ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА
(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра математичного забезпечення комп'ютерних систем
(повна назва кафедри (предметної, циклової комісії))

Кваліфікаційна робота

на здобуття ступеня вищої освіти «бакалавр»
(освітньо-кваліфікаційний рівень)

на тему Мобільний автономний пристрій для тестування алгоритмів одночасної локалізації і картографування
Mobile autonomous device to test the simultaneous localization and mapping algorithms

Виконав: студент денної форми навчання
спеціальності 123 – Комп'ютерна інженерія.
(шифр і назва напряму підготовки, спеціальності)
Освітня програма «Комп'ютерна інженерія»
(назва освітньої програми)
Шнайдер Антон Сергійович
(прізвище, ім'я, по-батькові)

Керівник к.ф.-м.н. Антоненко О.С.
(науковий ступінь, вчене звання, прізвище та ініціали, **підпис**)

Рецензент к.ф.-м.н., доц. Петрушина Т.І.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
№ 12 від « 09 » 06 2023 р.

Завідувач кафедри
_____ _Євгеній МАЛАХОВ_
(підпис) (ім'я, прізвище)

Захищено на засіданні ЕК № ___
протокол № __ від «___» ____ 2023 р.
Оцінка _____/_____/_____
(за національною шкалою, шкалою ECTS, бали)

Голова ЕК
_____ _Алла КОБОЗЄВА_
(підпис) (ім'я, прізвище)

Одеса - 2023

АНОТАЦІЯ

Ця кваліфікаційна робота присвячена проектуванню та побудові мобільного автономного пристрою для тестування алгоритмів одночасної локалізації та картографування.

Застосоване сучасне апаратне забезпечення, таке як одноплатний комп’ютер NVidia Jetson Nano, а також сенсори, зокрема лідар RPLidar A1 та відео камера IMX219-160. Під час побудови мобільного автономного пристрою, були враховані вимоги сучасних алгоритмів та зручність його використання під час проведення тестів у реальному середовищі. Робот має колісну платформу диференціального типу. Оглянутий процес калібрування двигунів робота. Налаштоване програмна платформа пристрою та засоби взаємодії з ним, які включають можливість віддаленого керування роботом, запуску алгоритмів та візуалізації даних з камер і лідару на комп’ютері користувача в режимі реального часу з використанням фреймворку ROS та бездротової мережі Wi-Fi. Робот протестований на алгоритмі одночасної локалізації та картографування GMapping. Побудовані мапи двох різних за розміром фізичних приміщень. Проаналізовані результати отриманих мап.

Результатом кваліфікаційної роботи є побудований мобільний автономний пристрій, який дозволяє тестувати алгоритмі одночасної локалізації та картографування.

ABSTRACT

This qualification work is devoted to the design and construction of a mobile autonomous device for testing simultaneous localization and mapping algorithms.

Modern, such as the NVidia Jetson Nano single-board computer, as well as sensors, including the RPLidar A1 lidar and the IMX219-160 video camera, are used. During the construction of a mobile autonomous device, the requirements of modern algorithms and the convenience of its usage during tests in a real environment were taken into account. The robot has a differential-type wheeled platform. The process of robot engine calibration is reviewed. Set up software platform of the device and ways of interaction with it, which include the ability to remotely control the robot, run algorithms and visualize data from cameras and lidar on the user's computer in real time using the ROS framework and a wireless Wi-Fi network. The robot is tested on the simultaneous localization and mapping algorithm GMapping. Constructed maps of two different physical buildings. The results of the obtained maps were analyzed.

The result of the qualification work is a built mobile autonomous device that allows testing algorithms of simultaneous localization and mapping.

ЗМІСТ

Стор.

ВСТУП

Обчислювальні ресурси, такі як центральні процеси, оперативна пам’ять, мікрочіпи, різноманітні сенсори та модулі - щороку стають більш доступними у фінансовому сенсі. Навіть не беручи до уваги кризи напівпровідників, соціальні та економічні потрясіння на рівні окремих країн та різноманітні геополітичні суперечки та війни, які призводять до розриву глобальних ланцюгів поставок сировини, компонентів і засобів виробництва, між країнами необхідних для такого високотехнологічного процесу як виготовлення мікрочіпів, ми можемо погодитись, що електронні компоненти стають дедалі дешевше, порівняно навіть із тим, що було десять років тому.

Слід зазначити, що не всі компоненти та деталі стають однаково доступнішими у відсотковому співвідношенні, адже важливу роль також грає попит на ті чи інші мікрочіпи і деталі, частота їх застосування у комерційних та побутових пристроях, але заперечувати загальну тенденцію спадання цін на товари електронної промисловості важко.

Результатами, або інколи одночасно і причинами такої тенденції, стає той факт, що у побуті та на різноманітних підприємствах та установах все більше впроваджується електрифікація та автоматизація різноманітних процесів. Електротехнічні рішення постійно інтегруються наше життя. Паралельно із доступністю таких елементів, зменшується також і фізичний розмір чіпів, водночас збільшується кількість транзисторів та їх щільність на квадратну одиницю відповідних чіпів. Хоча зараз вже досить складно полягятись, на так званий, Закон Мура, який являє собою емпіричне спостереження, щодо подвоєння кількості транзисторів на кристалі мікросхеми кожні двадцять чотири місяці (тобто кожні два роки), адже людство починає досягати деяких лімітів у можливості зменшення розмірів транзисторів (технологічний процес напівпровідникового виробництва), задля збільшення їх кількості. Причинами цього, , коли мова йде про розмір транзистора у сім, п’ять, три нанометри, є в тому числі, фізичні властивості та поведінка транзисторів яка стає менш контрольованою, менш стабільною і

передбачуваною на тому рівні, який би задовільняв потреби такого складного виробу як процесор. Тим не менш, розмір сучасних чіпів та впровадження відносно нових архітектур побудови процесорів, також призводить до дуже важливої характеристики, а саме зменшення електроенергії, яку споживають відповідні елементи.

Це дозволяє пристроям, задача яких полягає у функціюванні окремо від постійного джерела струму, тобто з використанням батарей, акумуляторів — джерел живлення, які є обмеженими у часі роботи, виконувати свої задачі довший проміжок часу і більш ефективно. Відповідно, ми маємо змогу спостерігати, як з'являється на ринку все більше автономних та мобільних продуктів та рішень у сфері електроніки.

Усі вищеописані явища, призводять також до того, що сильний поштовх у розвитку зазнала і робототехніка, особливо невеликі за розміром та капіталом компанії дуже просунулись у створенні різноманітних рішень для різних сфер використання: починаючи із роботів пилососів, і закінчуючи автономними платформами, які допомагають вести облік на великих складах і підприємствах, або виконувати розвідувальні та оглядові задачі у місцях, які є небезпечними, для виконання цих обов'язків людиною. Здебільшого, це можливо, завдяки більш дешевим чіпам та сенсорам.

Одним із актуальних напрямків робототехніки на сьогоднішній день є мобільні автономні роботи. Серед прикладів можна навести самокеровані автомобілі, роботи які застосовуються на складах та інших приміщеннях, щоб спростити працю людини, і навіть роботи пилососи, є також пристроєм, який можна віднести до цієї категорії. Проте для цих роботів також необхідно розробляти програмне забезпечення. Особливо те, що стосується їх навігації та пересування. Такі алгоритми називаються алгоритмами для одночасної локалізації та картографування, які дозволяють визначати положення роботу та будувати мапу, для подальшої роботи. Але одним із дуже важливих кроків після розробки таких алгоритмів, є їх тестування у реальному середовищі. Робити це на реальному продукті, не завжди є вигідним і безпечним,

особливо, якщо це безпілотний автомобіль. Тому виникає необхідність у мобільних автономних пристроях саме для тестування алгоритмів одночасної локалізації та картографування.

На сьогоднішній день існує певна кількість подібних рішень, але всі вони мають свої недоліки і не є повністю підходящими на думку автора. Тому метою цієї роботи є проектування та побудова мобільного автономного пристрою. Його засовуванням є тестування алгоритмів одночасної локалізації та картографування. Для досягнення цієї мети, необхідно виконати наступні задачі, а саме:

- зробити огляд популярних алгоритмів одночасної локалізації та картографування, описати принципи їх роботи;
- порівняти між собою доступні на ринку сенсори та обчислювальні модулі, та обрати їх таким чином, щоб їх характеристики давали змогу тестувати алгоритми одночасної локалізації та картографування;
- спроектувати та побудувати мобільний автономний пристрій за допомогою обраних сенсорів та обчислювальних модулів;
- провести встановлення та налаштування програмного забезпечення на пристрій, який будується;
- провести тестування алгоритму одночасної локалізації та картографування у реальному середовищі;
- проаналізувати отримані мапи приміщень.

1 ОДНОЧАСНА ЛОКАЛІЗАЦІЯ ТА КАРТОГРАФУВАННЯ

Для мобільних автономних роботів є важливою задача визначення власного положення, тобто локалізація, а також побудова карти навколишнього середовища, тобто картографування. Ці дві задачі об’єднані у однойменних алгоритмах, які дозволяють з тією чи іншою точністю їх вирішити в скінченний час. Існує досить велика кількість таких алгоритмів одночасної локалізації та картографування, саме тому побудова роботу, для їх тестування у реальних, принаймні більш наближених до реальних, порівняно із тими, що зустрічаються під час комп’ютерного моделювання, є корисною задачею.

Хоча самі алгоритми не спираються на якісь конкретні дані із якихось конкретних сенсорів, ці дані мають задовольняти певним умовам, для того, щоб на їх основі можливо було побудувати карту та визначити місцезнаходження робота. Наприклад, для алгоритмів необхідно мати значення точних відстаней до об’єктів, адже це дозволить точно співставити ті дані які були отримані раніше, із новими, щоб зробити відповідні оптимізації та поправки до карти, яка була побудована раніше [1]. Такі дані якраз може надати лідар, чому він і займає міцну позицію для вирішення таких задач. Замість цих даних не підійдуть, наприклад, дані з одометру, які записують інформацію щодо кількості обертів колес і відповідно до цього, вираховують поточне положення роботу.

З іншого боку, хоча реалізувати одночасну локалізацію та картографування і можливо з використанням виключно лідара, проте найчастіше використовують декілька сенсорів. Це дозволяє порівняти припущення, щодо поточної позиції відносно початковою та попередніх ґрунтуючись на даних з цих сенсорів окремо, після чого, виконати поправки до карти, враховуючи усі дані, а також впевненість в якості тих чи інших даних порівняно один з одним. Слід також зауважити, що усі данні, які б точні сенсори не були, мають свою певну похибку, яка з часом має

властивість накопичуватись, поки робот розвідує місцевість та будує карту.

Це призводить до розсинхронізації, щодо припущення положення робота між різними сенсорами. Цю похибку і мають на меті виправити алгоритми одночасної локалізації та картографування. Адже в ідеальних умовах, навколишнє середовище можна було б просто відсканувати лідаром, і побудувати карту, але під час реального застосування, виникає багато різних перешкод, похибок у вимірюваннях, зумовлені фізичними властивостями тих чи інших елементів, що майже унеможливлює виконання поставленої мети без відповідних алгоритмів. Так само, як похибка накопичується з часом, вона виправляється за допомогою алгоритму. І через те, що це задача оптимізації, вона не може бути вирішена повністю на сто відсотків [2].

Проте більша кількість ітерацій переміщення роботу по середовищу дозволяє алгоритму більш точно скоригувати мапу, аж допоки поточний результат не задовольнить поставленим користувачем вимогам. Загалом, роботу більшості SLAM алгоритмів можна представити у вигляді наступної блок-схеми (рис. 1.1):

Рисунок 1.1 — Загальна блок-схема алгоритмів SLAM

Результатом роботи алгоритму одночасної локалізації та картографування є відповідно мапа, яка представлена у вигляді клітинок однакового розміру, кожна з яких має певну вірогідність того, чи ця клітинка є перепоною, і робот не може через неї проїхати, чи навпаки, що клітинка вільна. Ця карта може бути збережена і потім прочитана, для подальшої навігації робота базуючись на ній.

На даний момент існує багато різновидів алгоритмів SLAM. Деякі з них наведені у табл. 1.1 з коротким описом їх особливостей:

Таблиця 1.1 — Короткий огляд існуючих алгоритмів SLAM.

