасть можливість працювати з доступною мовою програмування, дозволить створювати стимулятор, який підходить для перших класів роботів.

Для здійснення розробки імітаційної моделі позиційної системи рухів NAO-робота, яка має виконувати імітацію рухів робота, необхідно здійснити вибір програмних засобів реалізації моделі, а саме обрати операційну систему, мову програмування, бібліотеки для реалізації фізики, для реалізації графіки і позиціонування NAO-робота, бібліотеку для здійснення рендеринга простих 3D-об'єктів у віртуальному середовищі, середовище розробки UML-діаграм. В ході виконання дипломного проектування, згідно з постановкою завдання були сформульовані наступні загальні вимоги до реалізації моделі:

- програмне забезпечення повинно бути представлено у вигляді UML-діаграм створених в Papyrus (IDE Eclipse);
- для реалізації фізики NAO-Робота необхідно використовувати динамічно підключаємо бібліотеку відкритого доступу Open Dynamics Engine (ODE);
- для реалізації графіки та відображення симуляції необхідно використовувати бібліотеку ODE Drawstuff;
- мова програмування повинен бути C ++;
- створюване програмне забезпечення повинно функціонувати на операційної системі Linux.

Розглянемо детальніше кожен з обраних засобів реалізації імітаційної моделі позиційної системи рухів NAO-робота та наведемо переваги

застосування кожного з обраних засобів відповідно до вирішення завдань дипломного проектування.

Розробка імітаційної моделі здійснюється з застосуванням об'єктно-орієнтованого підходу, при якому вся програма розглядається як набір взаємодіючих один з одним об'єктів.

При цьому важливо враховувати характеристики об'єктів в системі, властивості і поведінку. Такий підхід допомагає будувати складні системи більш просто і природно завдяки тому, що вся предметна область розбивається на об'єкти і кожен з них має слабкий зв'язок з іншими об'єктами. Слабка зв'язаність виникає внаслідок дотримання трьох принципів: інкапсуляції, успадкування та поліморфізму. Застосування при створені моделі об’єктно-орієнтованого підходу зволяє спростити складний об'єкт, складаючи його з дрібніших і простих, що прискорює розробку [7].

2.2 Бібліотека моделювання Open Dynamics Engine

Для розробки імітаційної моделі позиційної системи рухів NAO-робота ODE (Open Dynamics Engine) – «відкритий динамічний движок», безкоштовна, бібліотека моделювання фізики твердого тіла. Його основними компонентами є система динаміки абсолютно твердого тіла і система виявлення зіткнень.

ODE надає великої ваги швидкості і стабільності ніж фізичної точності.

Швидка, гнучка і надійна. Підходить, наприклад, для симуляції транспортних засобів, істот з ногами, і рухомих об'єктів у віртуальному оточенні. ODE повнофункціональний, стабільний «движок» який має простий у використанні C / C ++ API. Він має вдосконалені типи з'єднань і інтегроване виявлення зіткнень з тертям. В даний час він використовується в багатьох комп'ютерних іграх, інструментах створення 3D і інструментах моделювання. Бібліотека Open Dynamics Engine надає розробнику наступні можливості [8]:

- з'єднання (joints), що зв'язують тверді тіла. У бібліотеці доступні багато типів сполук перехідних, (ball and socket – шарнірне, hinge – осьовий, slider – ковзне, fixed – зафіксоване, і більш складні типи). З'єднання мають велику кількість налаштувань і методів управління;
- стабільність і фізична точність. Інтегратор в ODE стійкий, відповідно і система, стійка. В ODE використовується інтегратор Ейлера, але є можливість написати свій інтегратор для більш високої точності;
- вбудована система зіткнень (частково заснована на бібліотеці OPCODE): підтримуються наступні види геометричних форм: промінь, площина, сфера, паралелепіпед, капсула, циліндр, опуклий багатогранник, карта висот, довільна трикутна сітка. Простір обробки: sweep-and-prune, quadtree, hash і simple простору;
- добре наближена модель тертя (Coloumb friction model);
- інтерфейс ODE реалізований у вигляді набору функцій, без використання об'єктно-орієнтованої моделі. Такий підхід спрощує використання бібліотеки і мінімізує проблеми з різними компіляторами і мовами;
- підтримка ODE мається на графічній бібліотеці GLScene і в багатьох інших бібліотеках. Версія ODE, адаптована для сучасних КПК, входить до складу Airplay SDK.
- має C інтерфейс (хоча майже вся ODE написана на C ++). C ++ інтерфейс поверх C інтерфейсу;
- написано безліч юніт тестів, і багато розроблюються;
- наявні специфічні оптимізації для різних платформ.

Цих можливостей досить для того щоб реалізувати безліч фізичних ефектів, наприклад: rag dolls, cloth simulation і т.д. В даний час ODE використовується в багатьох комп'ютерних іграх, 3D інструментах та інструменти симулювання.

2.3 Програмне забезпечення для проведення рендерінгу моделі

Рендеринг – це процес створення фотореалістичного або нефореалістичного зображення з 2D або 3D моделі за допомогою комп’ютерної програми. Отримане зображення називається рендер. Кілька моделей можна визначити у файлі сцени, що містить об'єкти суворо визначеною мовою або структурою даних. Файл сцени містить геометрію, точку зору, текстуру, освітлення та інформацію про затінення, що описують віртуальну сцену. Дані, що містяться у файлі сцени, потім передаються програмі візуалізації для обробки і виводяться у файл цифрового зображення або файлу растрової графіки. Термін «рендерінг» аналогічний поняттю враження художника від сцени, використовується для опису процесу обчислення ефектів у програмі відеомонтажу для отримання кінцевого відеовиходу.

Рендерінг є однією з основних підтем 3D-комп’ютерної графіки, і на практиці вона завжди пов’язана з іншими. Це останній важливий крок у графічному конвеєрі, який надає моделям та анімації остаточного вигляду. Рендерінг використовується в архітектурі, відеоіграх, симуляторах, візуальних ефектах фільмів та візуалізації дизайну, кожен із яких використовує різний баланс функцій та технік. Доступно широке різноманіття візуалізаторів. Деякі з них інтегровані у більші пакети моделювання та анімації, інші є самостійними, а інші – безкоштовними проектами з відкритим кодом. З внутрішньої сторони візуалізатор – це ретельно розроблена програма, заснована на багатьох дисциплінах, включаючи фізику світла, зорове сприйняття, математику та розробку програмного забезпечення. Хоча технічні деталі методів візуалізації різняться, загальні проблеми, які потрібно подолати при створенні 2D-зображення на екрані із тривимірного подання, що зберігається у файлі сцени, обробляються графічним конвеєром у пристрої візуалізації, такому як графічний процесор. GPU – це спеціально розроблений пристрій, який

допомагає центральному процесору у виконанні складних обчислень візуалізації. Якщо сцена виглядає відносно реалістичною та передбачуваною при віртуальному освітленні, програмне забезпечення для рендерингу має вирішити рівняння рендерингу. Рівняння візуалізації не враховує всіх явищ освітлення, але натомість діє як загальна модель освітлення для комп'ютерних зображень [9].

У випадку 3D-графіки сцени можна попередньо відтворити або створити в режимі реального часу. Попередній рендерінг – це повільний, обчислювальний процес, який зазвичай використовується для створення фільмів, де сцени можна створювати заздалегідь, тоді як рендерінг у реальному часі часто робиться для 3D-ігор та інших програм, які повинні динамічно створювати сцени. 3D апаратні прискорювачі можуть покращити ефективність рендерінгу в реальному часі.

Створена імітаційна модель позиційної системи рухів NAO-робота має рендер, який реалізовано за допомогою бібліотеки DrawStuff, що дозволяє здійснювати рендерінг простих 3D-об'єктів у віртуальному середовищі з метою демонстрації можливостей ODE.

2.4 Інтегроване середовище розробки Eclipse

Eclipse являє собою інтегроване середовище розробки (IDE) для розробки додатків з використанням різних мов програмування Java C / C ++, Python, PERL, Ruby і т. д. Платформа Eclipse, яка забезпечує основу для Eclipse IDE, складається з модулів і призначена для розширення за допомогою додаткових модулів, може використовуватися для розробки повнофункціональних клієнтських додатків, інтегрованих середовищ розробки і інших інструментів. Eclipse може використовуватися як IDE для будь-якої мови програмування, для якого доступний плагін. Java Development Tools (JDT) надає модуль, який дозволяє використовувати Eclipse в якості Java IDE, PyDev – це модуль, який дозволяє використовувати Eclipse як

Python IDE, плагін Eclipse Scala дозволяє використовувати Eclipse як IDE для розробки додатків Scala, а PHPeclipse – це плагін для eclipse, який забезпечує повну розробку інструмент для PHP [[10].

Eclipse Platform являє собою написану на Java середу розробки самого загального призначення, архітектура якої забезпечує для вирішення різних завдань інтеграцію різних інструментів і мов програмування (рис. 2.1).

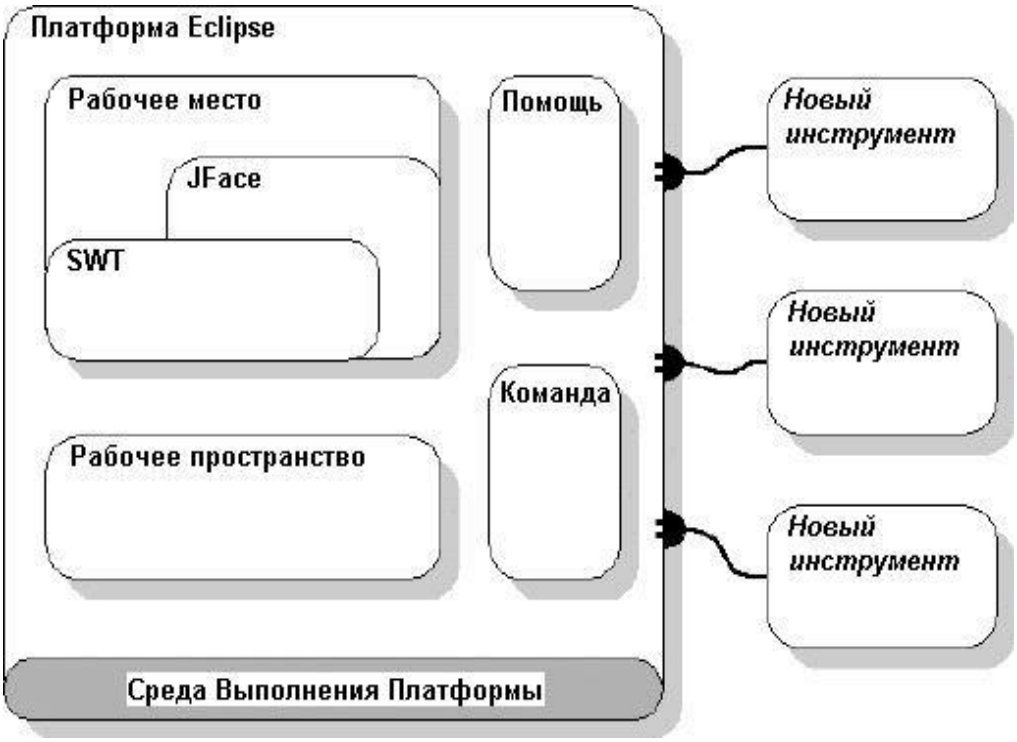


Рисунок 2.1 – Архітектура Eclipse Platform

Механізми подібної інтеграції дозволяють використовувати Eclipse Platform для побудови розвинених середовищ розробки, звільняють від рутини написання базових засобів на користь створення складних, спеціалізованих функцій. Тим самим не тільки вирішується проблема підтримки багатофункціональних і багатомовних середовищ розробки, але і закладається база для спрощення переходу від одного типу середовища до іншого в процесі їх еволюції.

Основою архітектури Eclipse Platform є принцип використання модулів

(plug-in). Як правило, за допомогою одного модуля реалізується найпростіша функціональність IDE на базі Eclipse Platform, в той час як більш складні інструментальні засоби компонуються з декількох модулів. При цьому майже вся вбудована функціональність Eclipse Platform також реалізована за допомогою таких модулів, за винятком невеликого ядра Platform Runtime. За допомогою спеціальних модулів реалізовані компоненти робочого простору (workspace) і призначеного для користувача інтерфейсу (workbench) платформи Eclipse [11].

Використання платформи розробки в відкритих кодах з гнучким механізмом модулів для реалізації додаткової функціональності відкриває необмежені можливості її вдосконалення Завданням технологічного проекту Eclipse є координація зусиль розробників, дослідників, членів наукових і освітніх організацій за визначенням перспектив Eclipse Platform і інших розробок з відкритим кодом.

Для розробки програмних модулів імітаційної моделі позиційної системи рухів NAO-робота використано пагін Eclipse для C/C++ Development Tools (CDT), який дозволяє використовувати Eclipse для розробки додатків з використанням мови програмування C ++. Мова програмування C ++ являє собою високорівневу компільовану мову програмування загального призначення зі статичної типізацією, яка підходить для створення самих різних додатків. Мова C ++ на даний час є достатньо популярною, в неї додані нові можливості, а саме класи, які роблять її не просто доповненням до Ci, а зовсім новою мовою програмування.

2.5 Мова моделювання UML

При створенні складних інженерних систем прийнято використовувати прийоми моделювання. Складність більшості створюваних сьогодні програмних систем не поступається складності багатьом інженерним спорудженням, тому моделювання програмних систем є досить актуальним

завданням. Більш того, у таких концепціях, як MDA (Model Driven Architecture – архітектура на основі моделей) і MDD (Model Driven Development – розробка на базі моделей), моделям приділяється центральна роль у процесі створення програмного продукту. Основною ідеєю цих концепцій є представлення процесу створення програмного продукту у вигляді ланцюжка трансформацій його вихідної моделі в готову програмну систему. Майже у всіх інструментальних засобах, що втілює ідеї MDD, як мова моделювання використовується мова UML – Unified Modeling Language (уніфікована мова моделювання). UML – це мова, призначена для візуалізації, специфікації, конструювання й документування програмних систем. Слово «уніфікований» у назві мови означає, що UML може використовуватися для моделювання широкого кола додатків від вбудованих систем і систем реального часу до розподілених web-додатків. Виразні засоби мови дозволяють описати систему зі всіх точок зору, що мають відношення до розробки й розгортання. Моделювання на чистому UML найближче до чисто об'єктно-орієнтованого підходу до розробки [12].