Назва алгоритму	Опис алгоритму
Extended Kalman Filter SLAM (EKF-SLAM)	метод, який базується на розширеному фільтрі Калмана для рішення задачі SLAM;
Distributed Particle SLAM (DP-SLAM)	один з підходів вирішенні завдань SLAM, який використовує свідчення далекоміра і фільтр частинок;
GMapping	підхід використовує фільтр частинок, в якому кожна частка несе окрему карту навколишнього середовища;
Feature-based SLAM	використовує елементи, що легко ідентифікуються в середовищі і створюють внутрішнє уявлення про простір з урахуванням розташування цих орієнтирів
Graph-based SLAM	SLAM на основі графів чи мереж також намагається створити карту за допомогою графа;
Grid-based SLAM	підхід, який базується на поділ досліджуваної області на сітку, та її використання.

Продовження таблиці 1.1

Назва алгоритму	Опис алгоритму
Topological SLAM	топологічне представлення проблеми SLAM спрямоване на створення графоподібного опису навколишнього середовища, а не точної метричної карти.

У цьому розділі будуть розглянуті принципи роботи двох алгоритмів, а саме Extended Kalman Filter SLAM, та GMapping.

1.1 Алгоритм Extended Kalman Filter SLAM

Об'єкт EKF-SLAM виконує одночасну локалізацію та картографування з використанням розширеного фільтра Калмана (EKF). Він бере спостережувані орієнтири з навколишнього середовища і порівнює їх з відомими орієнтирами, щоб знайти асоціації та нові орієнтири. Використовуючи асоціації, змінює стан і коваріацію стану. Нові орієнтири повинні бути доповнені вектором стану.

У EKF-SLAM фільтр Калмана обчислює стан системи в момент k станом $k-1$. Під час руху в якості точок інтересу обираються мітки, відносно яких визначається його положення в просторі. Мітками повинні бути окремі унікальні об'єкти досліджуваного простору, які здатний розпізнати алгоритм

Робота фільтра розділяється на два етапи: екстраполяція – прогноз стану k системи станом $k-1$ і корекція – обчислення відхилення отриманого спостереження від передбаченого спостереження. Положення пристрою (координата X, координата Y та орієнтація робота) зображуються колами, що містять в собі x з індексом. Кола, які містять u з індексом означають команду переміщення на місцевості. Коло з m позначають орієнтир, а кола, що містять u з індексом, призначені для вимірювань, означають вимірювання сенсорів орієнтирів, отриманих від робота [3].

Графічну модель алгоритму EKF-SLAM показано на рис. 1.2:

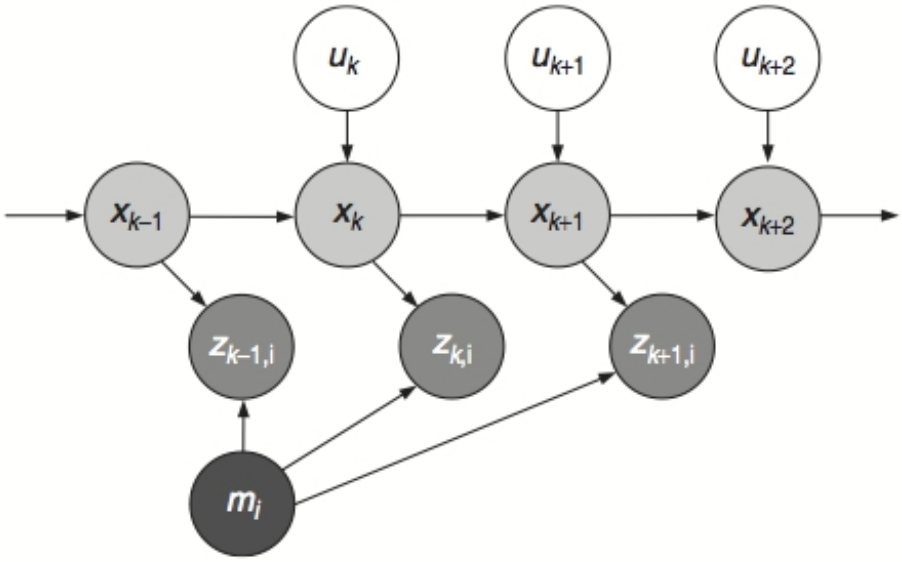

Рисунок 1.2 — Графічна модель алгоритму EKF-SLAM

У алгоритмі цілком можливі випадки виникання помилок. Це зумовлено неточністю роботи сенсорів. Коригувати положення пристрою допомагає асоціація отриманих даних.

Можливим варіантом поліпшення якості результату буде використання більшої числа міток, на яких виконувати корекцію поточного положення. При цьому обчислювальне навантаження різко зросте, оскільки складність алгоритму оцінюється як O(*N3*), де *N* — число міток. Це призводить то того, що необхідно застосовувати певні обмеження, а саме використання алгоритму у певних умовах, таких як території з меншою кількістю міток.

Фільтр Калмана, це фільтр, який базується на рекурсивній побудові алгоритму. Його вхідними даними є неповні виміри, або виміри із шумами. Задачею фільтру є оцінка поточного стану, із використанням описаних вхідних даних.

Стан цього фільтру в SLAM визначається двома основними складовими. Це оцінка вектора стану, яка представляє собою оцінку позиції

та орієнтації робота в конкретний момент часу. Вона оновлюється з кожним новим вимірюванням щодо руху пристрою. Друга складова це коваріаційна матриця помилок, яка показує міру точності оцінки вектора стану. Ця матриця відображає невизначеність та помилки в оцінці системи. Чим менше значення у коваріаційній матриці, тим точніша оцінка стану системи. Розширений фільтр Калмана (EKF) дуже схожий на простий фільтр Калмана, за виключенням, що він може бути використаний в нелінійних процесах.

Процес оцінки стану системи в рамках SLAM можна розділити на три основні етапи. Перший етап — це оновлення оцінки стану системи на основі одометричних даних. Це включає вимірювання руху робота за допомогою внутрішніх датчиків, таких як енкодери коліс, і використання цих даних для оновлення оцінки місцезнаходження та орієнтації робота. Другий етап — це оновлення оцінки стану системи на основі повторного виявлення орієнтирів. Коли робот пересувається, він може повторно засікати вже відомі орієнтири в середовищі. Ці повторні спостереження допомагають покращити оцінку місцезнаходження та орієнтації робота, враховуючи накопичену інформацію. Третій етап — це додавання нових орієнтирів до системи. Під час руху робота можуть виявлятися нові орієнтири, які раніше не були відомі. Ці нові спостереження використовуються для покращення мапи середовища та оцінки стану системи.

При всій своїй привабливості, EKF проте має свої недоліки, до яких можна віднести в першу чергу обмеження на кількість орієнтирів в системі. Зв'язано це з тим, що матриця P має розмірність mxm, де m — кількість виявлених орієнтирів. На кожному етапі оновлення матриці, бути оновлений кожен її елемент, у зв'язку з чим присутня нелінійна складність алгоритму.

Враховуючи особливості роботи алгоритма, можна зробити припущення, що використання його доцільне в випадках, коли досліджуване приміщення має орієнтири, які досить легко виділити та помітити, при цьому їх кількість також має бути невеликою.

1.2 Алгоритм GMapping

GMapping це високоефективний фільтр частинок Rao-Blackwellized для вивчення карт сітки за даними лазерного діапазону. Рао-Блеквелізовані частинки фільтру були представлені як ефективний засіб для вирішення проблема одночасної локалізації та картографування. У цьому підході використовується, так званий, фільтр частинок. Кожна частинка, при чому, являє собою окрему мапу навколишнього оточення. Алгоритм обчислює у короткій проміжок часу поліпшений розподіл даних по кожній частині бази. Це дозволяє безпосередньо використовувати більше інформації, отриману з датчиків при генерації частинок.

Головною особливістю даного алгоритма є оцінка апостеріорної ймовірності випадкової події (умовна ймовірність, присвоювана після врахування відповідного свідчення або вихідних даних). Його вихідними даними є потенційні траєкторії мобільного пристрою, натомість вхідними є дані з одометра. Ці ймовірності використовуються для вирахування апостеріорних ймовірностей. Для оцінки апостеріорної ймовірності по потенційним траєкторіям Рао-Блеквелла картографування використовує фільтр частинок, в якому окрема карта пов'язана з кожної вибіркою. Кожна карта будується з урахуванням спостережень і траєкторії , представленої відповідною часткою. Траєкторія робота розвивається відповідно до руху робота, і з цієї причини розподіл пропозицій вибирається еквівалентним ймовірнісної одометричної моделі руху. Фільтр Рао-Блеквелла для картографування поступово обробляє спостереження і свідчення одометра в міру їх надходження. Це робиться шляхом поновлення набору зразків, які представляють апостеріор на карті і траєкторію руху пристрою

Перевагою даного алгоритму є висока точність побудови карти. Однак недолік полягає в тому, що даний алгоритм має досить високу обчислювальну складність.

2 ОГЛЯД ІСНУЮЧИХ МОДЕЛЕЙ МОБІЛЬНИХ АВТОНОМНИХ ПРИСТРОЇВ

Звичайно, ідея побудови мобільних автономних пристроїв, простіше — роботів, не нова, і до цього часу було створено вже досить багато роботів, адже їх основна задача це автоматизація задача у фізичному середовищі, але щоб такі роботи будувати, необхідні і навчальні роботи, на яких будуть тестувати нові ідеї, алгоритми, тощо. Без значних витрат та небезпеки, якщо б це був, наприклад, легковий автомобіль із купою сенсорів. Що хоч і буде мати такий вигляд зрештою, якщо самокерований автомобіль це і є головний продукт, але саме тестування та розробку часто дешевше та безпечніше проводити саме на моделях. А помітно швидкий розвиток технологій призвів до того, що роботи можуть бути все меншими.

Отже огляд існуючих рішень був проведений саме в такому діапазоні, тобто серед роботів, які призначені для навчання, розробки та тестування різноманітних алгоритмів, в тому числі алгоритмів одночасної локалізації та картографування. Адже досить очевидно, що на ринку існує достатня кількість роботів із максимально точними сенсорами, наскільки ця технологія доступна за межами військової сфери та дуже потужними обчислювальними блоками. Проте такі роботи самі по собі вже є продуктами, які готові для конкретного застосування та часто мають достатньо високу ціну, щоб їх могли собі дозволити багато дослідницьких центрів. Одним із прикладів таких роботів, яких ми не беремо до уваги у цьому огляді, це робот Spot від американської компанії Boston Dynamics, який хоч і має високотехнологічне обладнання, проте за даними 2020 року, лише роботизована платформа, без додаткового навісного обладнання, такого як лідар, коштує близько 75 тисяч доларів, що звісно ставить його зовсім у іншу порівняльну категорію. Натомість ми зосередились на значно доступніших рішеннях, які існують сьогодні, із достатніми технічними можливостями, які б дозволили застосовувати та тестувати результати дослідницької діяльності у цій сфері.

2.1 Гусеничний робот від компанії XiaoR GEEK

Перший робот, який слід оглянути, це мобільний робот на гусеничній основі від компанії XiaoR GEEK. Наявність гусениць замість більш поширеної колесної бази є і перевагою і недоліком одночасно. Однозначним плюсом такої конструкції є висока проходимість, принаймні із загальних уявлень щодо принципу роботи гусениць. Проте, основним місцем застосування також роботів все ж таки є приміщення із плоскою та рівною підлогою. Тому таке рішення викликає сумніви, щодо своєї доцільності, адже це також призводить до декількох недоліків.

Перший із них, це його габарити, адже робот має такі розміри, як 28 см у довжину, 25 см завширшки та 30 см у висоту. Це хоча і не надто великий робот, але присутнє трохи надлишкове використання простору, яке вимагає апаратне забезпечення.

По-друге, це може сильно вплинути на точність такого сенсору як одометр, адже через особливість конструкції передача обертального моменту може передаватись неточно або неповністю, що ускладнює подальше використання таких даних у програмних реалізаціях алгоритмів.

До недоліків також можна віднести розташування камери, а саме її позиція понад лідаром, що не дозволяє вільно полягатися на дані з лідару для пересування, через те, що усі лідари в цій категорії роботів однополосні (з причини високої ціни сенсору), і у випадку знаходження якоїсь перешкоди над площиною лідару, але на рівні камери, це унеможливлює проїзд робота та може пошкодити камеру. До того ж, сама камера має не високу роздільну здатність та передачу даних через порт USB, що є повільним та не ефективним для потоку даних з такого сенсору як камера.

2.2 Робот моделі Yahboom AI Robot

Цей пристрій має лідар та камеру серед основних сенсорів. Хоча він вже не такий високий, як попередній пристрій і лідар знаходиться вже вище камери, проте понад лідаром також присутній дисплей розміром 6 дюймів. Хоча відображення корисної інформації про стан робота, дані, отримані під час тестування певного алгоритму, тощо, безумовно є перевагою, і може допомагати вирішувати задачі, пов’язані із тестуванням алгоритмів одночасної локалізації та картографування, проте є й суттєві недоліки. Перш за все, розташування цього дисплею не є найкращим, на думку автора, через те, що він теж знаходиться у площині робочої зони лідару, таким чином заважаючи робити скан повної окружності, втрачаючи приблизно двадцять градусів кута огляду, наскільки можна зробити таке припущення візуально. До того ж, це є ще один пристрій, який потребує електроживлення, що скорочує автономну роботу пристрою від наявних батарей або акумуляторів, не беручи до уваги той факт, що така модифікація відчутно впливає на остаточну ціну пристрою загалом. Даний мобільний автономний пристрій також має гусениці у якості своєї пересувної платформи, недоліки яких вже були описані у огляді попередньої моделі, але які загалом можна виразити у тому, що передача команд руху відбувається менш точно, аніж з колесами, та можливі менш точні дані, які будуть отримуватися із сенсору одометра. Серед переваг можна також відзначити досить велику площу, яка дозволяє встановити додаткові модулі, за необхідності.

2.3 Робот моделі TB3 BURGER 4

Наступним роботом, який ми оглянули, є модель TB3 BURGER 4 4GB. У даному випадку використовується вже колесна платформа, з двома головними керованими колесами та підтримкою у вигляді двох менших коліс. Робот має значно більш помірні розміри та більш продуману конструкцію. Таким чином, лідар знаходиться на самому верху, усі інші елементи нижче.

Перевагою також є велика кількість функціональних отворів на платформі, які дозволять зручно встановити інші апаратні елементи та сенсори за необхідності. До недоліків також можна віднести відсутність камери, що унеможливлює тестування алгоритмів, які полягаються в основному на камеру, як основний сенсор.

Ці три оглянуті моделі є досить репрезентативною вибіркою того, що з себе уявляє ринок мобільних автономних пристроїв у відносно доступному ціновому діапазоні. Усі інші варіанти є варіаціями на базі цих моделей із невеликим змінами. Усі три роботи мають вартість у одному ціновому діапазоні. Усі три використовують одноплатний комп'ютер RaspberyPi у якості головного обчислювального модулю, що на думку автора, є більше недоліком, аніж перевагою, адже сам комп'ютер був розроблений досить давно, що не вказує на його велику актуальність, і до того ж набув певного статусу і популярності завдяки рекламі, що негативно впливає на його вартість для споживачів.

2.4 Постановка задачі

Оглянувши доступні на ринку моделі мобільних автономних пристроїв для тестування алгоритмів одночасної локалізації та картографування, були виявлені певна кількість недоліків, які у певних випадках, можуть суттєво впливати на якість тестувань. Таким чином, можна прийти до переліку вимог до мобільного автономного пристрою, який проектується та будується, а саме:

- пристрій має мати такі фізичні сенсори як камера та лідар, адже лідар використовується у дуже багатьох алгоритмах одночасної локалізації та картографування;
- лідар має знаходитись на самому верху, щоб мати повний кут огляду;
- камера повинна мати достатню роздільну здатність та кут огляду;
- робот має бути оснащеним потужним обчислювальним модулем, щоб мати змогу обчислювати складні алгоритми та методи комп'ютерного

зору і потенційне застосування нейронних мереж;

- його пересування має забезпечуватися за допомогою колесної платформи;
- платформа має мати невеликі розміри відносно обчислювального модуля, двигунів та сенсорів, які використовуються для підвищеної мобільності;
- тип колісної платформи має бути диференціальним, тобто два колеса із незалежними двигунами;
- двигуни робота мають бути відкаліброваними, для коректного виконання своїх функції, щоб вносити мінімальну похибку у алгоритми, які тестуються;
- комунікація між роботом та комп'ютером має відбуватись через бездротову мережу Wi-Fi з використанням захищеного протоколу SSH, задля надання мобільності та захищеності;
- керування пересуванням мобільного автономного пристроєм має здійснюватись за допомогою клавіш стрілок на комп'ютері, за для створення однозначно зрозумілого і звичного інтерфейсу для багатьох користувачів;
- має бути присутня зручна можливість візуалізації даних, щодо положення роботи, та карти, яка будується, в режимі реального часу, щоб мати змогу відслідковувати поточні зміни, та виявляти місця некоректної роботи алгоритмів, якщо такі траплятимуться;
- результатом роботи робота після тестування алгоритму одночасної локалізації та картографування має бути мапа відсканованого приміщення у вигляді зображення, яка може використовуватися для подальшої навігації робота, якщо буде поставлена така задача;
- обрана роздільна здатність зображення мапи має бути достатньою, щоб можна було розрізнити перешкоди та вільні місця, і при цьому не надто детальною, щоб не навантажувати алгоритм до того рівня, коли його виконання завершується помилкою.

3 ОГЛЯД ВИКОРИСТАНОГО АПАРАТНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Обчислювальний модуль

Перед тим як почати конфігурацію апаратного забезпечення робота, слід визначитись із головним компонентом, за допомогою якого буде проводитись обчислення за обрахунки тієї логіки і тих алгоритмів, які надалі будуть використовуватися. Простіше кажучи, необхідно обрати процесор для робота. Наразі існує безліч пропозицій і варіантів від різних виробників, з різною архітектурою, характеристиками, тощо. Перед цим, однако, слід зауважити, що для побудови роботу для подальшого застосування на підприємствах або використання розроблених рішень поза межами будівель, тобто на відкритих ділянках, таких як вулиці міст, польові умови, необхідно також врахувати і можливість пристрою пересуватися у відповідних умовах та середовищі його використання.

Ця інженерна задача сама по собі є окремим напрямком досліджень, і з тих варіантів існуючих розробок різних фірм та дослідницьких компаній які вже існують на ринку, можна побачити цілу різноманітність платформ для пересування, починаючи від звичайної колесної платформи, і закінчуючи рішеннями, які надихались анатомією людин і тварин, створюючи людинопобідні роботи та роботи з системою пересування подібною до сімейства котових.

Ці роботи, відповідно, пересуваються або на двох “ногах”, або на чотирьох “лапах”. Одним із прикладів такої розробки є робот “Спот” (англ. — Spot), від Американської компанії Бостон Дайнемікс (англ. — Boston Dynamics). Однією із головних задач таких роботів є можливість пересуватися по рельєфу, який характерний більшості територій та середовищ діяльності людини. Навіть підйом та спускання по сходах є задачею складною і для більшості роботів неможливою. Таким чином,

фізичні розміри таких платформ в середньому є досить великими — орієнтовно від одного метру завдовжки, п'ятдесят сантиметрів в ширину та семидесяти сантиметрів у висоту відповідно. Це дозволяє встановити більшу кількість акумуляторів для більшої автономності. Що у свою чергу, дає можливість розмістити на роботі більший обчислювальний модуль, більші за розміром сенсори (у випадку, якщо збільшення розміру сенсору дозволяє покращити швидкість роботи, точність отримуваних даних, фізичний діапазон сприйняття даних тощо). Також є більше простору для вибору самих обчислювальних компонентів у сенсі їх швидкодії, адже великі за об’ємом акумулятори, як правило, можуть надати більшу електричну потужність. Тому і конфігурацію подібних пристроїв дійсно має сенс зробити самостійно обравши необхідний процесор, оперативну пам’ять, материнську плату, тощо. Але у нашому випадку, створювати такий великий пристрій не має сенсу, бо, по перше, подібне обладнання все ще коштує досить дорого, по друге, саме для тестування алгоритмів одночасної локалізації та картографування великий пристрій буде не дуже зручний, а компактний робот, навпаки, дозволить використовувати меншу кількість площі, для тестування самого пристрою.

З такою проблемою вже не одноразово стикалися велика кількість розробників, тому на ринку вже досить довго існують компактні рішення комп’ютерів, представлених у вигляді однієї плати. Вони зазвичай мають розміри не більше п’ятнадцяти сантиметрів у довжину і десяти сантиметрів у ширину, а інколи і менше. Зазначимо, що це, по своїй суті, є повноцінний комп’ютер, який має центральний процесор, оперативну пам’ять, та інші притаманні сучасним комп’ютерам компоненти. Доказом того, що такі рішення справді є комп’ютерами у сучасному розумінні цього слова є те, що вони здатні працювати із повноцінними сучасними операційними системами як GNU/Linux (та його різновиди), а також із деякими версіями ОС Windows.

3.1.1 Одноплатний комп’ютер Raspberry Pi

Одним із найпоширеніших варіантів такого комп’ютера на сьогоднішній день є RaspberryPi та його модифікації. Від початку, побудований як навчальний комп’ютер для студентів вищих навчальних закладів, пізніше цей комп’ютер набув неабиякої популярності через свою порівняно невелику вартість та гнучкість для розробки. Наразі, він вже не сприймається як виключно навчальний пристрій, а навпаки, досить часто використовується для розробки різноманітних рішень невеликими компаніями та стартапами, та нерідко інтегруються у комерційні продукти, хоча і не обов’язково у оригінальному своєму вигляді, та з відповідними модифікаціями.

Специфікації останньої актуальної моделі Raspberry Pi 4 Model B наведені у табл. 3.1:

Таблиця 3.1 — Специфікації моделі Raspberry Pi 4 Model B

Компонент	Характеристики
Модель	Raspberry Pi 4 Model B
Архітектура	ARMv8-A (64bit)
Система на кристалі (SoC)	Broadcom BCM2711
Центральний процесор (CPU)	4× Cortex-A72 1.8 GHz
Графічний процесор (GPU)	Broadcom VideoCore VI @ 500 MHz
Оперативна пам’ять	До 8 GB
Периферійні інтерфейси низького рівня	17× GPIO, 4× UART, 4× SPI, та 4× I2C
Номінальна потужність	600 mA (3 W) - 1.25 A (6.25 W)
Ethernet	10/100/1000 Mbit/s
Wi-Fi IEEE 802.11 wireless	b/g/n/ac dual band 2.4/5 GHz
Bluetooth	5.0

Як можна бачити із табл. 3.1, пристрій має достатні можливості, для створення різноманітних рішень та тестових зразків на його базі. Слід відзначити досить великий доступний обсяг оперативної пам’яті (до 8 Гб), а також велику кількість інтерфейсів низького рівня, які дозволяють використовувати велику кількість різноманітних зовнішніх модулів та сенсорів. До певного часу, а у деяких випадках і досі, цей одноплатний комп’ютер залишався стандартним вибором для розробки різних роботів та систем, для використання у побуті та виробництві.

Але останнім часом, в тому числі завдяки більш потужному та більш дешевому апаратному забезпеченню, а також завдяки значним проривам у цій області, набула популярності така сфера досліджень як машинне навчання, а саме глибинне навчання (англ. deep learning). Багато різних задач у різних напрямках науки отримали альтернативні варіанти вирішення задач за допомогою алгоритмів та моделей глибинного навчання. У деяких напрямках досліджень, в таких як комп’ютерний зір чи розпізнавання образів, використання моделей глибинного навчання взагалі є одним із найефективніших методів на сьогоднішній день. Але ці алгоритми є відносно універсальними, тому областей їх застосування є досить велика кількість і нові можуть з’являтися найближчим часом. Особливістю того, як працюють нейромережі, на яких і побудована абсолютна більшість моделей глибинного навчання є те, що більшість математичних операцій необхідних для обрахування результату можна представити у вигляді перемноження матриць, тобто важелів моделі. Хоча ці операції звісно можна виконувати і за допомогою центрального процесора, це займає значно більше часу, порівняно із тим, якщо використовувати для цього графічні процесори (англ. — GPU, Graphical Processor Unit). Такі графічні процесори першочергово розроблювались для задач, пов’язаних із графічними обчисленнями, але також знайшли своє застосування у такій області досліджень, як машинне навчання, де зараз є необхідним і незамінним пристроєм. На сьогоднішній день, у напрямку досліджень одночасної локалізації та картографування вже

існує певна кількість алгоритмів, які були розроблені і часто використовувані ще до активного розвитку нейронних мереж, і тому не мають необхідності у обов’язковому використанні графічний процесорів.