Для реалізації компонентного підходу на рівні моделювання, для UML можуть бути створені профілі, які визначають нові сутності та діаграми для моделювання компонентів. Також можливе створення DSL для моделювання компонентів, які не ґрунтуються на UML. Спостерігається тенденція, коли в інструментах моделювання ПЗ присутні два типи моделей: модель компонентів (більш абстрактна) і об'єктно-орієнтована модель (близька до коду). При прямому проектуванні для отримання коду на базі моделі компонентів шляхом трансформації компонентної моделі спочатку виходить об'єктно-орієнтована модель, а потім вже програмний код. При зворотному проектуванні з коду спочатку виходить об'єктно-орієнтована модель. В коді на основі такої моделі може бути автоматично отримана і компонентна модель. UML (Unified Modeling Language) – уніфікована мова об'єктно-орієнтованого моделювання, вікорістовується у парадігмі об'єктно-орієнтованого програмування. Є невід'ємною частиною уніфікованого

процесу розробки програмного забезпечення. UML может буті застосовано на всех етапах життєвого циклу аналізу бізнес-систем і розробки додатків. Різні види діаграм які підтримуються UML, и найбагатшій набір засобів робить UML універсальнім засоби опису як програмних, так и ділових систем. Діаграмі дають можливість представити систему у такому вигляді, щоб її можна було легко перевести в програмний код. Крім того, UML спеціально створювалася для оптимізації процесу розробки програмних систем, що дозволяє збільшити ефективність реалізації програмних систем у кілька разів. Засоби автоматичної генерації кодів дозволяють перетворювати моделі мовою UML у вихідний код об'єктно-орієнтованих мов програмування, що ще більш прискорює процес розробки. Практично усі CASE-засоби мають підтримку UML. Моделі розроблені в UML, дозволяють значний спростити процес кодування, описати об'єкт в єдиному заданому синтаксисі, тому де б не була створена діаграма, її правила будуть зрозумілі для всіх, хто знайомий з цим графічним мовою [13].

Створення UML-моделі з існуючого коду програми – реверс-інжиніринг, зворотня побудова, може застосовуватися, там де є написаний код, але документація неповна або відсутня. З моделей можна витягувати текстову інформацію і генерувати документи.

Для проектування і реалізації імітаційної моделі позиційної системи рухів NAO-робота було використано потужний інструмент моделювання UML з відкритим вихідним кодом Eclipse Papyrus.

2.6 Застосування бібліотеки Eclipse Papyrus

Профілі UML пропонують розробникам інтуїтивно зрозумілий спосіб створення предметно-орієнтованих мов моделювання шляхом повторного використання і розширення концепцій UML. Eclipse Papyrus – це потужний інструмент моделювання UML з відкритим вихідним кодом, який підтримує профілювання UML. Papyrus забезпечує графічне моделювання інструменту

UML з відкритим вихідним кодом на основі середовища Eclipse. Papyrus націлений на створення інтегрованого і зручного для користувача середовища для редагування будь-якої моделі EMF і, зокрема, підтримки UML і пов'язаних мов моделювання, таких як SysML і MARTE. Він надає діаграмні редактори для мов моделювання на основі EMF (серед них UML) і SysML. Papyrus також пропонує дуже просунуту підтримку профілів UML для визначення редакторів для DSL на основі стандарту UML. Головна особливість Papyrus – набір дуже потужних механізмів налаштування, які можуть бути використані для створення користувацьких перспектив [14].

Papyrus – це інструмент модельно-орієнтованого проектування промислового рівня з відкритим вихідним кодом. Папірус, зокрема, успішно використовується в промислових проектах і є базовою платформою для декількох інструментів промислового моделювання. Eclipse Papyrus надає редактори для всіх діаграм UML: діаграма класів, діаграма об'єкта, діаграма упаковки, схема складової структури, схема компонентів, схема розгортання, діаграма профілю, діаграма варіантів використання, діаграма діяльності, діаграма кінцевого автомата, схема зв'язку, схема послідовності, часова діаграма. Всі функції моделювання Eclipse Papyrus призначені для настройки і максимального повторного використання [15].

Отже, якщо потрібно адаптувати стандартну конфігурацію для конкретної області, нотації, практики моделювання або використовувати потужні механізми настройки Eclipse Papyrus, щоб адаптувати середу моделювання відповідно до ваших потреб. Багато конфігурації в Eclipse Papyrus засновані на моделях, тому настроювання можна виконати в реальному часі.

2.7 Компоновка файлів для компіляції за допомогою Makefile

Для компоновки файлів розробки імітаційної моделі позиційної системи рухів NAO-робота була використана утиліта Unix Make, що

призначена для запуску виконання make-файлу.

Makefile – це спеціальний файл, що містить команди оболонки, які створюються розробником і називаєте makefile (або Makefile в залежності від системи). Make відстежує час останнього оновлення файлів (зазвичай об'єктних файлів) і оновлює тільки ті файли, які необхідні, щоб підтримувати вихідний файл в актуальному стані. Якщо розробник має великий проекта з великою кількістю вихідних файлів і / або файлів заголовків, при зміні файлу, від якого залежать інші, потрібно перекомпілювати всі залежні файли. Без make-файлу це надзвичайно трудомістке завдання. Перебуваючи в каталозі, що містить цей make-файл, команди в make-файлі будуть виконані. Якщо розробник створюєте більше одного make-файлу, переконайтеся, що активним є поточний каталог в якому виконується команда make. Оскільки make-файл являє собою список команд оболонки, він повинен бути написаний для оболонки, яка буде обробляти make-файл. Makefile, який добре працює в одній оболонці, може функціонувати належним чином в інший оболонці. Makefile містить список правил. Ці правила повідомляють системі, які команди ви хочете виконати. У більшості випадків ці правила є команди для компіляції (або перекомпіляції) серії файлів. Правила, які повинні починатися в стовпці один, складаються з двох частин. Перший рядок називається рядком залежності, а наступні рядки називаються діями або командами. Рядок (і) дій повинен мати відступ з табуляцією [16].

3 МОДЕЛЮВАННЯ ПОЗИЦІЙНОЇ СИСТЕМИ РУХІВ NAO-РОБОТА

3.1 Функціональні характеристики NAO-робота

Nao – найрозвиненіша гуманоїдна платформа, що складається з 1400 деталей. Роботом Nao можливо здійснювати керування через призначену для користувача програму Choregraphe®, з програмованими модулями C ++. NAO-роботи знаходяться в повсякденній розробці і поліпшуються з кожним днем, отримуючи все новий функціонал. Але слід зазначити, що звичайний користувач не має відкритого доступу до бібліотеки IDE, яка дозволяє симулювати робота. Тому завданням є створення імітаційної моделі позиційної системи рухів NAO-робота, що забезпечить моделювання імітації рухів робота відповідно до встановлених вимог та технічних обмежень. Визначимо основні складові моделі робота, які наведені на рисунку 3.1.

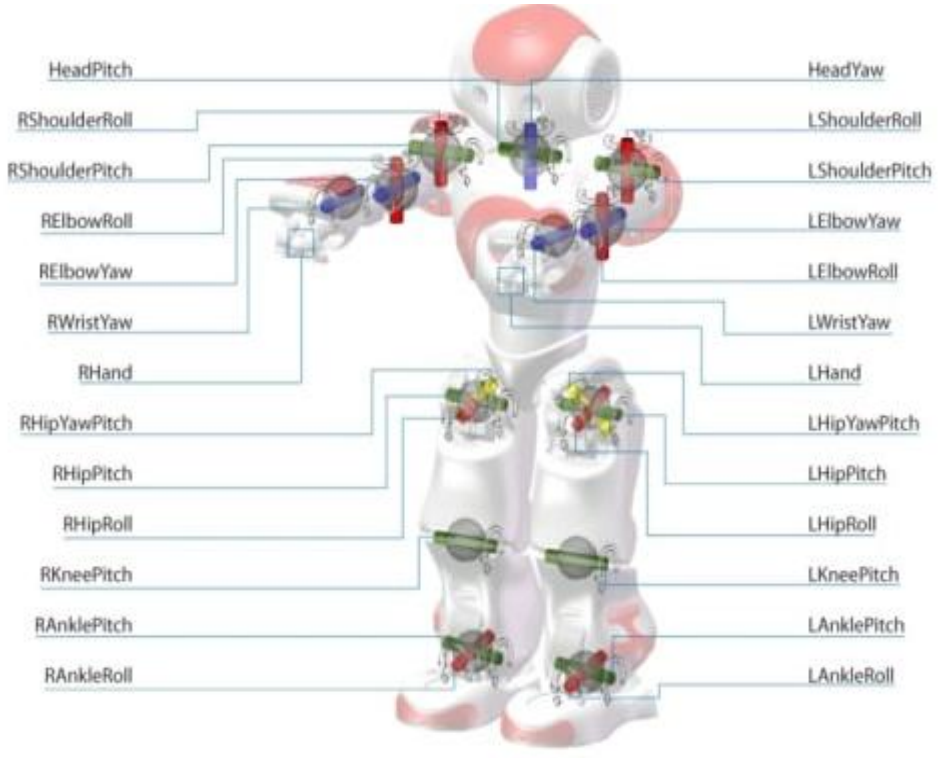


Рисунок 3.1 – Схема складових частин NAO-робота

Робот складається з двох (лівої / правої) ніг, двох (лівої / правої) рук, тулуба і голови. Користувач може дати одну з наступних команд тільки одному суглобу за раз:

– команда init;

– команда setAngle.

Визначимо функціональні вимоги до ноги робота. Нога повинна складатися з наступних частин: стегно, гомілка, стопа, колінний суглоб, з'єднаний із стегном та гомілкою, гомілковостопний суглоб, з'єднаний з гомілкою і стопою [6] (рис. 3.2).

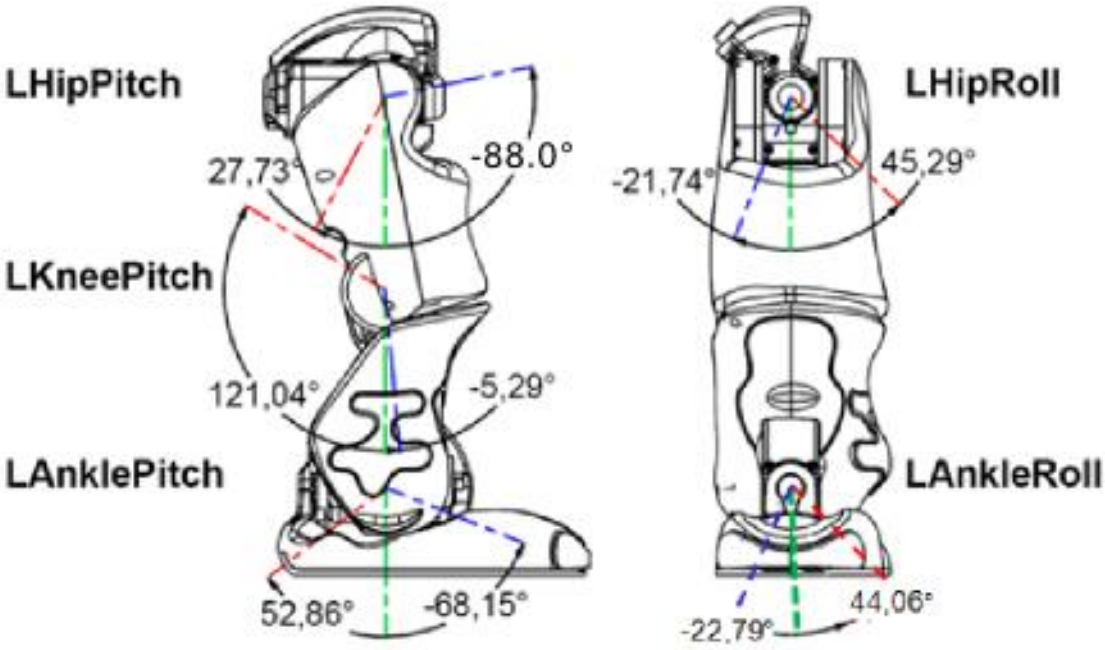


Рисунок 3.2 – Технічні характеристики колінного суглоба

Нахил колінного суглоба змінюється шляхом зміни кута по осі нахилу колінного суглоба. Щоб можна було відхилити колінний суглоб, заданий кут повинен знаходитися в припустимих межах, тому застосовується наступне обмеження:

nodAngleMin ≤ angle ≤ nodAngleMax.

Згідно специфікаціям встановлені наступні вимог та технічні

обмеження: лівий колінний суглоб (LKneeNod) nodAngleMin = -5,90° та nodAngleMax = 121,04°; правий колінний суглоб (RKneeNod) nodAngleMin = -5,90° та nodAngleMax = 121,47°.

Якщо це не так, зміна кута суглоба відбувається до nodAngleMin / nodAngleMax і потім залишається на цьому місці, так що після руху колінного суглоба завжди виконується наступне обмеження:

nodAngleMin ≤ nodAngle ≤ nodAngleMax.

Нахил гомілковостопного суглоба досягається за рахунок зміни кута по осі кроку гомілковостопного суглоба.

Щоб можна було нахилити гомілковостопний суглоб, то встановлений кут повинен знаходитися в припустимих межах, тому застосовується така умова:

nodAngleMin ≤ angle ≤ nodAngleMax

Згідно специфікаціям встановлені наступні вимог та обмеження: лівий гомілковостопний суглоб (LAnkleNod) nodAngleMin=-68,15° та nodAngleMax = 52,86°; правий гомілковостопний суглоб (RAnkleNod), nodAngleMin = -67,97° та nodAngleMax = 53,40°.

Якщо ці умови не виконуються, то зміна кута суглоба відбувається до nodAngleMin / nodAngleMax і потім залишається на цьому місці, так що після руху гомілковостопним суглобом завжди виконується наступне обмеження:

nodAngleMin ≤ nodAngle ≤ nodAngleMax

Піднімання гомілковостопного суглоба досягається зміною кута по осі крену гомілковостопного суглоба.

Щоб можна було піднімати гомілковостопний суглоб, кут що задається повинен знаходитися в припустимих межах, тому застосовується наступне обмеження:

rollAngleMin ≤ angle ≤ rollAngleMax

Згідно специфікаціям встановлені наступні вимог та обмеження: лівий гомілковостопний суглоб (LAnkleRoll) rollAngleMin= -22,79° та rollAngleMax = 44,06°; правий гомілковостопний суглоб (RAnkleRoll) rollAngleMin=-44,06°

та rollAngleMax = 22,80°.