Але з іншого боку, вже існують і такі, що намагаються вирішувати ці задачі за допомогою глибинного навчання, і не виключно, що попереду нас ще очікують появи рішень задач одночасного картографування та локалізації з використанням нейронних мереж.

До того ж, в основному ці задачі спираються на вихідні дані сенсору лідар. Натомість, це не єдиний сенсор, який дозволяє вирішувати подібні задачі. Існує також клас алгоритмів, як візуальне одночасне картографування та локалізація. Він спирається, як це свідчить в назві, на візуальні дані з камери, що одразу скеровує нас у ідею імплементації нейронних мереж.

3.1.2 Одноплатний комп’ютер NVidia Jetson Nano

Хоча, наведена конфігурація одноплатного комп’ютера Raspberry Pi 4 Model B і має графічний процесор, його присутність скоріше номінальна. Принаймні, швидкодія такого графічного процесора не задовольняє вимоги, достатні для використання моделей глибинного навчання. Тому, було прийнято рішення відмовитися від використання саме такого обчислювального блоку, і розглянути інші наявні варіанти.

Одним із таких варіантів є NVidia Jetson Nano, від компанії NVidia. Цей виробник виготовляє графічні процесори, як і багато інших конкурентів, але одним із першим почав приділяти увагу напрямку досліджень глибинного навчання та нейромереж, і наразі є головним постачальником графічних процесорів, які заточені під використання нейронних мереж. У 2007 році, компанія також представила набір із засобів розробки, утиліт і документації, який дозволяє суттєво пришвидшити розробку паралельних алгоритмів, і нейромереж як наслідок. Цей набір отримав назву CUDA (англ. Compute Unified Device Architecture), на його основі було побудовано багато інших

бібліотек та програмних засобів для роботи із нейронними мережами, і наразі поки не існує подібних конкурентноспроможних рішень від інших виробників графічних процесорів, що робить компанію фактично монополістом на ринку.

Одноплатний комп’ютер NVidia Jetson Nano також має архітектуру процесора ARM, що характерно для класу цих пристроїв. Але на відміну від вище розглянутого комп’ютера RaspberryPi, у цьому випадку наявний також високошвидкісний та багатоядерний графічний процесор, що дозволяє застосовувати нейронні мережі, за необхідністю. Технічні специфікації наведені у табл. 3.2:

Таблиця 3.2 — Специфікації одноплатного комп’ютера NVidia Jetson Nano

Компонент	Характеристики
Центральний процесор (CPU)	Quad-core ARM A57 @ 1.43 GHz
Графічний процесор (GPU)	128-core NVidia Maxwell
Оперативна пам’ять	4 GB 64-bit LPDDR4 25.6 GB/s
Периферійні інтерфейси низького рівня	GPIO, I2C, I2S, SPI, UART
Ethernet	1000 Mbit/s
Wi-Fi IEEE 802.11 wireless	підтримується, відсутній модуль

Комп’ютер також має піни, які дозволяють підключати зовнішні модулі та сенсори за допомогою периферійних інтерфейсів низького рівня. До недоліків, слід віднести відсутність модуля Wi-Fi та відсутність версії з 8Гб оперативної пам’яті.

3.2 Сенсори

Не дивлячись на те, що вже існує певна кількість алгоритмів, які дозволяють виконувати задачі одночасного картографування та локалізації за допомогою візуальних даних, тобто з використанням камери як основного сенсору, що дозволяє значно знизити собівартість такого пристрою, адже камери коштують у рази дешевше за інші сенсори, особливо лідари, проте у більшості випадків, без додаткових сенсорів і особливо лідарів, важко побудувати автономний пристрій. Тому радари та лідари посідають значне місце серед сенсорів у світі робототехніки. А також різноманітні інші сенсори, які дозволяють точно вимірювати відстань.

3.3 Ультразвуковий далекомір

Одним із таких сенсорів є ультразвуковий вимірювач відстані. Принцип його роботи полягає у використанні ультразвукових, тобто високочастотних, звукових хвиль. Конструктивно, такий сенсор має передавач та приймач, і генеруючи хвилю передавачем, вимірюється час відбиття і прольоту звукової хвилі і сприйняття її приймачем відповідно. Знаючи час та характеристики хвилі, можна вирахувати відстань до об’єкта, від якого хвиля відбилась. До безумовних переваг можна віднести їх простоту та надійність, низьку ціну. Їм також характерна простота обробки сигналу, та через це висока швидкість обчислення координат. Невелика кількість обчислень та простота роботи також призводить до невеликого енергоспоживання.

Проте такий ультразвуковий сенсор має і недоліки. Серед них низька швидкість оновлення, що зумовлена швидкістю розповсюдження звуку у повітрі. Відтак, хоча у найпоширенішого у застосуванні варіанта у застосуванні ультразвукового сенсору частота оновлень становить 50 Гц, такі характеристики недостатні, для його застосування для задач одночасної локалізації та картографування.

3.4 Лідар

Тому для побудови мобільного автономного пристрою також був використаний лідар. Лідар (англ. LIDAR — Light Identification Detection and Ranging) — це технологія, яка дозволяє отримувати та обробляти інформацію, щодо віддалених об’єктів. Відбувається це за допомогою активних оптичних систем. Вони використовують фізичні явища відбиття світла та розсіювання світла у прозорих та напівпрозорих середовищах. Принцип роботи лідару полягає в тому, щоб направити лазерний промінь, зазвичай з довжиною хвилі від 600 до 1000 нанометрів, а потім сприйняти відбиття цього променю від оточуючих об’єктів, і зрештою, вирахувати відстань до об’єкту, вимірявши час відбиття променю. Отже, щоб вирахувати відстань до об’єкту d, використовується наступна формула:

$$d = \frac{c \bullet t}{2} \tag{3.1}$$

де c – швидкість світла;

t – час, за який промінь було виявлено після відбиття.

Це дозволяє досить точно дізнатись відстань до об’єктів навколо, на відміну від камери. Адже для того, щоб отримати інформацію про розташування предметів та глибину кадру, необхідно використовувати принаймні дві камери, тобто стерео зір, та/або одну камеру, але у постійному русі, щоб мати змогу порівняти сусідні по часу кадри, та визначити, що і на скільки на сильно на них змістилось відносно одне одного, базуючись на чому, можна зробити деякі припущення, стосовно розташування об’єктів одне від одного. Але цей спосіб все ще не надає точних даних, щодо відстані до вимірюваних об’єктів, на відміну від лідара.

Якщо ж багаторазово таким чином надсилати промінь у різні точки простору, можна отримати відповідні відстані. Тому, щоб отримати якомога більше інформації про навколишнє середовище, цей механізм зазвичай

обертається навколо своєї осі на 360 градусів. Якщо обертати таким чином лідар в одній площині, ми зможемо отримати мапу того, що оточує сенсор, а точніше мапу відстаней до нього від усіх об’єктів в цій площині.

Існують також і лідари, які спроможні обертати сенсор не тільки в одній площині, і таким чином будувати мапу відстаней усього навколишнього середовища. Принцип їх роботи, при цьому, залишається такий самий. Слід зазначити, що такі пристрої значно більші за розміром, потребують більше живлення, та, що головне, сильне дорожчі. Це обмежує їх використання та впровадження у деяких продуктах, проте вони активно використовуються у сферах, де їх наявність дійсно виправдана, так як, на приклад, у багатьох автомобілях з функціями самостійного керування та навігації.

До переваг використання лідару, можна віднести також той факт, що через вище описані принципи роботи лідару, освітленість приміщення або простору не має впливу. Тобто лідар буде там само працювати і у абсолютно темному приміщенні, на відміну від камери, яка не отримає необхідної інформації через відсутність світла.

У нашому ж проекті, ми використовуємо однополосний лідар, тобто лідар, який вимірює відстані на 360 градусів навколо себе, але у одній площині. Це може здаватися недостатнім, але для більшості випадків навігації у приміщенні, цього зазвичай вистачить.

Серед наявних на ринку лідарів, було обрано пристрій RPLidar A1, виробника SLAMTEC. Хоча на ринку існує велика кількість лідарів із видатними характеристиками, які дозволяють сканувати оточуюче середовище у трьох вимірному просторі, проте лідар є одним із самих дорогих за ціною сенсорів такого виду, тому їх ціна значно перевищує можливості багатьох невеликих незалежних дослідницьких центрів, і робить їх використання виправданим безпосередньо у таких продуктах, як безпілотний автомобіль. Найлижчим конкурентом обраного лідару є модель Slamtec RPLIDAR A2M12. Він працює на більші відстані від обраного, та має деякі конструктивні поліпшення, однак навіть його ціна у два з половиною

рази більша за лідар, який використовується у цьому проекті. Тому RPLidar A1 є фактично єдиним доступним варіантом, який і було обрано. Це однополосний лідар, з кутом огляду 360 градусів. Слово однополосний означає, що лідар сканує навколишнє середовище лише в одній площині, на відміну від того виду лідарів, які здатні будувати мапу у трьох вимірному просторі. Проте знову вирішальним фактором є їх ціна, яка робить їх взагалі окремим типом лідарів, з поширеним використанням у промислових розробках, або знову є таки, у безпілотних автомобілях. До того ж, обраний лідар має підтримку і відповідні драйвери у фреймворку ROS [4], що значно полегшує подальшу з ним роботу, та вказує на його актуальність. Специфікації лідару RPLidar A1 наведені у табл. 3.3:

Таблиця 3.3 — Специфікації лідару RPLidar A1

Характеристика	Значення
Діапазон вимірюваної відстані	0.15 - 12 м
Частота дискретизації	8000 Гц
Частота сканування	5.5 Гц
Кутовий діапазон	360 градусів
Тривалість запису семплу	0.125 мс
Довжина хвилі лазеру	785 нм
Швидкість передачі даних	115200 біт на секунду
Напруга на двигуні	5-9В
Напруга сканеру	5-5.5В
Вага	170г
Розміри	5x7x10 см

Як можна зрозуміти із табл. 3.3, розміри лідару порівняно невеликі, що дозволяє розмістити його на досить компактному роботі без особливих проблем, а також не створювати зайве фізичне навантаження вагою на двигуни та інші компоненти системи.

3.5 Камера

Для можливості розширення потенціалу роботу та можливості проведення подальших досліджень з використанням пристрою, була також встановлена камера. Хоча камера рідко виступає у якості єдиного сенсору саме для задач одночасного картографування та локалізації, проте важливість її наявності важко переоцінити, адже це все ще дуже корисне джерело інформації, до того ж сильно дешевше (порівняно із іншими сенсорами), а також яке споживає небагато енергії.

Способи її використання починаються від передачі візуальної інформації, для проведення спостережень або огляду, і до розпізнавання специфічних знаків, предметів, візуальних міток, тощо. Одним із прикладів такої функції є використання камер у автомобілях з адаптивним круїз контролем або автопілотом у певній мірі, де за допомогою них вдається тримати автомобіль в полосі, детектуючи на відео потоці дорожні полоси, розпізнавання знаків та світлофорів, і відповідне коригування траєкторії руху та поведінки автомобіля. Використання камер у сфері автомобільної промисловості навіть значно перевищує використання лідарів, хоча і не забезпечує повноцінний автопілот, який на сьогоднішній день прийнято вважати задачею ще не вирішеною.

У якості камери був використаний модуль IMX219-160. Найближчим його конкурентом є модуль Camera OV5647 5MP, який також дуже часто застосовується у парі з одноплатними комп’ютерами RaspberryPi. Серед переваг конкурента слід визначити його доступність у сенсі ціни. Вона є одною з найнижчих серед порівняних модулів. Також для неї присутня достатня кількість необхідної документації. Проте на цьому переваги закінчуються. До недоліків треба віднести те, що її максимальна роздільна здатність запису відео становить full hd, тобто 1920x10080 пікселів. Сенсор містить 5 мегапікселів, проти восьми у обраному варіанті. Також, кут її огляду значно менший, що відповідно, надає менше корисної інформації з

одного кадру. Тому було обрано саме камеру з модулем IMX219-160. ЇЇ характеристики наведені у табл. 3.4:

Таблиця 3.4 — Характеристики камери IMX219-160

Характеристика	Значення
Матриця	8 мп
Сенсор	IMX219
Роздільна здатність	3280 × 2464
Діафрагма	f/2.35
Фокусна відстань	3.15мм
Кут огляду	160 градусів
Спотворення	<14.3%
Живлення	3.3В
Розміри	25мм × 24мм

Як можна побачити із табл. 3.4, камера має достатню роздільну здатність, для потенційних можливих задач детекції, класифікації або сегментації. У нашому проекті камера дозволяє спостерігати за тим, де робот знаходиться в поточний момент для коригування його траєкторії руху, якщо це необхідно. Слід зазначити, що одноплатні комп’ютери, подібні до того, що використовується в цій роботі, мають спеціальний порт, який необхідно використовувати із відповідним шлейфом. Наш випадок, не виключення. Це дозволяє значно швидше передавати дані відео потоку, особливо, якщо роздільна здатність камери перевищує 1920x1080 пікселів. Ще одним варіантом є передача даних від камери через порт USB, але такий варіант у даному випадку не має переваг.