Якщо ці умови не виконуються, то зміна кута суглоба відбувається до rollAngleMin / rollAngleMax і потім залишається на цьому місці, так що після руху гомілковостопним суглобом завжди виконується наступне:

rollAngleMin ≤ rollAngle ≤ rollAngleMax.

Таким чином здійснено визначення вимог та обмежень щодо позиціонування рухів колінним та гомілковостопним суглобами NAO-робота.

Визначимо функціональні вимоги до руки робота. Рука повинна складатися з наступних частин: надпліччя, передпліччя, лікоть, який з'єднує надпліччя та передпліччя; зап'ястя,яке з'єднує передпліччя і кисть [6].

Згинання ліктьового суглоба досягається зміною кута на осі відхилення ліктя. Щоб мати можливість згинати лікоть, кут що задається повинен знаходитися в припустимих межах, тому застосовується наступне обмеження:

yawAngleMin ≤ angle≤ yawAngleMax

Згідно специфікаціям лівий лікоть (LElbowYaw) yawAngleMin = -119.5° та yawAngleMax=-119.5°; правий лікоть (RElbowYaw) yawAngleMin = -119.5° и yawAngleMax = -119.5°.

Якщо ці умови не виконуються, зміна кута суглоба відбувається до yawAngleMin / yawAngleMax і потім залишається на цьому місці, так що після руху ліктьовим суглобом завжди виконується наступне обмеження:

yawAngleMin ≤ yawAngle ≤ yawAngleMax

Згинання ліктя досягається зміною кута по осі згинання ліктя.

Щоб мати можливість повертати лікоть, кут що задається повинен знаходитися в припустимих межах, тому застосовується наступне обмеження:

gierWinkelMin ≤ winkel ≤ rollWinkelMax.

Згідно специфікаціям лівий лікоть rollAngleMin= -88,5° та rollAngleMax = -2°; правий лікоть (RElbowRoll) rollAngleMin = 2° та rollAngleMax = 88,5°.

Якщо ці умови не виконуються, зміна кута суглоба відбувається до

rollAngleMin / rollAngleMax і потім залишається на цьому місці, так що після руху ліктьового суглобу завжди виконується наступне обмеження:

rollAngleMin ≤ rollAngle ≤ rollAngleMax

Згинання зап'ястя досягається за рахунок зміни кута по осі згинання зап'ястя [6] (рис. 3.3).

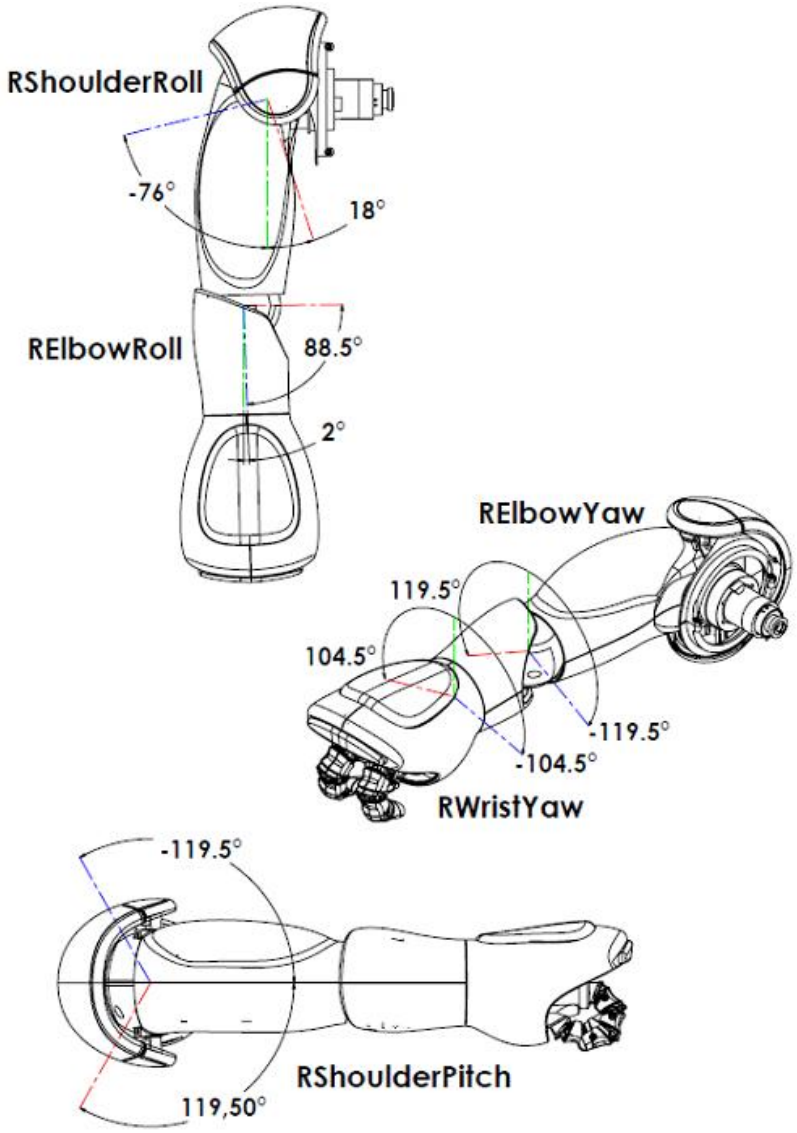


Рисунок 3.3 – Технічні характеристики суглобів руки

Щоб зап'ясті можна було повертати, кут, який повинен бути встановлений повинен перебувати в припустимих межах, тому застосовується наступне:

yawAngleMin ≤ yawAngle ≤ yawAngleMax

Згідно специфікаціям (LWristYaw) yawAngleMin=-104,5° та yawAngleMax= 104,5°; (RWristYaw) yawAngleMin=-104,5° та yawAngleMax = 104,5°).

Якщо наведені умови не виконуються, зміна кута зап'ястя відбувається до yawAngleMin / yawAngleMax і потім залишається на цьому місці, так що після руху зап'ястя завжди виконується наступне обмеження:

yawAngleMin ≤ yawAngle ≤ yawAngleMax

Також необхідним є надання функціональної можливості відкривати руку, якщо вона була закрита. (зліва LHand / праворуч RHand). Крім того повинна бути реалізована можливість закрити руку, якщо вона була відкрита. (зліва LHand / праворуч RHand).

Надалі буде розглянуто корпус робота та визначені функціональні вимоги корпусу робота, який складається з наступних частин: тулуб, таз, плечовий суглоб, який з'єднує тулуб з плечем, тазовий суглоб, який з'єднує тулуб з тазом, тазостегновий суглоб, який з'єднує таз з стегном. Нахил плечового суглоба досягається за рахунок зміни кута нахилу осі плечового суглоба. Щоб можна було відхилити плечовий суглоб, кут що задається повинен знаходитися в припустимих межах, тому можна застосувати наступне:

nodAngleMin ≤ angle ≤ nodAngleMax

Згідно специфікаціям: лівий плечовий суглоб (LShoulderNod) nodAngleMin=-119,5° та nodAngleMax=119,5°; равий плечовий суглоб (RShoulderNod) nodAngleMin=-119,5° та nodAngleMax=119,5°.

Якщо це умови не виконуються, зміна кута з'єднання до nodAngleMin / nodAngleMax і потім залишається на цьому місці, так що після нахилення плечового суглоба завжди застосовується наступне обмеження:

nodAngleMin ≤ nodAngle ≤ nodAngleMax

Кут нахилу тазового суглоба досягається за рахунок зміни кута осі. Щоб можна було нахиляти та відхиляти тазовий суглоб, використовується

ось, яка знаходиться точно під кутом 45° до осей руху суглобу. Встановлюваний кут повинен знаходитися в допустимих межах, тому застосовується наступне обмеження (рис. 3.3):

angleMin ≤ angle ≤ angleMax.

Згідно специфікаціям: лівий тазовий суглоб (LHipYawNod) angleMin = -65,62° та AngleMax=42,44°; правий тазовий суглоб (RHipYawNod) AngleMin = -65,62° та AngleMax = 42,44°.

Якщо умови не виконуються, проводиться зміна кута суглоба до AngleMin / AngleMax і потім залишається на цьому рівні, так що після нахилу тазового суглоба завжди застосовується наступне обмеження (рис. 3.2):

AngleMin ≤ Angle ≤ AngleMax

Піднімання плечового суглоба досягається зміною кута по осі крену плечового суглоба. Щоб можна було піднімати плечовий суглоб, кут що задається повинен знаходитися в припустимих межах, тому застосовується наступне обмеження (рис. 3.3):

rollAngleMin ≤ rollAngle ≤ rollAngleMax

Згідно специфікаціям лівий плечовий суглоб (LShoulderNod) rollAngleMin=-88,5° та nodAngleMax=-2°; авий плечовий суглоб (RShoulderNod) rollAngleMin = 2° та rollAngleMax = 88,5).

Якщо умови не виконано, зміна кута з'єднання становить до rollAngleMin / rollAngleMax, а потім залишається там, так що після піднімання суглоба завжди застосовується наступне обмеження (рис. 3.3):

rollAngleMin ≤ rollAngle ≤ rollAngleMax

Нахил тазового суглоба досягається зміною кута нахилу осі нахилу тазового суглоба. Щоб можна було нахилити тазовий суглоб, кут що встановлюється повинен знаходитися в припустимих межах, тому застосовується наступне обмеження:

nodAngleMin ≤ angle ≤ nodAngleMax (см. Модель)

Згідно специфікаціям: лівий тазостегновий суглоб (LHipNod)

nodAngleMin=-88° та nodAngleMax=27,73°; равий тазостегновий суглоб (RHipNod) nodAngleMin=-88° та nodAngleMax = 27,73°.

Якщо умови не виконуються, зміна кута тазостегнового суглоба становить до nodAngleMin / nodAngleMax, а потім залишається там, так що після наклону суглоба завжди застосовується наступне обмеження:

nodAngleMin ≤ nodAngle ≤ nodAngleMax.

Піднімання тазового сустава досягається зміною кута нахилу осі нахилу тазового суглобу. Щоб можна було підняти тазовий суглоб, кут що встановлюється повинен знаходитися в припустимих межах, тому застосовується наступне обмеження:

yawAngleMin ≤ angle ≤ yawAngleMax

Згідно специфікаціям: лівий тазостегновий суглоб (LHipYaw) yawAngleMin=-21.74° та yawAngleMax=-45.29°; правий тазостегновий суглоб (RHipYaw) yawAngleMin = -88° та yawAngleMax= 21.74°.

Якщо це не так, зміна кута тазостегнового суглоба становить до yawAngleMin / yawAngleMax, а потім залишається там, так що після піднімання суглоба завжди застосовується наступне обмеження:

yawAngleMin ≤ yawAngle ≤ yawAngleMax.

Наступною частиною, яка була розглянута, є голова робота. Визначемо функціональні вимоги до цієї частини, яка складається з: черепу та шиї.

Кивок шиї досягається за рахунок зміни кута осі кивання шиї.

Щоб можна було нахилити шию, кут, який повинен бути встановлений, повинен перебувати в припустимих межах, тому застосовується наступне обмеження [6] (рис. 3.4):

nodAngleMin ≤ angle ≤ nodAngleMax

Згідно специфікаціям: шия (HeadNod) nodAngleMin=-38,5° та nodAngleMax = 29,5°.

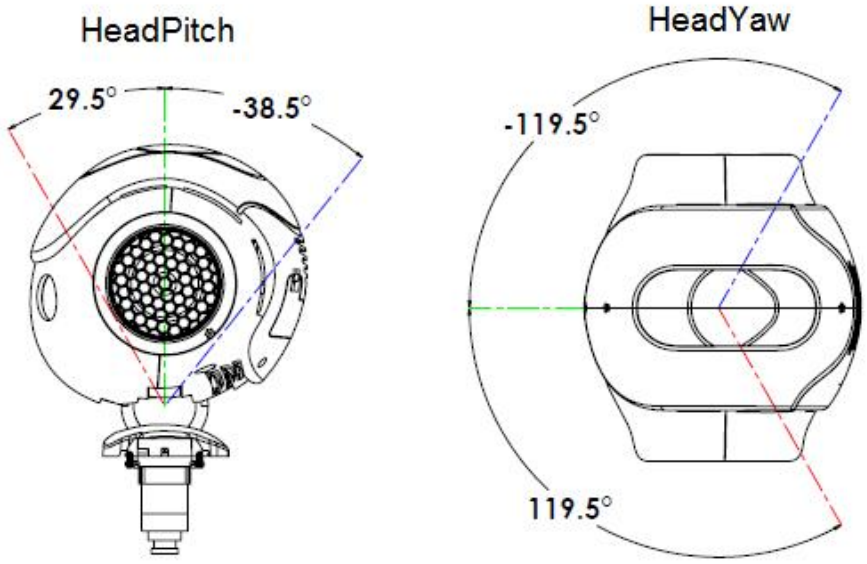


Рисунок 3.4 – Технічні характеристики голови та шиї робота

Якщо умови не виконано, кут кивка змінюється до nodAngleMin / nodAngleMax, а потім залишається там, так що після кивання шиєю завжди застосовується обмеження:

nodAngleMin ≤ nodAngle ≤ nodAngleMax

Згинання шиї досягається зміною кута на осі відхилення шиї. Для того, щоб мати можливість згинати шию, кут, який слід встановити, повинен бути в межах дозволеного діапазону, тому застосовується наступне обмеження:

yawAngleMin ≤ yawAngle≤ yawAngleMax.

Згідно специфікаціям: кут згинання шиї (HeadYaw) yawAngleMin = -119,5° и yawAngleMax=119,5°. Якщо умови не виконано, зміна кута шиї становить до yawAngleMin / yawAngleMax, а потім залишається там, так що після піднімання суглоба завжди застосовується наступне обмеження:

yawAngleMin ≤ yawAngle ≤ yawAngleMax.

Згідно з постановкою завдання, вимогами та обмеженнями імітаційна модель робота повинна схематично відображатися користувачу в тривимірному просторі. Крім того, повинна відображатися інформація про стан робота, а саме: кут нахилу кожного суглоба.

Імітаційна модель позиційної системи рухів NAO-робота повинна

відображати рухи наступних частин робота: ліва і права нога; стегна; шию; ліву, праву стопу; ліва і права руку; плече; передпліччя; корпус; торс; таз; голову.

3.2 Моделювання діаграми варіантів використання

Для розробки імітаційної моделі позиційної системи рухів NAO-робота в середовищі Papyrus було створено та реалізовано проект NaoSimulation.

Було прийнято рішення розділити проект на 2 підпакети – Modelling (моделювання) і Design (дизайн) (рис. 3.5).

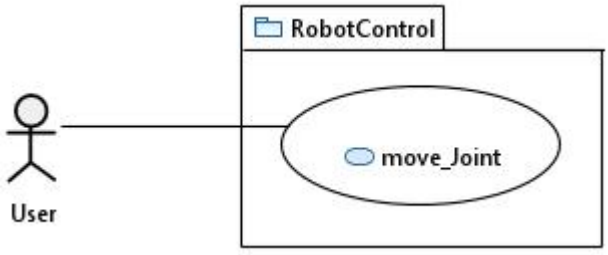


Рисунок 3.5 – Діаграма варіантів використання пакету Modelling

Пакет Modelling відповідає за моделювання NAO-робота і включає в себе частини тіла, такі як голова, руки, ноги та тулуб. Кожна частина тіла у свою чергу складається з суглобів, яки можливо рухати та частин тіла, які ці суглоби з’єднують.

Пакет Design відповідає за об’єктно-орієнтовану частину проекту, яка складається з ієрархічної структури пакетів, до яких включено: NAO-робот, його частини тіла, та взаємодії між NAO-роботом та його відображенням у просторі і фізиці NAO-робота.

Пакет Modelling містить діаграми варіантів використання, що забезпечує користувача можливістю здійснювати управління NAO-роботом. Залежно від вибору користувача, NAO-робот управляє різними суглобами.

3.3 Моделювання діаграми варіантів управління роботом

Для розробки імітаційної моделі позиційної системи рухів NAO-робота було розроблено чотири підпакета: Head (Голова), Leg (Нога), Arm (Рука), Body (Тулуб) (рис. 3.6).

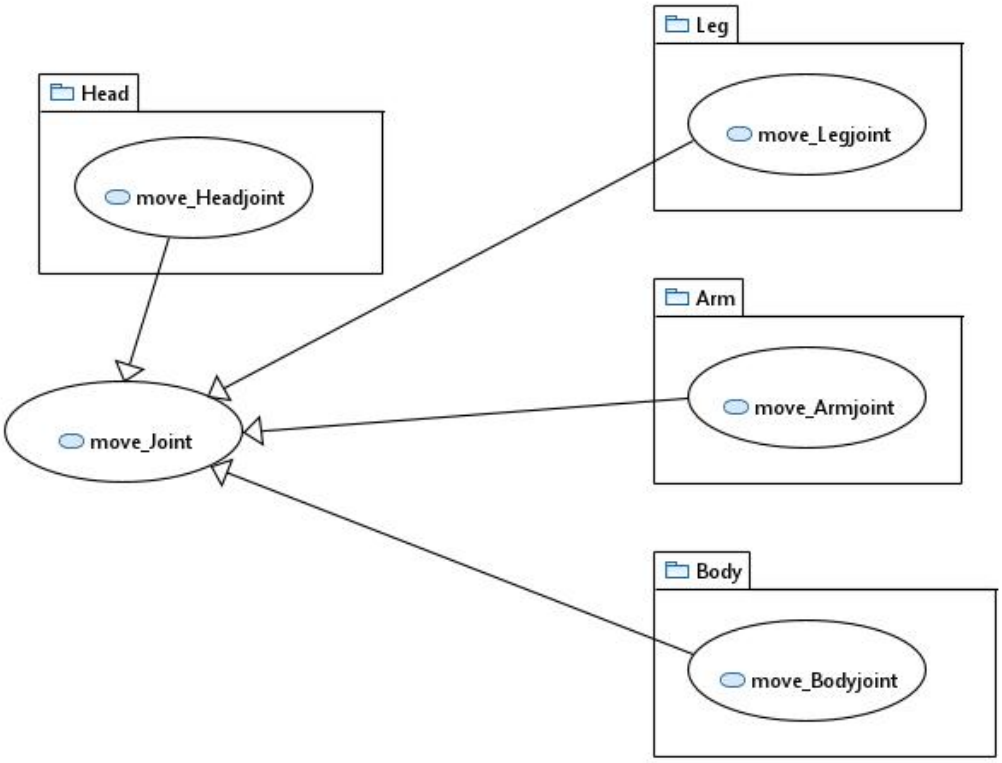


Рисунок 3.6 – Діаграма варіантів використання стимуляційної моделі

Для кожного підпакета розроблені свої класи, які відповідають за імітацію рухів суглобами, та класи, які відповідають за частини тіла робота, які забезпечують з'єднання частин тіла і суглобів.

Для кожного підпакета реалізовані потрібні функція (move_Headjoint, move_Legjoint, move_Armjoint, move_Bodyjoint), які дозволяють імітувати рух суглобів NAO-робота. Усі вони були успадковані від функції move_Joint. Щоб отримати такий результат, використано один з «Шаблонних методів» (Freeman, Baes, Sierra, Robson), відповідно до якого всі рухи суглобів

узагальнюються до функції «переміщення суглоба», або як продемонстровано у UML-діаграмі – функції move_Joint.

3.4 Моделювання схеми пакета управління роботом та діаграми класів

Для розробки імітаційної моделі системи рухів NAO-робота, щоб мати можливість зручно керувати частинами тіла робота, було розроблено чотири підпакети, в яких було застосовано принцип ієрархічної структури (рис. 3.7).

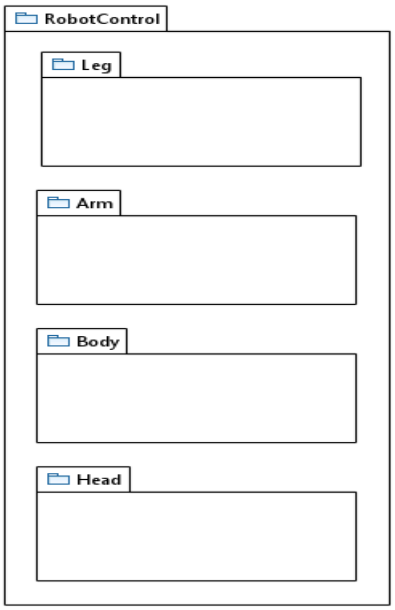


Рисунок 3.7 – Схема пакета управління роботом

На найвищому рівні в ієрархії – пакет управління роботом (RobotControl), який має ще 4 підпакети, які відповідають частинам тіла робота, а саме: Head, Leg, Arm, Body.

Для розробки імітаційної моделі позиційної системи рухів NAO-робота був створений клас робота. Руки, ноги, голова та тулуб є частинами роботами, що наведено на схемі діаграми класів крізь агрегування (рис. 3.8).

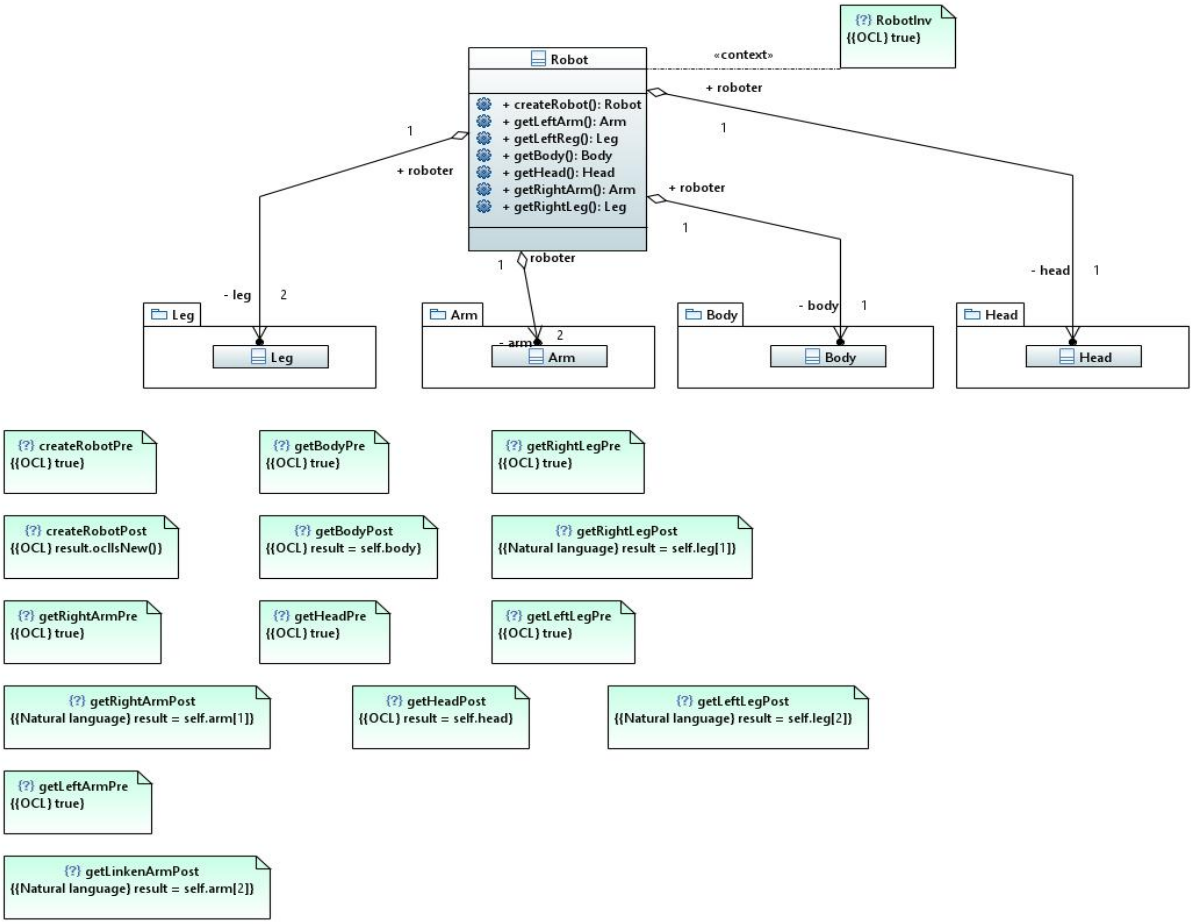


Рисунок 3.8 – Діаграма класів управління роботом

Робот складається з однієї голови, одного тулуба, двох ніг та двох рук, що наведено за допомогою потужності асоціації між класом робота та класами частин тіла. Ця схема також демонструє не тільки наявні класи, але й показує, як користувач може створювати робота та управляти ним. Класи були реалізовані шляхом агрегування з класом робот.

3.5 Моделювання діаграми варіантів використання та діаграми класів голови робота

Діаграма варіантів використання голови робота реалізує команда рухів голови, завдяки ієрархічній структурі та розширенням взаємозв'язку виникають наступні команди: «рухати шиєю» (move Neckjoint), «поворот

шиї» (yaw Neckoint) та «нахил шиї» (nod Neckjoint). Коментарі показують, як ці команди можна використовувати (рис. 3.9).

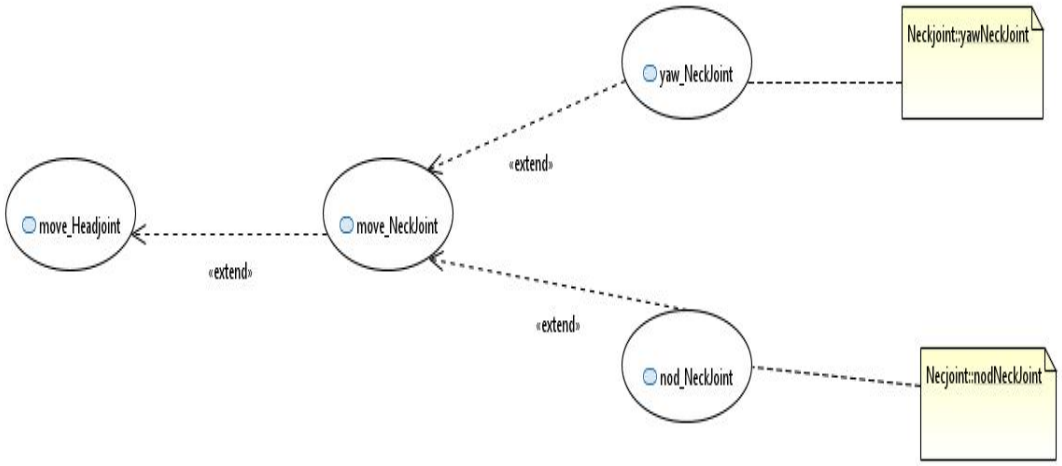


Рисунок 3.9 – Діаграма варіантів використання класу голови робота

Діаграма класів голови робота відображає клас Head (Додаток А). Голова складається з черепа і шиї і пов’язана з тулубом шиєю. Загалом, кожна частина тіла повинна мати суглоб, щоб реалізувати рух, але суглоба немає, тому весь рух здійснюється шиєю. Торс був імпортований із діаграми класу тулуба. Голова класу має єдиний метод createHead. Метод приймає за параметри шию і череп, оскільки голову можна створити лише в тому випадку, якщо вони вже створені. Завдяки карданальності можливо побачити, що голова має саме один череп і одну шию. Череп можна створити таким же чином, але для порівняння конструктор класу не приймає жодних параметрів. Щоб створити шию, череп вже повинен бути створений. Весь рух також реалізується через шию.

Всього створено шість методів, якими можна контролювати систему рухів шиї. Для кожного методу застосовані обмеження із зазначенням кутів, в яких можливо здійснювати рухи. Постумова полягає в тому, що коли новий об'єкт створюється і повертається, кожен атрибут повинен мати відповідне значення після призначення.

3.6 Моделювання діаграм пакету Design

3.6.1 Діаграма пакетів NaoModel

При розробці імітаційної моделі позиційної системи рухів NAO-робота одна з основних взаємодій проекту є взаємодія між підпакетами Modelling і Design (рис. 3.10).