3.6 Зв'язок

Як було вище описано, одноплатний комп'ютер NVidia Jetson Nano має підтримку можливості використання Wi-Fi, проте не містить самого модулю. Тому у цьому проекті був встановлений модуль від компанії Intel Dual Band Wireless-AC 8265. Його специфікації наведені у табл. 3.5:

Теблиця 3.5 — Специфікація Wi-Fi модуля Dual Band Wireless-AC 8265

Характеристика	Значення
Кількість Вхідних/Вихідних потоків	2x2
Діапазон передачі даних	2.4, 5 ГГц
Максимальна швидкість	867 Мб на секунду
Версія Wi-Fi	Wi-Fi 5 (802.11ac)
Версія вбудованого блютуз	4.2

3.7 Платформа роботу

Для того, щоб робот мав змогу пересуватись, а також для можливості встановлення на нього необхідного апаратного обладнання, була обрана стандартна колісна платформа круглої форми. Така форма має перевагу, яка полягає в тому, що робот, у випадку зіткнення із перешкодою за дотичною траєкторією, з більшою ймовірністю відштовхнеться у протилежну від перешкоди сторону. У випадку квадратної чи прямокутної форми, є вірогідність у такій ситуації навпаки, розвернутись навколо осі у якості цієї перешкоди, зачепившись кутом платформи, та застрягнути, що призведе до необхідності зупинити або перезапустити експеримент. До того ж, завдяки геометричним властивостям форми круга, він має найбільшу площу відносно своєї довжини, порівняно з іншими геометричними фігурами, і зрештою не має кутів, які можуть заважати, під час пересування роботу. Обрана платформа має спеціальні дужки для встановлення двигунів.

3.8 Двигуни

У якості двигунів було обрано модель DC 12 V DIY Encoder Gear Motor від виробника Garosa. Обраний елемент є тактовим двигуном постійного струму із редуктором, має вищу максимальну кількість обертів за хвилину, вбудований енкодер та контролер. З досвіду попередніх побудов роботів на колісних платформах, також слід відмити вищу точність виконання обертів відносно даних які посилаються енкодеру для обертання двигуна. Це полегшує подальшу роботу, дозволяє більше полягятись на надійність отримуваних даних із сенсора одометру та точніше керувати пересуванням роботу. Конструктивно, використовується два таких двигуна і два колеса по боках платформи відповідно. Для підтримки рівноваги та балансу робота, були використані також два менших колеса без двигунів та з вільним обертом на 360 градусів. Така конструкція ще називається диференційною, адже повертання роботу відбувається за допомогою різної швидкості обертів на двигунах, на відміну від системи кермового управління, яку можна зустріти майже на будь якому сучасному автомобілі, де одночасно та на однаковий кут повертаються передні колеса, а задні при цьому не повертаються та виконують рушійну функцію у загальному випадку. Використання диференційної платформи у нашому випадку має перевагу, яка полягає у високій мобільності, адже робот тепер може розвернутися у будь яку сторону на місці, що дозволяє використовувати у значно більш тісніших приміщеннях або умовах.

3.9 Побудова роботу

Після того, як були обрані описані вище компоненти, з їх використанням, був побудований безпосередньо сам мобільний автономний пристрій. Таким чином, робот має необхідне апаратне забезпечення, включаючи його обчислювальний модуль, сенсори, такі як камера та лідар, Wi-Fi модуль, а також платформа та двигуни з колесами.

Зв’язок модулів мобільного автономного пристрою між собою наведено на рис. 3.1:

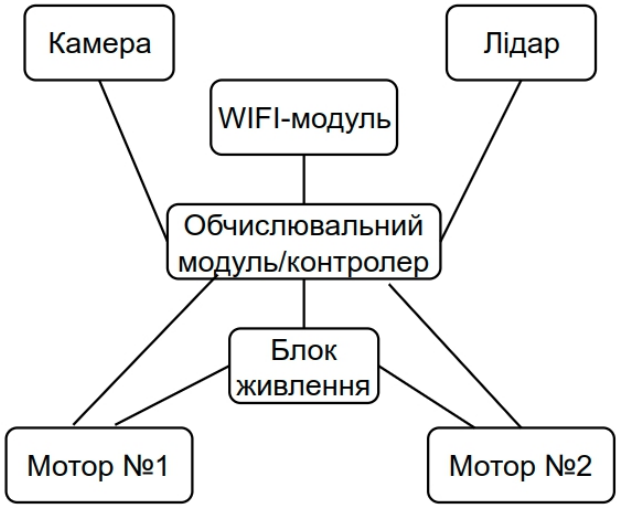

Рисунок 3.1 — Структура модулів роботу

Що ж стосується фізичних розташувань цих модулів, сенсорів та елементів, це зображено на рис. 3.2:

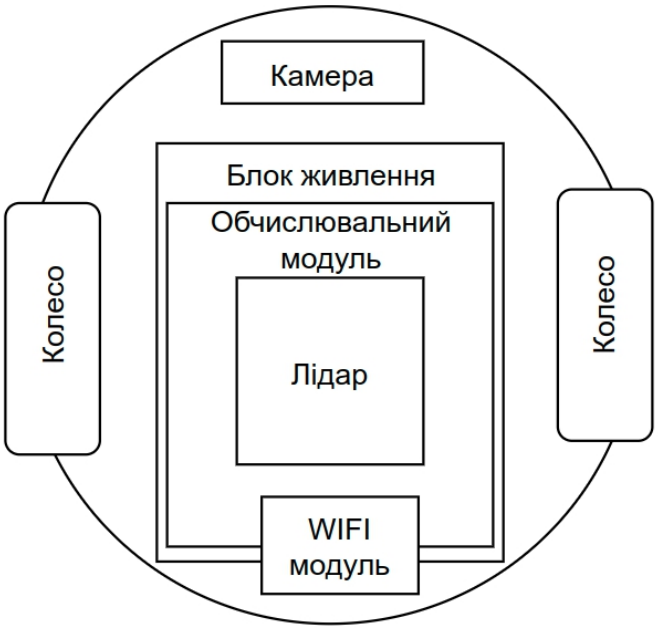

Рисунок 3.2 — План розташування модулів на роботі

3.10 Взаємодія з роботом

Побудований робот має певним чином отримувати команди від користувача, базуючись на яких він має переміщуватись. Простіше кажучи, має існувати спосіб керування мобільним автономним пристроєм. Зв’язок із роботом буде відбуватися за допомогою технології бездротових мереж, а саме у мережах стандарту IEEE 802.11, тобто Wi-Fi. Комунікація із системою роботу відбувається із використанням технології захищеного віддаленого консольного доступу SSH. Більш детально використання цих технологій у проекті описано у розділі 5. Запуск необхідних програм та алгоритму відбувається на мобільному пристрої саме за допомогою протоколу захищеного віддаленого доступу до консолі системи робота. Отримані дані з робота можна спостерігати як у текстовому вигляді у консолі, так і оглядати їх візуально, якщо специфіка даних це дозволяє, наприклад через модуль rviz. Опис цього модулю також наведено у розділі 4. Для того, щоб керувати пересуваннями робота, слід також використовувати функціональність фреймворку ROS (розд. 4), який містить необхідні драйвери для багатьох поширених моделей двигунів [5], що значно полегшує взаємодію із пристроєм. Конкретно, для задання команд пересування використовуються клавіші стрілки на клавіатурі. Відповідно, під час натискання клавіші стрілки вперед, робот прямує чітко вперед. Бокові клавіші стрілки зупиняють робот, після чого він виконує розворот на місці проти або за часовою стрілкою, відповідно до натиснутою клавіші. Клавіші сприймаються не одночасно, і актуальною вважається та, що була натиснута останньою. Відповідно, одночасно може виконуватися або обертання, або пересування вперед/назад. Це зроблено для простоти взаємодії, і за необхідності, більш складніша поведінка може бути налаштована. У даній роботі, робот використовується для тестування алгоритму одночасної локалізації та картографування, тому важливою є інформації про мапу, яка будується. Її ми спостерігаємо завдяки модулю rviz, у парі також із відео потоком з камери робота.

4 ПРОГРАМНА ПЛАТФОРМА МОБІЛЬНОГО АВТОНОМНОГО ПРИСТРОЮ

Наразі індустрія робототехніки розвивається доволі активно, і відповідно, багато різних стартапів, компаній, інженерів і окремих ентузіастів зайняті у цій сфері. І хоча існує багато різних роботів, які виконують різні задачі, усе одно на певних етапах, особливо на початкових, під час будування роботів виникають досить схожі проблеми і задачі. Для того, щоб не вигадувати кожен раз велосипед і витрачати час саме на вирішення специфічної для кожного продукту задач, був створений ROS (Robot Operating System) [6]. Хоча в назві фігурує словосполучення Operating System, тобто Операційна Система, насправді це є ОС у класичному розумінні цього слова. Скоріше це фреймворк, набір різноманітних компонентів, які закликані полегшити розробку роботів. Більше того, на сьогоднішній день майже не існує поширених альтернатив подібного роду, що з одного боку, вказує на велику кількість переваг фрейморку, а також робить його фактично індустріальним стандартом, для побудови роботизованих систем.

Головним чином, ROS дозволяє відокремлювати та не змішувати модулі і компоненти між собою. Адже однією із головних проблем, яка виникає під час побудови роботизованих систем є узлагодження коду для різних модулів між собою, і чим більше різноманітних сенсорів, датчиків, двигунів та інших елементів містить робот, тим складніше їх поєднати між собою та синхронізувати. Замість того, щоб будувати свою системи комунікації між цими модулями, що не обов’язково є поганою ідеєю, особливо у випадках, коли необхідно наприклад привести швидкодію системи у максимальний стан, або забезпечити функціональність, яку існуючі рішення не надають, все ж таки у загальному випадку, є сенс витрачати час на унікальні та нові розробки, і скористуватися готовими бібліотеками для базового функціоналу.

Незважаючи на швидкодію фреймворку та мінімальні затримки під час його роботи, ROS не є операційною системою реального часу (англ. RTOS,

Real Time Operating System), хоча він і надає можливості інтеграції коду, який має виконуватися як система реального часу. Загалом, ROS створює абстракції над апаратним забезпеченням, надає засоби керування пристроями низького рівня, має реалізацію загальнопоширених функціональностей і, що важливіше, надає засоби комунікації між процесами за допомогою повідомлень. Це дозволяє писати різні частини коду для різних задач на різним мовах програмування, адже комунікацію бере на себе фреймворк.

Система комунікації в загальному складається із вузлів (англ. Nodes) та тем (англ. Topics), уся система, відповідно, складає із себе граф [7]. Вузол представляє собою один окремий процес, який виконується в системі. Ці вузли іменовані, тобто кожен із них має своє ім’я, за яким з ним можна комунікувати. Найчастіше, вузли і містять в собі користувацький код, тобто логіку, яку користувач задає для кожного окремого модулю. Вузол може отримувати інформацію від інших вузлів, відправляти інформацію іншим, отримувати і робити запроси [8]. Теми, натомість, це іменовані шини даних, по яких відбувається передача інформації між вузлами. В самій інформації, що передається, може бути будь що: дані з сенсорів, команди для контролю двигунами, тощо.