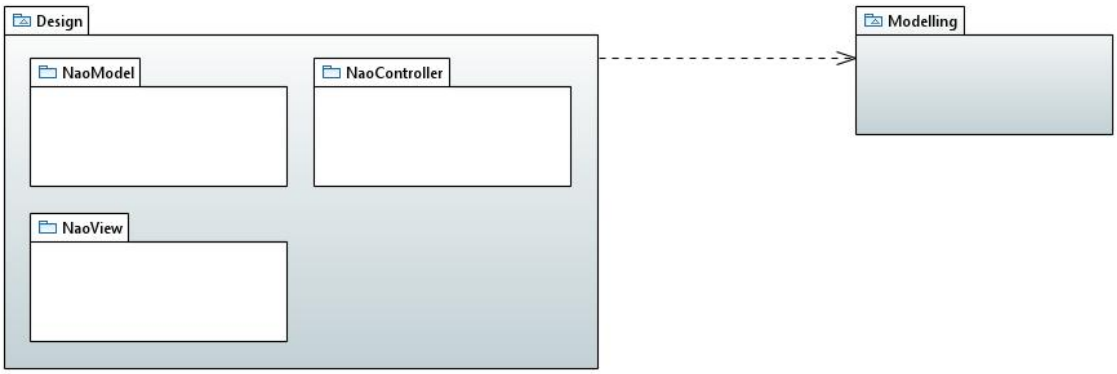


Рисунок 3.10 – Діаграма пакетів Design

Пакет Design має три підсистеми: NaoModel (Модель NAO-робота), NaoView (відображення NAO-робота) та NaoController (контролювання NAO-робота). Кожен пакет отримає з пакету моделювання дані, які необхідні для вірного відображення і взаємодії користувача із NAO-роботом. Взаємозв’язок пакету Modelling та пакету Design продемонстровано на наведеній діаграмі та відображає залежність пакету моделювання. Всі дані необхідні для пакету Design будуть імпортовані с пакету Modelling.

Для розробки імітаційної моделі позиційної системи рухів NAO-робота була реалізована спеціалізована підсистема NaoModel. Розглянемо реалізовану у дипломній роботі діаграму пакетів NaoModel. Діаграма NaoModel має чотири підпакети: частини тіла (BodyParts), суглоби (Joints), ODE та світ World (рис. 3.11).

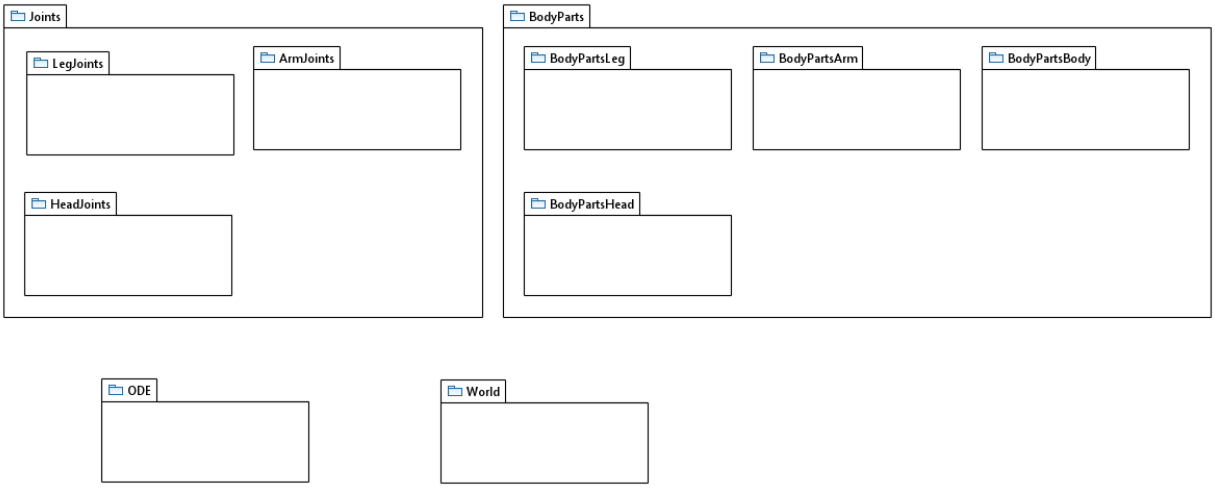


Рисунок 3.11 – Діаграма пакетів NaoModel

Розроблено пакети частин тіла (BodyParts) та пакети суглобів (Joints), які мають на два окремих підпакета, тому що у розробленій системі імітації суглоби не повинні відображатися бібліотекою Drawstuff, яка збережена та відображена у пакеті світу World, а бачити у імітації лише частини тіла NAO-робота, такі як голова, рука чи нога. Фізика була реалізована бібліотекою ODE (Open Dynamics Engine).

Підпакет світу World використовується для створення ефекту навколишнього оточення NAO-робота, у якому розроблена модель робота і буде відображатися та відповідного відображення NAO-робота у просторі.

3.6.2 Діаграма пакетів NaoView

Для розробки імітаційної моделі позиційної системи рухів NAO-робота також потрібна схема, яка б відповідала за користувальницький інтерфейс. Таким чином, для кожної частини тіла розроблено окремий підпакет, у якому зберігається інформація відповідно до частини тіла, які треба відображати у просторі (рис. 3.12).

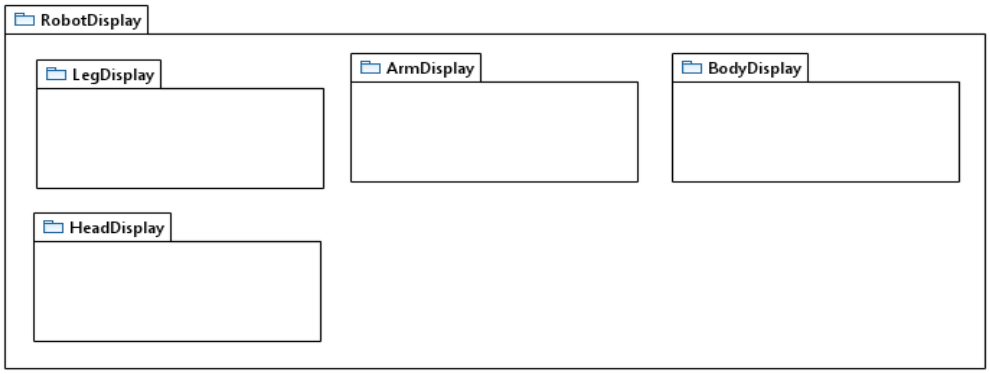


Рисунок 3.12 – Діаграма пакетів NaoView

На найвищому рівні в ієрархії – пакет управління роботом RobotDisplay, який включає у собі ще чотири підпакети, які відповідають частинам тіла робота, асаме: Head, Leg, Arm, Body. Частини тіла відображаються у просторі за допомогою бібліотеки Drawstuff та реалізованого пакету NaoView.

3.6.3 Діаграма пакетів NaoController

Для розробки імітаційної моделі позиційної системи рухів NAO-робота була розроблена схема, яка містить усі можливі команди, за допомогою яких роботом можна управляти, залежно від обраних частин робота. (рис. 3.13).

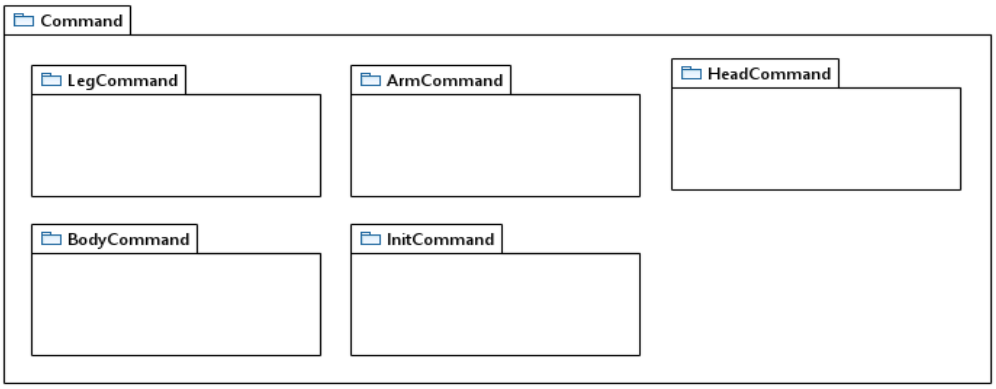


Рисунок 3.13 – Діаграма пакетів NaoController

Крім того, існує також підпакет InitCommand, який використовується для створення частин тіла, щоб можна було розпочати моделювання. Крім нього для кожної частини тіла розроблено відповідний підпакет.

3.7 Моделювання діаграм пакету Modelling

3.7.1 Діаграма пакету Head

Для розробки імітаційної моделі позиційної системи рухів NAO-робота у пакеті моделювання був розроблений пакет RobotControl. Один з наявних в ньому пакетів є пакет Head, котрий відповідає за те, щоб користувач міг імітувати рух головою NAO-робота у просторі, який буде відображатися за допомогою бібліотек ODE та Drawstuff. Пакет Head складається лише з одного класу NeckJoint або суглоба, який відповідає за рух шиєю. За специфікацією шия – це подвійний шарнірний суглоб, клас якого вспадкували від класу подвійного шарнірного суглоба (рис. 3.14).

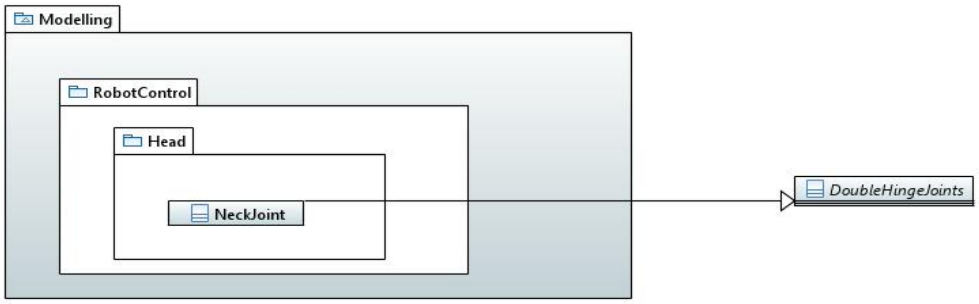


Рисунок 3.14 – Діаграма пакетів Head

Для розробки імітаційної моделі позиційної системи рухів NAO-робота створено схему класів, в якій реалізовані всі команди, які можуть керувати головою. Усі команди, які відповідають за рух головою, успадковуються від JointCommand. Рух голови реалізований за рахунок нахиляння та повернення суглоба, які міститься у шиї (рис. 3.15).

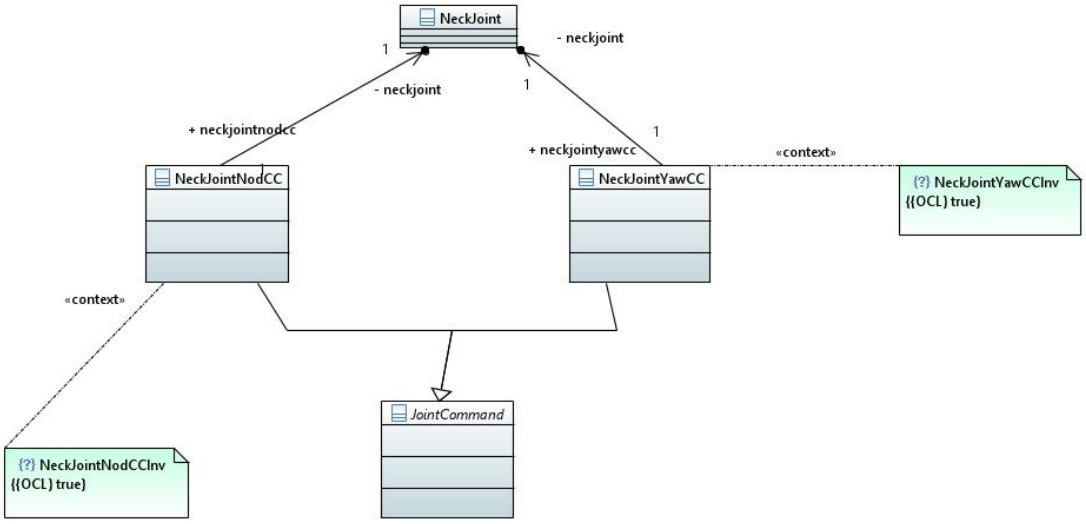


Рисунок 3.15 – Діаграма класів «Команди голови»

Отже, два методи виконання та встановлення параметрів повинні бути належним чином переписані для кожного успадкованого класу. Шия є подвійним шарнірним з'єднанням і тому може здійснювати два типи рухів: кивати і нахилятися.

3.7.2 Діаграма класів NaoController

Для розробки імітаційної моделі позиційної системи рухів NAO-робота була реалізована діаграма класів взаємодії API (інтерфейс прикладного програмування) та пакету команд користувача по управлінню роботом (Command). Пакет команд включає у себе клас управління суглобами (JointCommand) та інтерфейсу «команда» (Command) від якого успадковані всі класи управління суглобами.

Клас управління суглобами та інтерфейс команди зв'язані с класом інтерфейсу прикладного програмування за допомогою спільної асоціації (рис. 3.16).

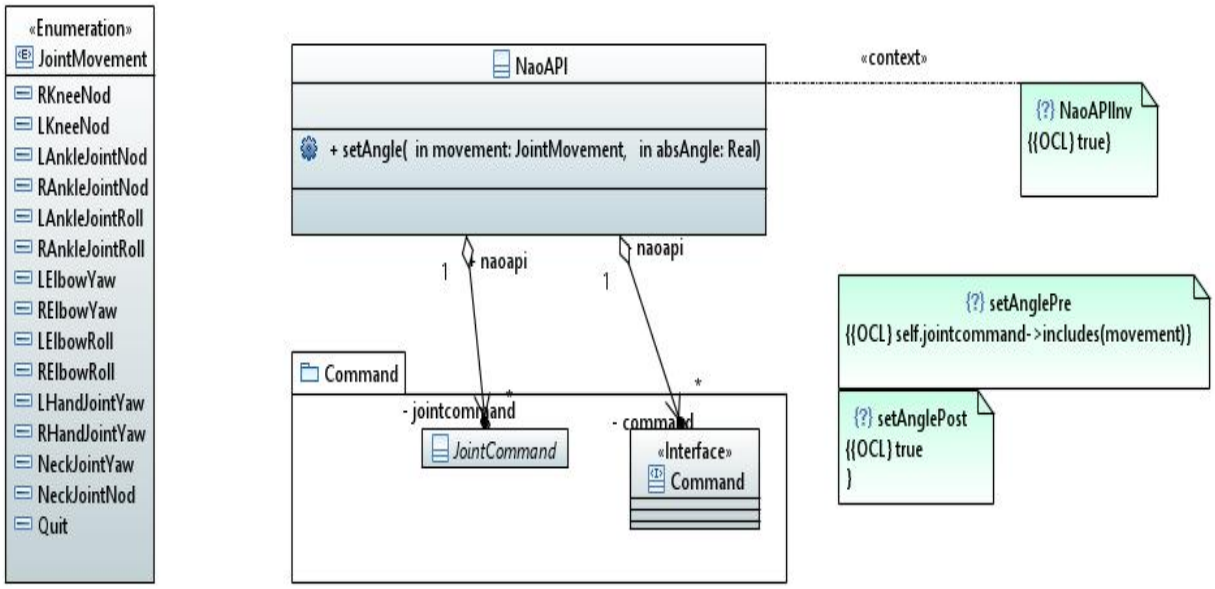


Рисунок 3.16 – Діаграма класів NaoController

Взаємозв'язок спільної асоціації (також відомий як агрегація) є більш слабкою формою відносин асоціації частин, описаних вище. Зв'язок "Спільна асоціація" відображається у вигляді стрілки, де хвіст прикріплений до елемента деталі, а білий алмазний наконечник стріли прикріплений до всього елемента компонента. Відносини спільної асоціації виявляють слабку семантику власності, якщо ціла частина буде видалена, всі частини, що належать цілій частині, не будуть видалені.

3.7.3 Діаграма класів NaoView

Для розробки імітаційної моделі позиційної системи рухів NAO-робота була реалізована діаграма класів, яка відповідаю за відображення робота у просторі (рис. 3.17).

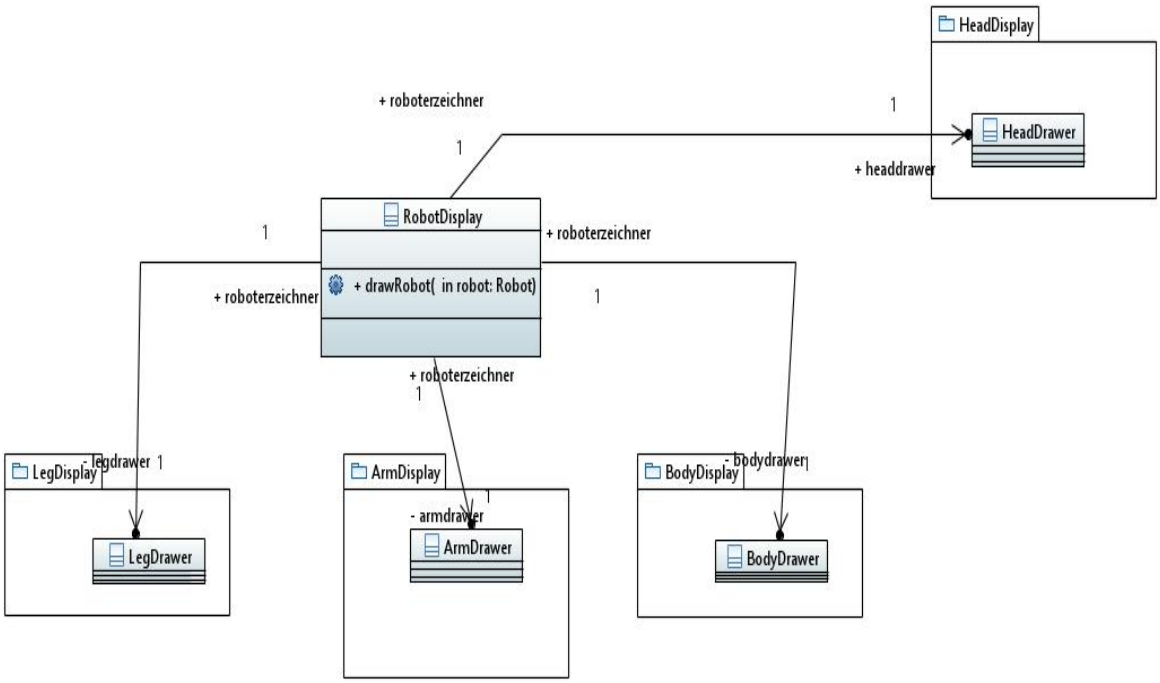


Рисунок 3.17 – Діаграма класів NaoView

Від класу «відображення роботу» (RobotDisplay) сходять спрямовані асоціації до відповідних класів у пакетах для відображення відповідних частин тіла. Потужність асоціації дає інформацію щодо кількості таких класів, і, як показано на схемі, кожний пакет має саме один такий клас, який відповідний за відображення NAO-робота у просторі.

3.7.4 Діаграма класів Freedom Degree

Для розробки імітаційної моделі позиційної системи рухів NAO-робота одним з головних класів є клас «Кут свободи» (freedom_degree) який так чи інакше впливає на весь процес рухів робота. Для кожного класу суглобів робота, який має методи руху NAO-робота, встановлені допустимі межі, у яких цей суглоб має можливість рухатися. Так, у схемі показано, що усі суглоби асоційовані до класу «кут свободи» і усі методи,які відповідають за рух NAO-робота мають залежність від значення, яке має «кут свободи» відповідної функції (рис. 3.18).

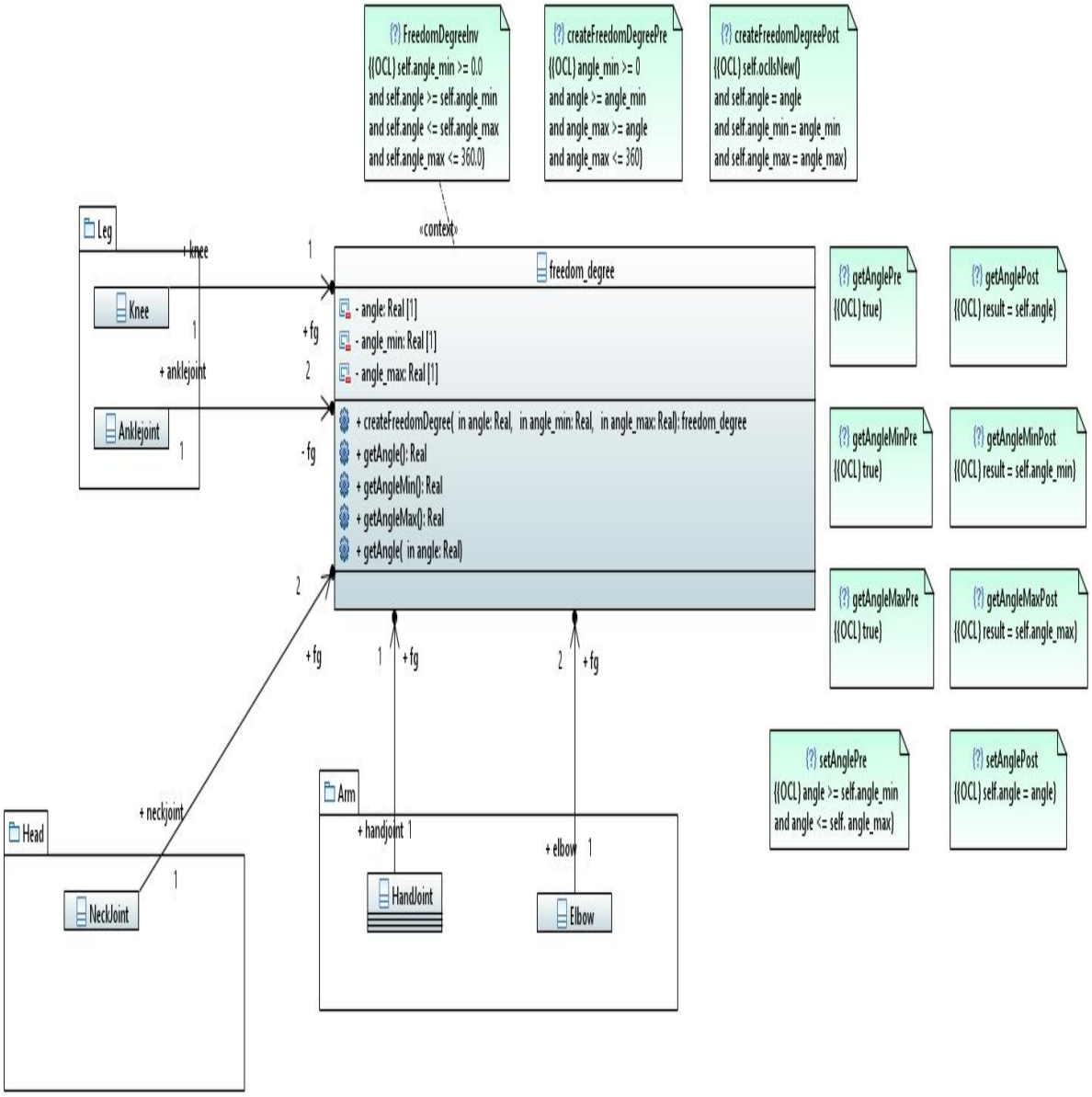


Рисунок 3.18 – Діаграма класів Freedom Degree

Таким чином при виконанні дипломної роботи проведено моделювання позиційної системи рухів NAO-робота. Створена імітаційна модель надасть користувачу можливість управляти роботом за допомогою команд. Команди описують зміну кутів шарнірів робота. Для програмного забезпечення операція setAngle (суглоб, кут, кутова швидкість) моделюється з модуля руху Nao, який змінює кут даного суглоба на заданий кут зміни. Параметр кутової швидкості, який доданий у модель, виконує функцію сумісності. Операція init виконує ініціалізацію систему.

Розроблена модель позиційної системи рухів NAO-робота відповідає вимогам сформульованим у постановки завдання з врахуванням вимог та обмежень: налаштування кутів модельованого робота ніколи не повинні приймати значення за межами припустимого діапазону, крім того слід дотримуватися обмежень щодо запобігання зіткнень.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ІМІТАЦІЙНОЇ МОДЕЛІ

4.1 Симуляція позиційної системи рухів робота у просторі ODE

Для здійснення програмної реалізації і відображення дії імітаційної моделі позиційної системи рухів NAO-робота з застосуванням мови C++ та за допомогою бібліотек ODE і Drawstuff була розроблена система, яка може симулювати рухи NAO-робота, а саме виконувати наступні дії: нахиляти ліве коліно; нахиляти праве коліно; нахиляти лівий гомілковостопній суглоб (щиколотку); нахиляти правий гомілковостопній суглоб; підіймати лівий гомілковостопній суглоб; підіймати правий гомілковостопній суглоб; згинати лівий лікоть; згинати правий лікоть; підіймати лівий лікоть; підіймати правий лікоть; згинати ліве зап'ястя; згинати праве зап'ястя; нахиляти голову; повертати голову.

Для здійснення розробки імітаційної моделі позиційної системи рухів NAO-робота було використано наступне програмне забезпечення: ОС Linux; безкоштовні бібліотеки загального доступу Open Dynamics Engine для реалізації фізики NAO-робота та бібліотека Drawstuff для реалізації графіки і відображення позиціонування NAO-робота; мова програмування C ++.

Для того, щоб запустити програму, потрібно за допомогою команди make у консолі Linux викликати MakeFile в відповідній папці, яка у проекті має ім’я naoSimCode. Після компіляції у консолі потрібно ввести команду ./naoSimTest, яка запустить симуляцію. Після цього відкриється вікно, у якому можливо побачити симуляцію NAO-робота. Через вікно консолі і відповідні команди, можливо імітувати рухи NAO-робота.

При вдалому старті проекту, перше що побачить користувач це NAO-робота у «нульовий» позиції. Кожна з частин тіла має свої задані специфікацією властивості та характеристики, які реалізовані у програмному коді та продемонстровані на рисунку 4.1.

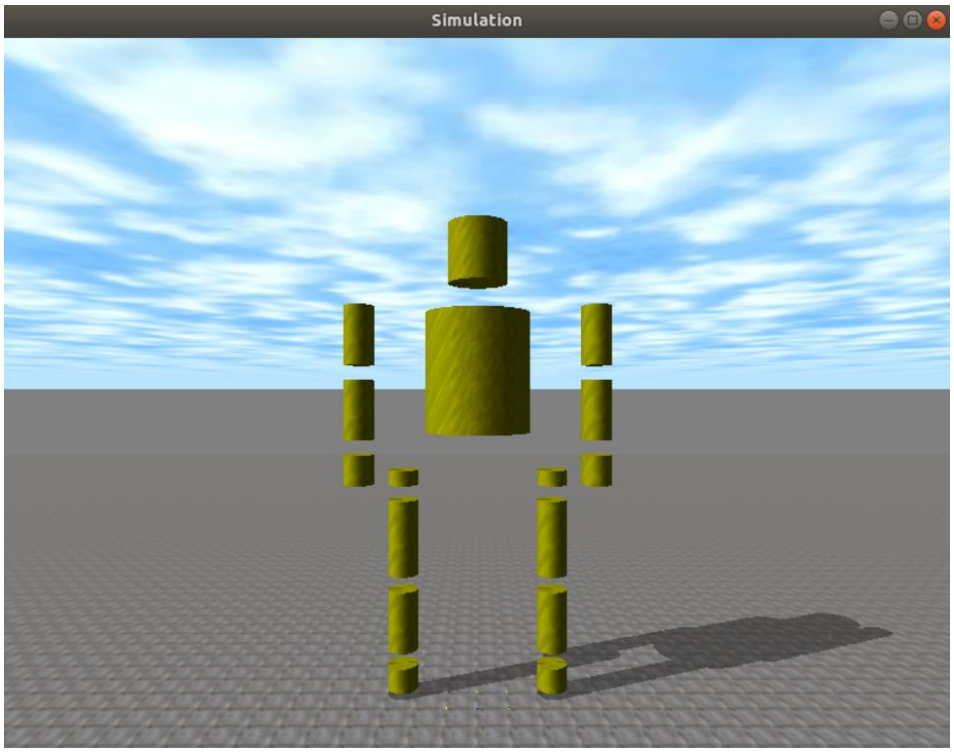
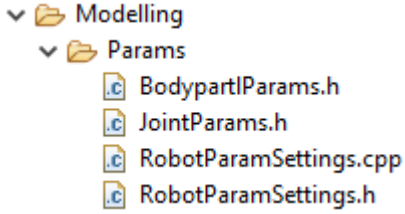


Рисунок 4.1 – Імітаційна модель NAO-робота у «нульовий» позиції

4.2 Реалізацію пакету Modelling

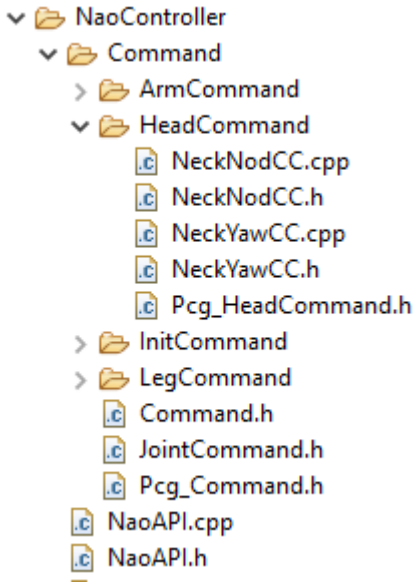
Для розробки імітаційної моделі позиційної системи рухів NAO-робота також був створений пакет Моделювання, у якому була збережена інформація щодо фізичних властивостей частин NAO-робота, таких як позиція суглоба або частина тіла у просторі, сторона, у яку рухається суглоб, швидкість нахилу суглоба та мінімальний і максимальний доступний кут нахилу.