ROS також містить багато драйверів для поширених у застосуванні компонентів. Серед компонентів, які має ROS є також багато корисних утиліт для візуалізації інформації про робота, його місцезнаходження, відображення даних із різних сенсорів та багато іншого. Цей інструмент називається rviz [9], і він, посеред інших, багато використовувався для тестування побудованого роботу. Більш конкретно, цей інструмент дозволяє будувати візуалізації пристрою у трьох вимірному просторі, причому як під час моделювання, так і його відображення під час тестування реального роботу у фізичному середовищі. Інтерфейс модуля поділений на декілька вікон, які можуть бути вибірково розміщені у зручному порядку, щоб відображати саме ту інформацію, яка цікавить користувача. Модуль містить багато корисної інформації про стан робота, елементи керування та налаштування модулів, їх

встановленні значення. Одним із головних вікон є вікно саме з трьох вимірної візуалізацією робота. Камеру спостереження, тобто позиція, з якою відображається середовище, можна повертати та змінювати її положення. Також є можливість цю камеру прив'язувати до певних об'єктів, таким чином, якщо цей об'єкт рухається, віртуальна камера рухається разом із ним, що призводить до відображення змін відносно саме обраного об'єкта. До того ж, цей модуль має вбудований функціонал, який забезпечує швидку передачу відео потоку, що дозволяє також відобразити у окремому вікні сам візуальні дані з камери пристрою, якщо вона, звісно присутня на роботі. Ця функціональність була корисною і була використана у роботі під час тестування мобільного автономного пристрою. Приклад візуального інтерфейсу модуля rviz наведено на рис. 4.1:

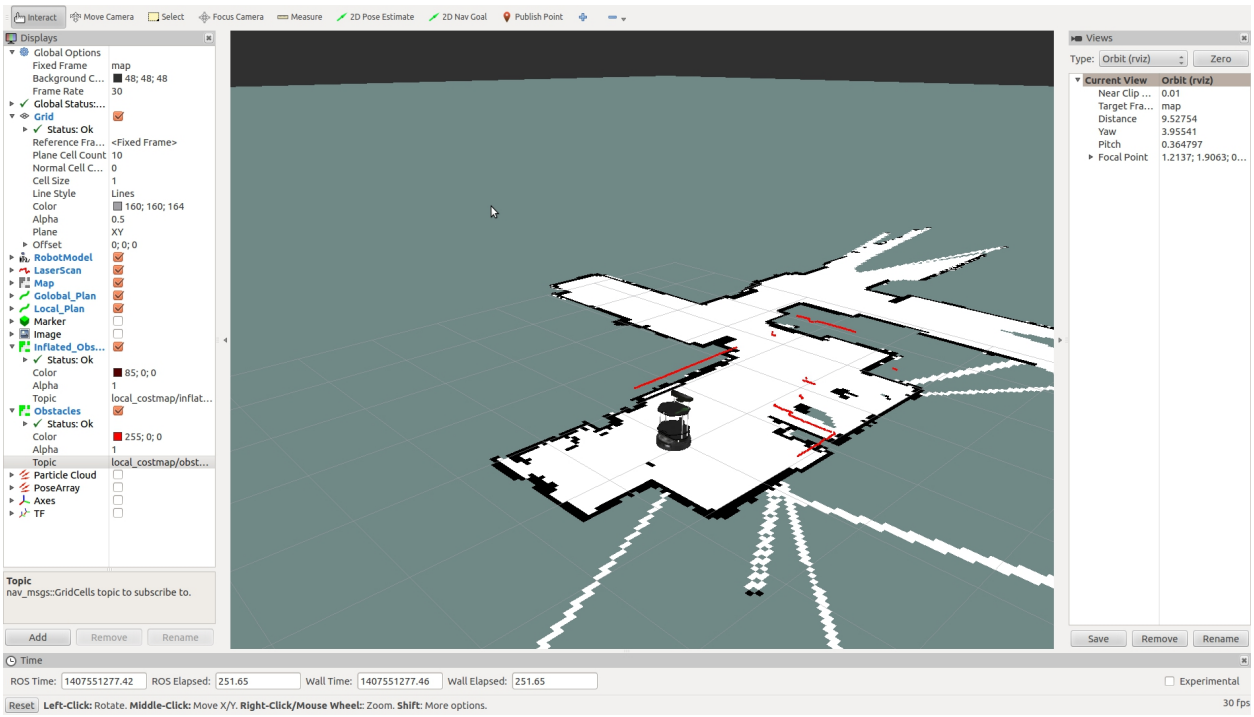

Рисунок 4.1 — Приклад інтерфейсу модуля rviz

Окрім модулів для роботи з алгоритмами, їх візуалізацією та контролю, у екосистемі фреймворку ROS присутні також і модулі самих алгоритмів безпосередньо. Ці алгоритми мають відкритий вихідний код, як і сам

фреймворк ROS. Вони, як правило, являються конкретними імплементаціями вже існуючих алгоритмів. Список таких деяких популярних алгоритмів або бібліотек наведений у табл. 4.1:

Таблиця 4.1 — Перелік популярних алгоритмів та бібліотек ROS

Назва бібліотеки	Опис бібліотеки
pick_it	Модуль для вирішення задач зворотньої кінематики .
fastrtps	Імплементація алгоритму розподілу даних у реальному часу.
ackermann_steering_controller	Система рулювання за допомогою кінематики Аккермана.
ruckig	Модуль для миттєвої генерація руху для роботів і машин.
perception_pcl	Бібліотека для роботи із хмарами точок у трьох вимірному просторі.
navigation2	Пакет для побудови маршрутів та навігації на попередньо отриманих мапах.
nav2_planner	Система для планування переміщень роботу.
robot_calibration	Модуль для калібрування робота.
libpointmatcher	Модуль співставлення точок у алгоритмах комп’ютерного зору.
vision_opencv	Портування популярної бібліотеки OpenCV, які містить безліч алгоритмів комп’ютерного зору.

І хоча вони здебільшого не є найбільш ефективними або швидкими імплементаціями серед можливих, проте вони мають достатню стабільність, для того, щоб використовувати їх у різноманітних проектах, особливо на етапі прототипування пристроїв. У певних випадках, їх функціонал,

швидкодія та надійність більш ніж достатня, для швидкого вирішення задач, із використанням роботів. Таких алгоритмів, модулів та бібліотек існує безліч, адже будь який користувач може створити свою власну, тому їх вибір та засовування залежить від задач конкретної системи.

Відтак, одним із таких модулів є пакет GMapping [10]. Як слідує із його офіційної документації, це є реалізація однойменного алгоритму одночасної локалізації та картографування, який базується на даних, отриманих з сканів лідару, та дозволяє побудувати двовимірну мапу приміщення. Цей модуль фреймворку ROS має велику кількість параметрів (близько 37), які дозволяють налаштувати його роботу та поведінку. Список налаштувань, застосованих у даній роботі, наведений у додатку В. Натомість, перелік головних параметрів з їх описом, наведений нижче у табл. 4.2:

Таблиця 4.2 — Перелік основних параметрів алгоритму GMapping.

Назва параметру	Опис параметру
map_update_interval	Як часто, у секундах, виконувати оновлення до мапи. Більш часте оновлення призводить то точнішої мапи, проте збільшує обчислювальне навантаження.
maxUrange	Максимальне значення застосованого діапазону лідара.
sigma	Значення сігми, для виконання зпівставлення знімків лідару.
kernelSize	Розмір ядра, в якому необхідно шукати співвідношення.
iterations	Кількість ітерацій, для виконання зпівставлення сканів.
linearUpdate	Частота опрацювання сканів, відповідно до відстані, які була проїхана роботом.

Продовження таблиці 4.2

Назва параметру	Опис параметру
angularUpdate	Частота опрацювання сканів, відповідно до куту, на який було повернуто робот.
temporalUpdate	Частота опрацювання сканів, якщо останній скан був оновлений n секунд раніше.
particles	Кількість часток, для застосуванні у фільтрі.
delta	Роздільна здатність мапи (або ж розмір клітинки) у метрах.
transform_publish_period	Частота передачі оновлень мапи для візуалізації.
occ_thresh	Поріг сприйняття клітинки як зайнятої.
maxRange	Максимальна відстань роботи лідару. Усе що сприймається далі цієї відстані, вважається за замовчуванням вільними клітинками.

Таким чином, при правильному та коректному налаштуванні параметрів алгоритмів, такі модулі можуть успішно використовуватися та виконувати поставлені перед ними задачі навіть в комерційних продуктах, без необхідності розробляти або імплементувати такі алгоритми власноруч, що є затратною задачею стосовно часу та вимагає залучення вимагає висококваліфікованих спеціалістів. Що може стати нераціональним використанням наявних коштів, якщо сфера застосувань та задачі мобільного автономного пристрою не вимагають цього.

5 НАЛАШТУВАНЯ СИСТЕМИ

Після того, як робота було зібрано, усі контакти під’єднано, настав час встановлювати та налаштовувати програмне забезпечення. У якості операційної системи було обрано GNU/Linux, а саме дистрибутив Ubuntu 18.04 для робота, та Ubuntu 20.04 для локальної робочою машини. Як можна побачити, версії операційних систем не співпадають, більше того, не співпадають і версії ROS, адже відповідні ревізії фреймворку мають встановлюватись на відповідні ОС, але завдяки гнучкості фреймворку ROS, комунікація між машинами відбувалась безперешкодно.

Вибір ОС був швидким, адже усього, в загальному, існує три варіанти, два з яких це ОС Windows та MacOS, розробки компаніях Microsoft та Apple відповідно. Хоча вони обидві є дуже поширеними серед користувачів по всьому світу і досить зручними у повсякденному використанні, вони обидві є проприєтарними системами із закритим вихідним кодом. Натомість ОС GNU/Linux є відкритою системою. Вона набула великої популярності не тільки як серверна ОС, а й основна система для нових робототехнічних розробок, адже має увесь необхідний функціонал, включаючи можливість мережевої взаємодії, міжпроцесну комунікацію та інше.

Одноплатний комп’ютер, який використовується у якості головного обчислювального компоненту робота не має власного накопичувача у вигляді жорсткого диска чи твердотілого накопичувача, проте присутня можливість використання портативної флеш-карти пам’яті, на яку треба записати відповідно образ ОС, для того, щоб мати можливість працювати з роботом.

Це є у даному випадку перевагою, адже можна самостійно обрати необхідний об’єм пам’яті, та замінити картку за необхідності. Після встановлення операційних систем на комп’ютер робота, та локальну машину, необхідно налагодити взаємодію між цими комп’ютерами. Хоча обчислювальний модуль NVidia Jetson Nano, який використовується має інтерфейс для передачі цифрових відео та аудіо даних у вигляді HDMI (англ.

High Definition Multimedia Interface) і характеристика самого комп'ютера дозволять встановити і використовувати ОС з графічним інтерфейсом, саме підключення до монітора має відбуватись за допомогою кабелю. З одного боку, це дозволить використовувати графічний інтерфейс системи, що корисно, для різноманітних візуалізацій, але такий варіант взаємодії із роботом має суттєві недоліки.

Побудований робот є мобільним і автономним, проте підключення до монітору за допомогою кабелю унеможливлює його мобільність, тобто пересування по зонах його тестування. Або ж навпаки, не має можливості отримувати детальну інформацію про його стан, якщо робот знаходиться у режимі тестування, і не підключений до графічного дисплею (проте слід зазначити, що підключення до дісплею все ж таки необхідне на початку роботи, для підключення до бездротової мережі, наприклад Wi-Fi, у випадку, коли немає можливості підключення за допомогою кабелю Ethernet).

Для вирішення подібних проблем існує технологія SSH (англ. Secore Shell Protocol). Це криптологічний мережевий протокол транспортного-прикладного рівня, якщо звертатися до моделі OSI. SSH надає можливості захищеної комунікації із іншим комп’ютером по незахищеній мережі. Найпоширенішим засобом використання є віддалена автентифікація і авторизація та виконання запитів командного рядка. Так як комунікація відбувається по мережі, ми можемо використовувати будь який її стандарт, в тому числі стандарт IEEE 802.11, тобто Wi-Fi. Це дозволить нам керувати роботом віддалено та безпосередньо під час його роботи та тестування. Для того, щоб підключитись до віддаленої машини за допомогою SSH, необхідно спочатку дізнатись ip-адресу пристрою, до якого має відбуватись підключення, а також ім’я користувача, у чию робочу область буде відбуватись підключення. Це також може бути і так званий root користувач, тобто привілейований користувач, який за замовчуванням має набагато більше прав, ніж будь який інший користувач у системах, побудованих на основі ОС GNU/Linux. Таким чином, формат команди для підключення через

ssh, буде виглядати наступним чином (рис. 5.1):

```
ssh [ім’я_користувача]@[IP_адреса_серверу]
```

Рисунок 5.1 — Формат команди для встановлення з’єднання через ssh

Слід зазначити, що Як тільки підключення відбулося, у нас є можливість працювати із роботом віддалено, без застосування підключення фізичного монітору до обчислювального модулю робота, як було зазначено вище.

Наступним кроком, стало встановлення фреймворку ROS. З тої причини, що встановлення багатьох бібліотек та необхідних модулів у систему відбувається у вигляді, так званих, пакетів, перш ніж їх встановлювати, слід оновити пакетним менеджер. Результатом виконання такої команди буде оновлення списків доступних пакетів, їх версій, задля уникнення конфліктів під час встановлення, а також використання свіжих версій пакетів. Хоча для різних версій реалізацій ОС на базі GNU/Linux розроблені різні пакетні менеджери, принцип взаємодії з ними залишається схожим. У нашому випадку, використовується ОС сімейства Ubuntu, і стандартним пакетним менеджером для неї є apt (apt-get у старіших версіях). Щоб оновити списки пакетів, використовується наступна команда (рис. 5.2):

```
sudo apt-get update
```

Рисунок 5.2 — Команда для оновлення списку доступних пакетів

Ця команда використовується разом із програмою sudo, адже оновлення списків означає фізичний запис або оновлення інформації у файлах, які вважаються системними, і модифікація яких, за замовчуванням не доступна усім користувачам. А команда sudo дозволяє делегувати певні задачі користувачам, які не мають прав суперкористувача.

Після цього вже можна сміливо встановлювати сам фреймворк ROS. Цей фреймворк є програмним комплексом, що досить активно розвивається і оновлюється, тому існує багато його версій. На момент виконання роботи,

було обрано версію під назвою Melodic. Проте, завжди слід перевіряти офіційні джерела, на наявність більш нових варіантів. Встановлення відбувалося за допомогою наступної команди (рис. 5.3):

```
sudo apt install ros-melodic-desktop-full
```

Рисунок 5.3 — Встановлення фреймворку ROS

Варіант команди із розширенням desktop-full вказує на встановлення одразу багатьох часто використовуваних модулів екосистеми ROS для багатьох задач. Існує можливість встановлення і більш конкретних варіацій збірок, із меншою кількістю додаткових модулів, що може бути корисним у випадках, коли об’єм дискового пространства сильно обмежений. Проте обране нами апаратне забезпечення дозволяє уникнути такого виду незручностей.

У фреймворку ROS спів існує багато програм, тому для зручного їх використання та їх взаємодії, треба запустити у консолі файл налаштувань, який зазвичай знаходиться у /opt/ros/melodic/ та має назву setup.sh. Він написаний на мові сценарію оболонки Shell script і дозволяє завертатися до різних програм фреймворку без вказання їх точного знаходження у файловій системі, і взагалі дозволяє використовувати ROS за призначенням. Хоча Shell script призначений для запуску у інтерпретатору командного рядка Bourne shell, у тому ж каталозі також знаходяться файли setup.bash та setup.zsh, для більш поширеніших у використанні оболонок Bash та Zsh відповідно.

Для запуску цього файлу, використовується команда source, яка дозволяє запустити подібні скриптові файли (рис. 5.4):

```
source /opt/ros/melodic/setup.bash
```

Рисунок 5.4 — Виконання файлу налаштувань ROS

Проте, після перезавантаження ОС, для подальшої роботи із ROS треба буде знов виконати цю команду. Більше того, її необхідно буде виконати для кожної командної оболонки, яка відкривається, адже це є новим сеансом. Це може бути незручним під час активної роботи із ROS, тому можна призначити виконання цієї команди на початку кожного завантаження командної оболонки, дозаписавши команду, яку необхідно виконувати, у файл .bashrc у домашній директорії користувача (рис. 5.5):

```
echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
```

Рисунок 5.5 — Автоматизація налаштування середовища для ROS

Далі слід подібним чином встановити інші залежності, повний список яких наведено у додатку А. Коли усі необхідні пакети встановлені, можна протестувати, чи працює ROS, запустивши команду roscore, яка в свою чергу запустить ROS Master, а також сервер параметрів ROS та вузол для збереження та демонстрації логів. У нашому випадку процес запустився без критичних помилок, із чого можна зробити висновок, що ROS встановлено коректно. Ці операції були виконані на одноплатному комп’ютері NVidia Jetson Nano, тобто на обчислювальному модулі робота, і майже в такому самому вигляді необхідно їх відтворити і на головній машині (робочому комп’ютері, тощо), з урахуванням того, що там також використовується ОС заснована на GNU/Linux, зокрема Ubuntu. Слід лише врахувати і коректну версію ROS відповідно до встановленої ОС. З рештою, увесь процес налаштування аналогічний.

Для того, щоб перейти до етапу створення проекту, теж необхідно виконати деякі операції, які є специфічними для проектів розробляємих для ROS. Одним із головних відмінностей, є особлива система збірки, яка відрізняється від загальнопоширених систем, головним чином через те, що була побудована для використання у середовищі ROS. Взагалі, головною задачею система збірки (англ. build system) є генерація так званих цілей (англ. targets) на основі вихідного, які можуть бути використані користувачем. У

якості цілей можуть поставати бібліотеки, виконувані файли, згенеровані скрипти, інтерфейси, або будь-що, що не є статичним кодом.

Щоб генерувати такі цілі, система зборки вимагає інформацію щодо розташування компонентів набору інструментів (англ. toolchain), розташування самого вихідного коду та його залежностей, в який спосіб та де ці цілі мають бути знегеровані, куди встановлені, тощо. Усі ці налаштування та параметри, як правило, мають бути записані у спеціальний файл, який і прочитується системою збірки.

Наразі поширені такі системи збірки як GNU Make, CMake, Apache Ant та багато інших, які інтегровані у середовища розробки. Різні системи також більш поширені для певних мов програмування. Хоча ROS і є одним фреймворком, багато модулів, які в ньому можуть використовуватися написані на різних мовах програмування, із своїми системами збірки, які самі по собі можуть бути складними у опануванні та налаштуванні (переважно через велику кількість залежностей), що безперечно викликає неабиякі незручності. Для вирішення цієї проблеми була створена rosbuild, а пізніше catkin, яка і є стандартною системою збірки для проектів у екосистемі ROS на сьогоднішній день. Вона була розроблена на основі CMake та покликана уніфікувати систему збірки для ROS, незалежно від мови програмування.

Для роботи була створена директорія catkin_ws. Сам код буде знаходитись у піддиректорії src, а згенеровані цілі знаходитимуться у каталогах build та devel. Каталог build є стандартним для системи CMake, натомість devel є директорією для згенерованих файлів catkin. Сама генерація відбувається за допомогою команди catkin_make.

За для того, щоб користуватися такими сенсорами як лідар та камера, які використовуються у даній роботі, необхідно встановити функціональні пакети для цих сенсорів. Для лідару був використаний модуль rplidar_ros від офіційного розробника самого пристрою — Slamtech. Для його встановлення необхідно спочатку завантажити необхідний код з репозиторію GitHub, де він зберігається у відкритому доступі. Далі цей модуль слід збудувати, за

допомогою систему збору catkin, в результаті чого також отримаємо відповідний оновлений файл із скриптом сценарію оболонки, який треба запустити, для подальшої коректної роботи. Загалом послідовність команд виглядає наступним чином (рис. 5.6):

```
cd ~/catkin_ws/src
git clone https://github.com/Slamtec/rplidar_ros.git
cd ~/catkin_ws && catkin_make && source
~/catkin_ws/devel/setup.bash
```

Рисунок 5.6 — встановлення бібліотеки лідару

Оператор && у даному випадку означає об’єднання декількох команд у одну та послідовне їх виконання, у разі успішного завершення кожної попередньої підкоманди.

Аналогічним чином були встановлені із репозиторія фреймворку ROS драйвери [11] для камери моделі IMX219-160 яка використовується у роботі, а також модуль для більш зручної взаємодії із камерою на прикладному рівні.

Повний список команд для встановлення необхідного ПЗ для камери наведено нижче (рис. 5.7):

```
cd ~/catkin_ws/src
git clone https://github.com/peter-moran/jetson_csi_cam.git
git clone https://github.com/ros-drivers/gscam.git
cd gscam
sed -e "s/EXTRA_CMAKE_FLAGS = -
DUSE_ROSBUILD:BOOL=1$/EXTRA_CMAKE_FLAGS = -DUSE_ROSBUILD:BOOL=1
-DGSTREAMER_VERSION_1_x=On/" -i Makefile
cd ~/catkin_ws && catkin_make && source
~/catkin_ws/devel/setup.bash
```

Рисунок 5.7 — встановлення ПЗ камери

У даному випадку також довелось внести деякі оновлені флаги компіляції у файл зборки Makefile, після чого знову скомпілювати проект та запустити оновлений файл командної оболонки.

6 КАЛІБРУВАННЯ ДВИГУНІВ

Через те, що багато алгоритмів одночасної локалізації та картографування, окрім інших сенсорів, використовують також одометр, дуже важливо налаштувати робота так, щоб було чітке співпадіння між тим, які дані посилаються двигунам, і тим, як насправді перемістився пристрій після виконання отриманих команд. Адже невиконання або нечітке виконання команд, пов’язаних із переміщенням буде лише накопичувати похибку, і заторможувати або взагалі унеможливлювати подальшу роботу алгоритму після його запуску.

Як вже було сказано у попередніх розділах, у даному роботі використовується диференційоване керування, тобто присутні два незалежні двигуни, і його переміщення відбувається через передачу окремих сигналів кожному з двигунів відповідно. І хоча двигуни, які використовуються, є достатньо високої якості, щоб задовольняти потреби і задачі, які ставляться перед ними у цій роботі, будь який двигун має свої особливості, та може трохи відрізнятися від іншого у прийнятних межах.

Під час запуску робота у режимі керування, якраз виникла така ситуація. При команді їхати чітко вперед, робот потрохи повертав на одну із сторін, тобто один із двигунів обертався швидше за інший, хоча і отримували вони однакові керуючі дані, що і спричиняло повертання. Якщо залишити таку проблему невирішеною, це може призведе до наслідків, коли причина неконтрольованої або непередбачуваної поведінки робота може бути не очевидною і полягати у багатьох пунктах, що могло піти не так. Тому такий простий крок калібрування обов’язково виконується перед тим, як приступити до запуску або тестування алгоритмів будь якого виду [12].

Для того, щоб калібрувати двигуни, необхідно спочатку визначити кількісну величину похибки, яка трапляється. Для цього, використовуючи мобільний пристрій із вбудованим компасом, такий як мобільний телефон, наприклад, було встановлено зверху робота. Після чого, була запущена

команда обертання навколо своєї осі на 360 градусів. Через те, що один із двигунів повертається швидше, ніж потрібно, це призведе до того, що кут на компасі після обертання буде більший за 360, якщо ж двигун обертається повільніше, кут, відповідно, буде менший за 360. Отримавши ці дані, необхідно вирахувати коефіцієнт відношення *k* за наступною формулою:

$$k = k\_prev \bullet \frac{m}{t} \tag{6.1}$$

де *k_prev* – попередній коефіцієнт калібрування, початково 1;

m – отримане під час вимірювань значення;

t – значення, яке було задано попередньо і очікувалось отримати.

Після отримання коефіцієнту, необхідно оновити дані у відповідному launch файлі, щоб робот зміг його застосувати та провести повторне тестування, та зафіксувати результати. Якщо отримані дані не влаштовують, повторити розрахунок, але вже з врахуванням параметру *k_prev* зі значенням попередньо вирахуваного значення *k*. Таким чином повторюючи описані операції, можна отримати задовільну поведінку двигунів. Також відкалібрувати слід не тільки кутову кореляцію, тобто повороти, а й лінійну. У цьому випадку, достатньо вимірити певну відстань, наприклад один метр, і після запуску робота чітко вперед, порівняти реальну відстань, яку проїхав пристрій, і ту, що була задана попередньо. У випадку критичного неспівпадіння, слід за допомогою вищеописаної формули провести аналогічні кроки та оновити відповідне значення лінійної кореляції. У нашому випадку, довелось калібрувати обидва показники. В результаті проведених операцій, у файлі налаштувань були оновлені необхідні параметри (рис. 6.1):

```
<param name="linear_correction" value="1.027"/>
<param name="angular_correction" value="1.05"/>
```

Рисунок 6.1 — Файл налаштувань робота

7 ТЕСТУВАННЯ ПРИСТРОЮ

Після побудови та налаштування усіх необхідних систем, необхідно було протестувати роботу пристрою у задачах одночасної локалізації та картографування. Тобто запустити певний алгоритм для його використання у реальному фізичному середовищі. У якості алгоритма для тестування було обрано GMapping. Тестування було проведено у двох приміщеннях. Перше має досить невеликі розміри, що складає 3 метри у ширину та 5 метрів у довжину. А також алгоритм був запущений і в більшому приміщенні, розміри якого вже складали 15 метрів завширшки та 25 метрів завдовжки.