У класі RoboterParamSetting для кожного суглоба та частини тіла була збережена вся вище перелічена інформація.



4.3 Реалізація пакету Design

Для розробки імітаційної моделі позиційної системи рухів NAO-робота був створений проект на мові C++ який дозволяє симулювати рух NAO-робота засобами пакету NaoController у Eclipse. Одним с найважливіх підпакетів для цього існує підпакет NaoController та Command. У цих двох пакетах було розроблено та реалізовано код, який дозволяє завдяки бібліотеки ODE викликати відповідні методи, які і здійснюють рух суглобів NAO-робота.



Розглянемо програмну реалізацію нахилу лівого коліна імітаційної моделі NAO-робота. За специфікацією Aldebaran, методу, який відповідає за нахил лівого коліна, були присвоєні наступні характеристики, які відповідають за: позицію суглоба у просторі; за те, у яку сторону буде рухатися суглоб, за швидкість нахилу суглоба та за мінімальний і максимальний доступний кут нахилу.

```
JointParams leftKneeRollParams = {
    // Position
    {-2.5, 0, 3.75},
    // Direction
    {nod[0], nod[1], nod[2]},
    // max. Speed
```

```
    1.0,
    // max. Angle
    2.112528,
    // min. Angle
    -0.092346
};
```

Після імітації руху моделі робота, а саме виконання команди для нахилу лівого коліна отримаємо наступний вигляд моделі (рис. 4.2).

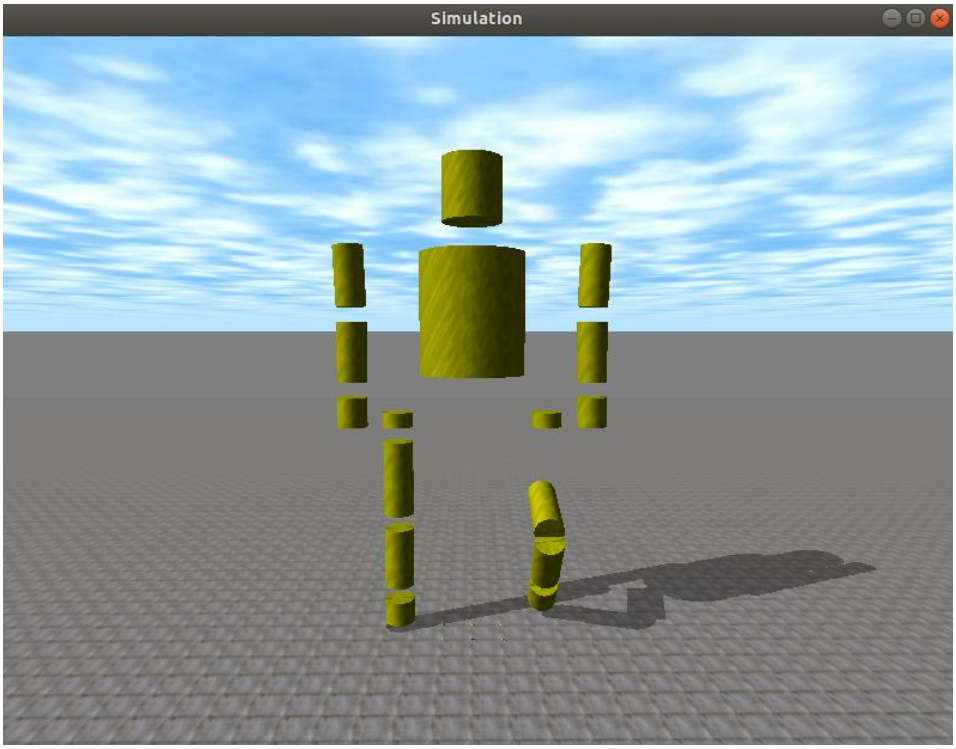


Рисунок 4.2 – Імітація виконання нахилу лівого коліна NAO-робота

Розглянемо програмну реалізацію нахилу правого коліна імітаційної моделі NAO-робота. За специфікацією Aldebaran, методу, який відповідає за нахил правого коліна, були присвоєні наступні характеристики, які відповідають за: позицію суглоба у просторі; за те, у яку сторону буде рухатися суглоб, за швидкість нахилу суглоба та за мінімальний і максимальний доступний кут нахилу.

```
JointParams rightKneeRollParams= {
```

```
        // Position
        {2.5, 0, 3.75},
        // Direction
        {nod[0], nod[1], nod[2]},
        // max. Speed
        1.0,
        // max. Angle
        2.120198,
        // min. Angle
        -0.103083
};
```

Після імітації руху моделі робота, а саме виконання команди для нахилу правого коліна отримаємо наступний вигляд моделі (рис. 4.3).

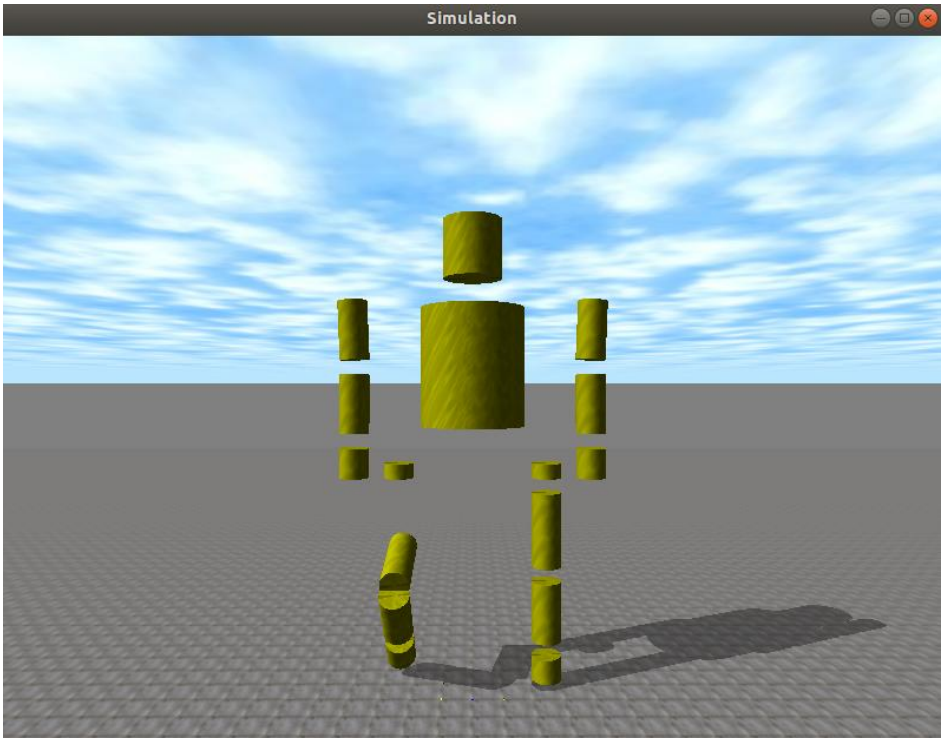


Рисунок 4.3 – Імітація виконання нахилу правого коліна NAO-робота

Розглянемо програмну реалізацію нахилу лівого гомілковостопного суглобу імітаційної моделі NAO-робота. За специфікацією Aldebaran, методу, який відповідає за нахил лівого гомілковостопного суглобу, були присвоєні наступні характеристики, які відповідають за: позицію суглоба у просторі; за те, у яку сторону буде рухатися суглоб, за швидкість нахилу суглоба та за

мінімальний і максимальний доступний кут нахилу.

```
JointParams lefAnkleJointNodParams = {
    // Position
    {-2.5, 0, 1.125},
    // Direction
    {nod[0], nod[1], nod[2]},
    // max. Speed
    1.0,
    // max. Angle
    0.922747,
    // min. Angle
    -1.189516
};
```

Після імітації руху моделі робота, а саме виконання команди для нахилу лівого гомілковостопного суглобу отримаємо наступний вигляд моделі (рис. 4.4).

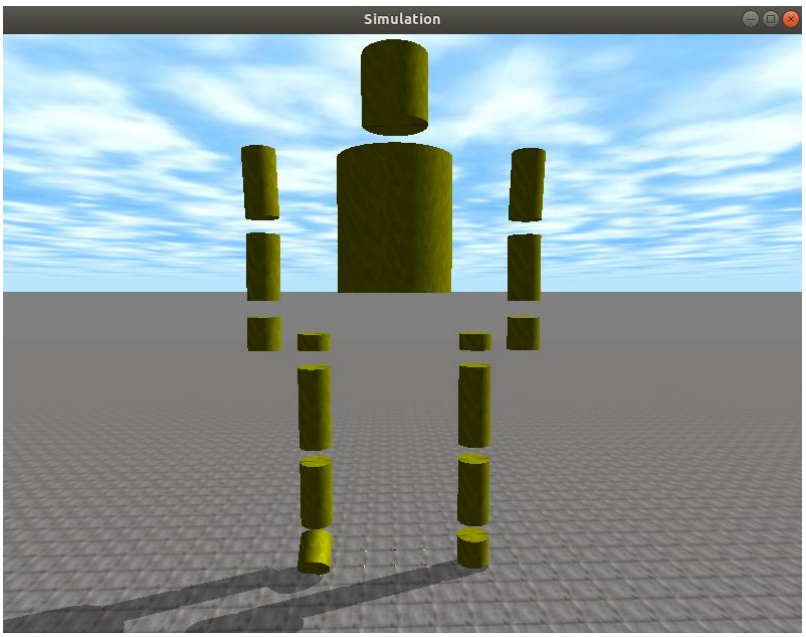


Рисунок 4.4 – Імітація нахилу лівого гомілковостопного NAO-робота

Розглянемо програмну реалізацію нахилу правого гомілковостопного суглобу імітаційної моделі NAO-робота. За специфікацією Aldebaran, методу, який відповідає за нахил лівого гомілковостопного суглобу, були присвоєні

наступні характеристики, які відповідають за: позицію суглоба у просторі; за те, у яку сторону буде рухатися суглоб, за швидкість нахилу суглоба та за мінімальний і максимальний доступний кут нахилу.

```
JointParams rightAnkleJointNodParams = {
    // Position
    {2.5, 0, 1.125},
    // Direction
    {nod[0], nod[1], nod[2]},
    // max. SPeed
    1.0,
    // max. Angle
    0.932056,
    // min. Angle
    -1.186448
};
```

Після імітації руху моделі робота, а саме виконання команди для нахилу правого гомілковостопного суглобу отримаємо вигляд (рис. 4.5).

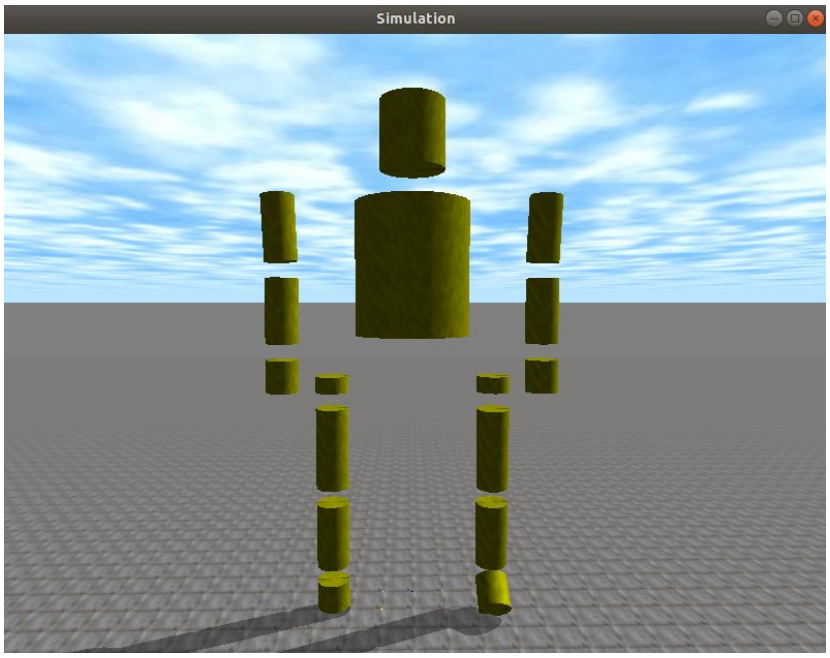


Рисунок 4.5 – Імітація нахилу правого гомілковостопного суглобу

Розглянемо програмну реалізацію нахилу голови імітаційної моделі NAO-робота. За вимогами специфікації, методу, який відповідає за нахил

голови, має характеристики: позицію суглоба голови у просторі; за те, у яку сторону буде здійснено нахил суглобу голови, за швидкість нахилу суглоба голови та за мінімальний і максимальний доступний кут нахилу.

```
JointParams neckNodParams = {
    // Position
    {0.0, 0, 10.0},
    // Direction
    {nod[0], nod[1], nod[2]},
    // max. Speed
    1.0,
    // max. Angle
    0.5149,
    // min. Angle
    -0.6720
};
```

Імітації руху нахилу голови моделі робота має вигляд (рис. 4.6).

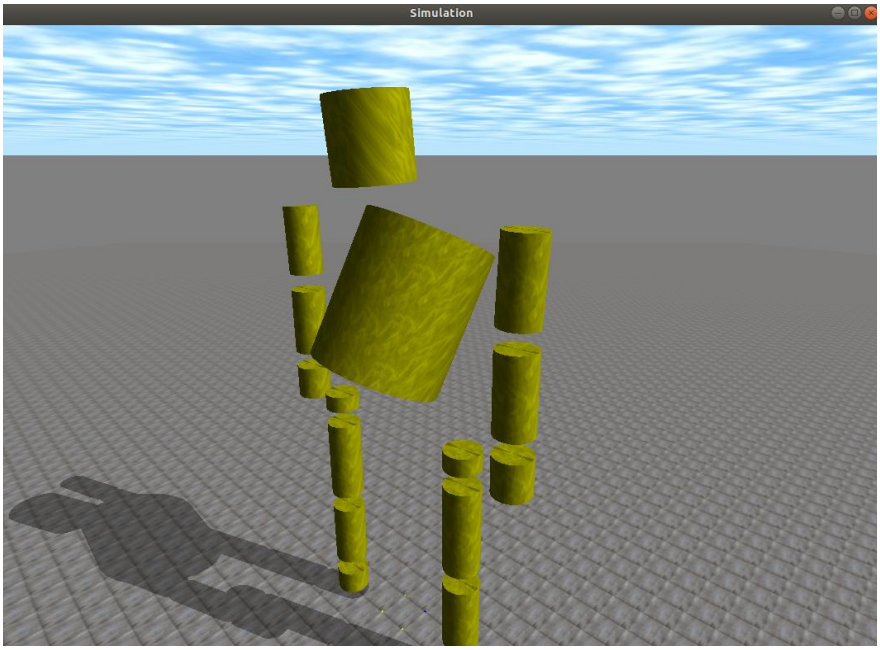


Рисунок 4.6 – Імітація нахилу голови NAO-робота

Таким чином, у дипломній роботі була створена реалізація рухів імітаційної моделі NAO-робота. Продемонстровано як частини та суглоби тіла пов'язані між собою, та виконана симуляція рухів NAO-робота.

ВИСНОВКИ

В результаті виконання дипломної роботи було здійснено проектування та програмну реалізацію імітаційної моделі позиційної системи рухів NAO-робота, що реалізує можливість симуляції рухів у просторі. Розробка імітаційної моделі дозволяє кожному, не маючи фізичної моделі NAO-робота, мати можливість експериментувати з NAO-роботом та створювати нові функції, що покращує чи доповнює вже існуючий функціонал.

В ході дипломного проектування була побудована повноцінна UML-модель, яка описує поведінку NAO-робота та взаємозв'язок його частин тіла, суглобів та простору, у якому відбувається симуляція. Технічні параметри та система обмежень рухів кожного з суглобів робота виконані за умовами специфікації та офіційної документації Aldebaran Robotics, яка є виробником NAO-робота.

Програмна реалізація імітаційної моделі здійснювалась в операційної системи Linux з використанням середовища розробки Eclipse, інструмента модельно-орієнтованого проектування Papyrus, бібліотеки ODE для реалізації фізики NAO-робота та бібліотеки Drawstuff для відображення робота у просторі.

Перевагами використання розробленої імітаційної моделі є відкритість коду і можливість безкоштовного доступу до програмного забезпечення, що дозволить розробникам привносити внесок в розвиток функціональних можливостей NAO-роботів, і як наслідок, впровадження отриманих результатів на фізичних моделях NAO-роботів. Розробка імітаційної моделі позиційної системі рухів NAO-робота дозволила створити каркас і базовий функціонал робота, який з часом може доопрацьовуватися, що прискорить процес створення роботів та їх розповсюдження у повсякденному житті людей.

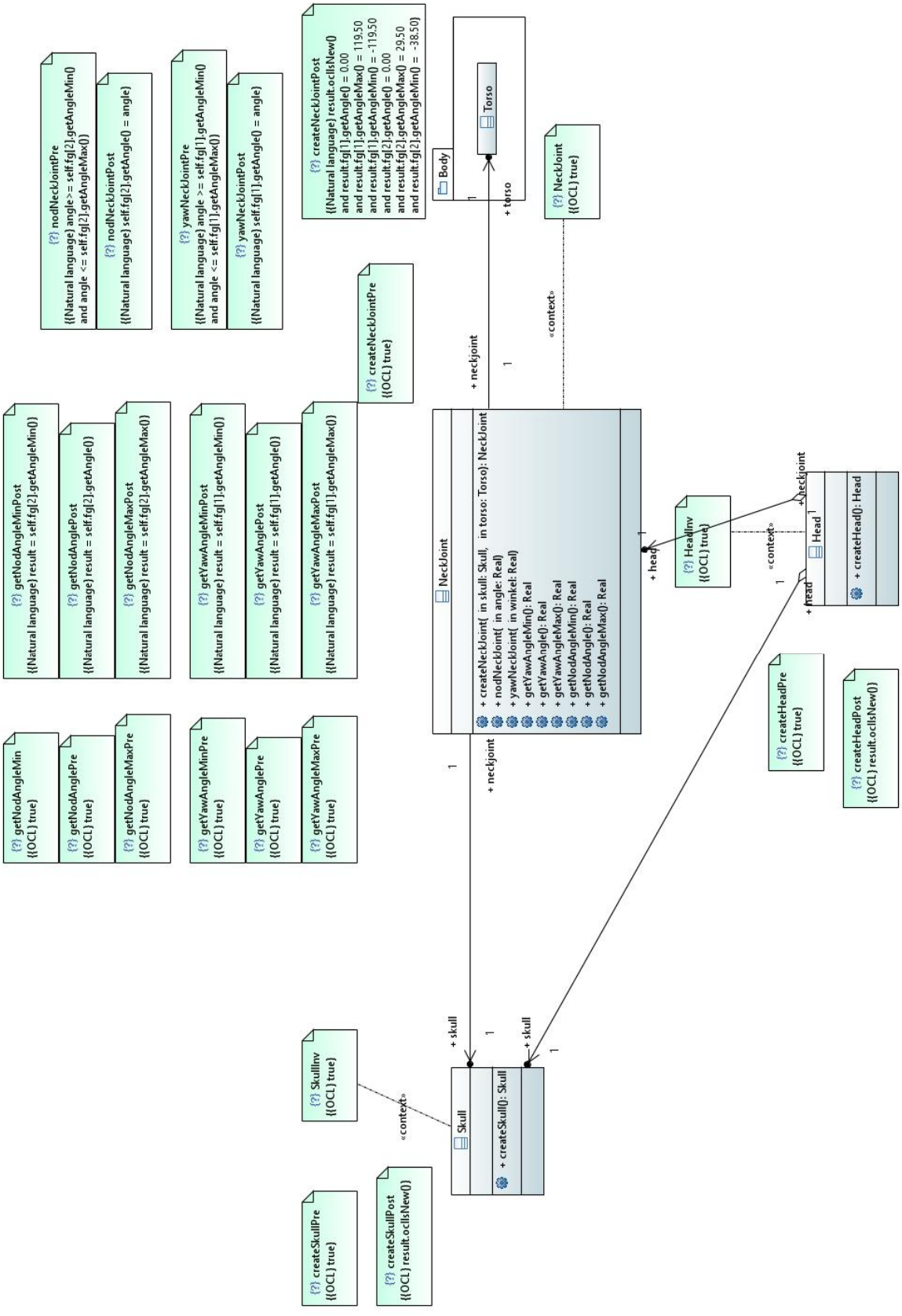
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. NAO (робот) – Википедия. URL: https://wikinew.wiki/wiki/Nao_(robot) (дата звернення 06.04.2021).
2. Aldebaran Robotics SOLIDWORKS. Использование решений SOLIDWORKS для создания инновационных роботов, помогающих людям. URL: https://www. solidworks.com/ru/story/aldebaran-robotics (дата звернення 12.04.2021).
3. Роботы наступают. Обзор индустрии робототехники от Руслана Синицкого | AIN.UA URL: https://ain.ua/2014/04/14/roboty-nastupayut-obzor-industrii-robototexniki/ (дата звернення 08.04.2021).
4. Будущее за роботами: 11 трендов развития робототехники в ближайшие годы. URL: https://trends.rbc.ru/trends/innovation/ 5d6feaba9a79 479e9bfce47e (дата звернення 14.04.2021).
5. В.Б. Неруш, В.В. Курдеча Імітаційне моделювання систем та процесів: Електронне навчальне видання. Конспект лекцій. К.: НН ІТС НТУУ «КПІ», 2012. 115 с. URL: https://ela.kpi.ua /bitstream /123456789/15598/1/Konspect_lekciy_Imit_modelyr_syst_process%28CHANGED%29.pdf (дата звернення 01.05.2021)
6. Aldebaran Documentation: Joints – Aldebaran 2.1.4.13 documentation. URL: http://doc.aldebaran.com/2-1/family/robots/joints_robot.html (дата звернення 16.04.2021).
7. Лафоре Р. Объектно-ориентированное программирование в C++. 4-е издание. СПб., Питер, 2004, 928с.
8. ODE: Open Dynamics Engine /Физика / Термины / Программирование игр / GameDev.ru URL: https://gamedev.ru/code /terms/ ODE#opisanie (дата звернення 07.04.2021).
9. Библиотека drawstuff. URL: http://opende.sourceforge.net/docs/group _drawstuff.html (дата звернення 11.04.2021).

10. Машнин Т.С. Eclipse: разработка RCP-, Web-, Ajax- и Android-приложений на Java». СПб., БХВ-Петербург, 2013, 384с.
11. Наталья Дубова Платформа разработки Eclipse | Открытые системы. СУБД | Изд-во «Открытые системы» Открытые системы. СУБД. 2005. – №03 URL: https://www.osp.ru/os/2005/03/185394 (дата звернення 21.04.2021).
12. Александр Леоненков. Самоучитель UML, 2-е издание. Изд-во: БХВ-Петербург, 2004, ISBN: 5-94157-342-1 , 589с.
13. Уніфікована мова моделювання UML | Портал знань, портал знаний, дистанційне навчання URL: http://www.znannya.org/?view=uml (дата звернення 05.05.2021).
14. Папирус URL: https://www.eclipse.org/papyrus/ (дата звернення 07.04.2021).
15. Технический обзор Платформы Eclipse © IBM Corporation 2001, 2003 URL: http://khpi-iip.mipk.kharkiv.edu/library/extent/prog/ETO/ETO.html (дата звернення 18.04.2021).
16. Компоновка файлов для компиляции с помощью makefile URL: http://www.sis.pitt.edu/mbsclass/tutorial/advanced/makefile/whatis.htm (дата звернення 11.04.2021).

ДОДАТОК А

Діаграма класів пакету Head



ДОДАТОК Б

Реалізація класу NAO-робота пакету Modelling

```
// Code generated by Papyrus C++
#include "Body/Body.h"
#include "Head/Head.h"
#define NaoSimulation_Modelling_RobotControl_Robot_BODY

/*********************************************************
Robot class body
*********************************************************/

// include associated header file
#include "Robot.h"


namespace NaoSimulation {
namespace Modelling {
namespace RobotControl {

Robot::Robot(Robot& other) : leg(), arm(), body(), head()
{
leg[0] = new
::NaoSimulation::Modelling::RobotControl::Leg::Leg(*(other.getLe
ftLeg()));
leg[1] = new
::NaoSimulation::Modelling::RobotControl::Leg::Leg(*other.getRig
htLeg());
arm[0] = new
::NaoSimulation::Modelling::RobotControl::Arm::Arm(*other.getLef
tArm());
arm[1] = new
::NaoSimulation::Modelling::RobotControl::Arm::Arm(*other.getRig
htArm());
body = new
::NaoSimulation::Modelling::RobotControl::Body::Body(*other.getB
ody());
  head = new
::NaoSimulation::Modelling::RobotControl::Head::Head(*other.getH
ead());
}
Robot::Robot(::NaoSimulation::Modelling::RobotControl::Leg::Leg*
leftLeg,
  ::NaoSimulation::Modelling::RobotControl::Leg::Leg* rightLeg,
  ::NaoSimulation::Modelling::RobotControl::Arm::Arm* leftArm,
  ::NaoSimulation::Modelling::RobotControl::Arm::Arm* rightArm,
  ::NaoSimulation::Modelling::RobotControl::Body::Body*
initBody,
  ::NaoSimulation::Modelling::RobotControl::Head::Head*
initHead) {
leg[0] = leftLeg; leg[1] = rightLeg;
```

```
arm[0] = leftArm; arm[1] = rightArm;
body=initBody; head=initHead;
}


Robot::~Robot() {
  // Delete left/right Leg
  delete leg[0];
  delete leg[1];
  // Delete left/right Arm
  delete arm[0];
  delete arm[1];
  // Delete Body
  delete body;
  // Delete Head
  delete head;
}

Robot* Robot::createRobot(
  ::NaoSimulation::Modelling::RobotControl::Leg::Leg* leftLeg,
  ::NaoSimulation::Modelling::RobotControl::Leg::Leg* rightLeg,
  ::NaoSimulation::Modelling::RobotControl::Arm::Arm* leftArm,
  ::NaoSimulation::Modelling::RobotControl::Arm::Arm* rightArm,
  ::NaoSimulation::Modelling::RobotControl::Body::Body* body,
  ::NaoSimulation::Modelling::RobotControl::Head::Head* head,
  NaoSimulation::Entwurf::NaoModel::ODE::dWorldID world
) {
  // Create Robot from his Parts
  Leg::Leg*
lLeg=::NaoSimulation::Modelling::RobotControl::Leg::Leg::createL
eftLeg(nullptr, nullptr, nullptr, nullptr, nullptr, world);
  Leg::Leg*
rLeg=::NaoSimulation::Modelling::RobotControl::Leg::Leg::createR
ightLeg(nullptr, nullptr, nullptr, nullptr, nullptr, world);
  Arm::Arm*
lArm=::NaoSimulation::Modelling::RobotControl::Arm::Arm::createL
eftArm(nullptr, nullptr, nullptr, nullptr, nullptr, world);
  Arm::Arm*
rArm=::NaoSimulation::Modelling::RobotControl::Arm::Arm::createR
ightArm(nullptr, nullptr, nullptr, nullptr, nullptr, world);

  Robot* robot = new Robot(
    lLeg,
    rLeg,
    lArm,
    rArm,

body=::NaoSimulation::Modelling::RobotControl::Body::Body::creat
eBody(rArm,lArm,rLeg,lLeg,world),

head=::NaoSimulation::Modelling::RobotControl::Head::Head::creat
eHead(nullptr,body->getTorso(), nullptr, world)
  );
```

```
  return robot;
}

::NaoSimulation::Modelling::RobotControl::Leg::Leg*
Robot::getLeftLeg() {
  return leg[0];
}

::NaoSimulation::Modelling::RobotControl::Leg::Leg*
Robot::getRightLeg() {
  return leg[1];
}

::NaoSimulation::Modelling::RobotControl::Arm::Arm*
Robot::getLeftArm() {
  return arm[0];
}

::NaoSimulation::Modelling::RobotControl::Arm::Arm*
Robot::getRightArm() {
  return arm[1];
}

::NaoSimulation::Modelling::RobotControl::Body::Body*
Robot::getBody() {
  return body;

}
::NaoSimulation::Modelling::RobotControl::Head::Head*
Robot::getHead() {
  return head;
}

bool Robot::inMovement() {
  return leg[0]->inMovement() || leg[1]->inMovement() || arm[0]-
>inMovement() || arm[1]->inMovement() || head->inMovement();
}

} // of namespace RobotControl
} // of namespace Modelling
} // of namespace NaoSimulation
```