Під час побудови карт за допомогою алгоритму одночасної локалізації та картографування, пересування самого роботу керувалось віддалено за допомогою комп’ютера, який був підключений до пристрою через захищену консоль по протоколу ssh у спільній мережі Wi-Fi. Головною задачею під час картографування було відвідати усі доступні для проїзду місця у даному просторі, щоб забезпечити лідару можливість відсканувати усі видимі перешкоди на його шляху. Візуалізація поточних даних, зокрема спостерігання відео потоку з камери робота та поточна мапа, яка будується, а також візуалізація даних з лідару виконувалась за допомогою модуля rviz (див. розділ 4). Це значно полегшує сприйняття роботи алгоритму, та покращує загальне уявлення про поточну ситуацію, особливо у випадках, коли виникають помилки у роботі алгоритму, або отримуються неочікувані результати. Спостереження за цими даними у режимі реального часу, що дозволяє фреймоворк ROS та його пакет rviz, значно полегшує процес знаходження можливих причин некоректної поведінки мобільного пристрою. Хоча rviz відображає велику кількість інформації, у цій роботі, для наочної демонстрації використовувалися головним чином два вікна цієї програми. Перше вікно відображає відео потік з встановленої на роботі камери. Це необхідно для того, щоб мати уявлення про реальне поточне

місцезнаходження роботу та про предмети, які його оточують. А також, для визначення точного місця, у якому алгоритм перестав працювати належним чином, якщо таке вийшло. Провівши швидкий огляд відео кадрів, можна зробити припущення, щодо об’єктів навколо, наприклад їх фізичних характеристик пов’язаних із відбиттям світа, які призводять до отримання ненадійних або нестабільних даних з лідару, тощо. У другому вікні, присутня саме візуалізація умовної моделі робота у просторі, разом із мапою, яка будується, а також даними сканів лідару. У якості приклада, наведений скріншот роботи модулю rviz із вказаними візуалізаціями під час тестування алгоритму GMapping (рис. 7.1):

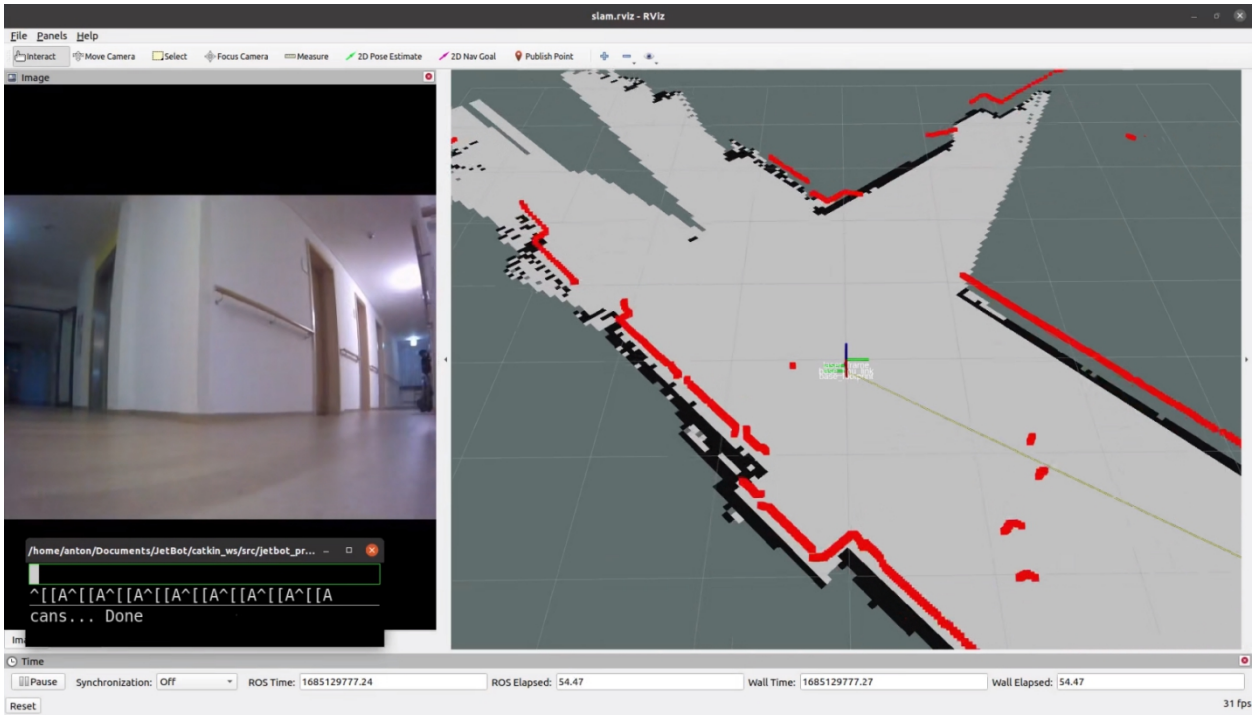

Рисунок 7.1 — Візуалізація даних модулем rviz під час тестування пристрою

Відтак, червоним позначені поточні скани з лідару, а чорним, частини вже побудованої мапи. Перехрестя у середині візуалізації зправа позначає поточне положення робота, згідно з уявленням алгоритму. Хоча передача даних, які стосуються мапи та сканів з лідару є досить швидкою, адже не так багато інформації треба, щоб описати ці явища, передача йде через

бездротову мережу Wi-Fi, і якщо до неї, під час тестування, підключено багато користувачів, або відбувається ще якась передача інших даних, окрім описаних, або ж характеристики самого мережевого обладнання є застарілими, передача саме відео потоку, особливо без застосування компресії, може призводити до повільнішого оновлення даних, значних паузах між кадрами, підвисання, тощо. Особливо враховуючи, що обрана камера має роздільну здатність 3280 на 2464 пікселів, задля уникання подібних проблем, саме для передачі відео даних для відображення у rviz, було обрано роздільну здатність 640 на 480, що все ще вистачає, для створення чіткого зображення з передачею такою самою корисною інформацією, але вже без зайвого навантаження на мережу.

Швидкість пересування була встановлена невелика, відносно тої, яку здатні розвивати двигуни, а саме 1.5 км/год із 8 ми км/код, з якою здатна рухатися побудована колісна платформа. Таке рішення було прийнято навмисно, адже, як правило, саме побудова карти відбувається один раз, після чого ця карта може використовуватись багаторазово як для задач навігації самого робота, так і для задач, зовсім не пов’язаних із пристроєм, але де необхідна подібна карта, і робот використовується саме для її отримання. Тому у будь якому випадку, карту слід побудувати достатньо точно, наскільки це дозволяє обладнання та обраний алгоритм. Низька швидкість, натомість, у певній мірі забезпечує пересування робота більш передбачувано та контрольовано. Це зменшує ризик фіктивного прокручування колеса робота, у випадку слизької поверхні, дає сенсорам, одометру та зокрема лідару, більше часу для отримання інформації про навколишнє середовище, просканувати видимі зони декілька раз із невеликою різницею у позиціюванні, та порівняти отримані скани, для продовження роботи алгоритму.

У обох приміщеннях, хоча вони за своїм плануванням не дозволяють зробити круг, пересуваючись навколо якоїсь колони чи кабінету, проте, після досягнення найвіддаленіших точок, після сканування усіх кутків, процедура

побудови карти не була завершена моментально, а навпаки, робот був скерований на початкову позицію, звідки стартував його запуск, щоб дати роботу зайвий раз порівняти отримані дані із уявленням алгоритму, щодо побудованої мапи. У деяких випадках, це дозволить точніше зійтись алгоритму.

Загалом, побудова мапи у першому, меншому за розмірами приміщеннями зайняло 50 секунд, у випадку більшого приміщення, процес відбувався приблизно 5 хвилин. Тобто це залежить не від роботи самого алгоритму, а від відстані, яку фізично треба подолати пристроєм, щоб відскакувати зони, які нас цікавлять. Результатом роботи алгоритму є мапа відсканованого приміщення. Вона постає у вигляді зображення формату PGM (англ. Portable Gray Map), що є простим форматом зображення, яке може бути представлене у вигляді набору звичайних символів таблиці ASCII і являє собою по суті текстовий файл. Відповідно до назви, цей формат зберігає кожен піксель як величину відтінку сірого кольору від 0 до 255, що складає 8 біт, або один символ у таблиці ASCII.

Таким чином чорний піксель має значення 0, білий, відповідно, 255. У проміжку, колір є відтінками сірого. Що ж щодо інтерпретації цих значень у якості мапи, то чорний колір позначає перешкоду, таку як, наприклад, стіна, ніжки стільця чи якась коробка, якщо мова йде про приміщення. Вважається, що на такі пікселі робот не має можливості заїжджати, бо вони є недоступними і мають відображати реальні об’єкти фізичного світу, і колізії з ними треба оминати, якщо тільки не стоїть зворотня задача перед пристроєм. Білі пікселі навпаки, позначають зони, в яких робот може вільно пересуватися. Сірі пікселі, вважаються територією ще розвіданою та не картографованою. Такі пікселі можна як оминати, так і намагатися туди проникнути, якщо стоїть задача дорозвідки місцевості та оновлення мапи.

Перевагою зберігання мапи у вигляді зображення такого формату, окрім простоти самого формату, є і той факт, що як і будь яке інше зображення, цю мапу можна відкрити у графічному редакторі, і замалювати

необхідні пікселі білим, сірим або чорним кольором, таким чином оновивши мапу, або виправивши деякі неточності, без використання якогось спеціалізованого ПЗ.

У якості прикладу можна навести випадок, якщо якийсь предмет у приміщенні, позначений на мапі, більше у цьому приміщені не знаходиться, або ж нас цікавить лише план приміщення, не беручи до уваги те, що там має бути, ми можемо просто замалювати такі ділянки білим кольором (позначивши ці ділянки як вільні), та продовжити використання мапи, без необхідності пересканування та побудови нової. Приклад мапи, яку ми отримали в результаті тестування алгоритму одночасної локалізації та картографування у маленькому приміщені наведено нижче (рис. 7.2):

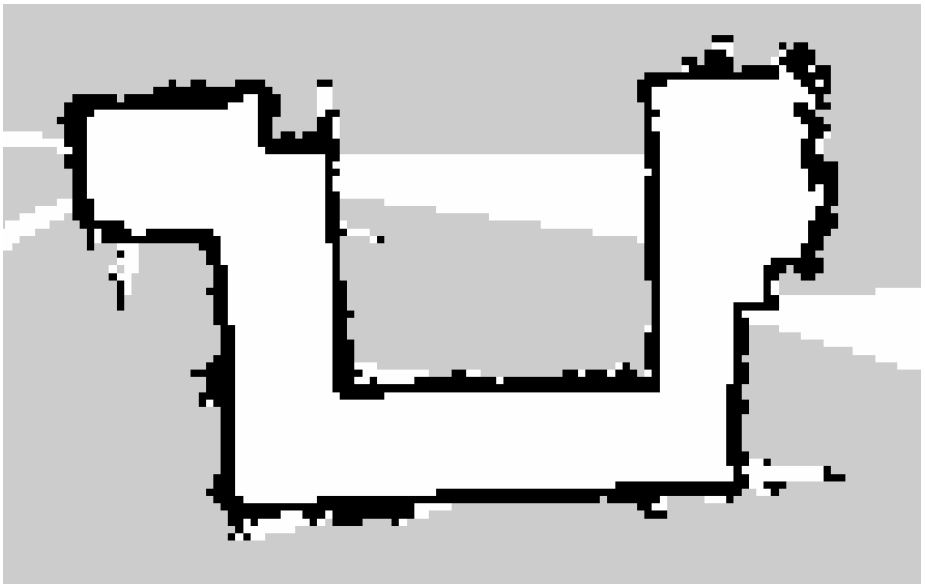

Рисунок 7.2 — Мапа першого відсканованого приміщення

Слід зазначити, що під час роботи алгоритму картографування, пікселі не мають одразу крайніх значень, такі як 0 та 255 (чорний та білий відповідно), як це показано на мапі результату. Натомість, у кожного пікселя є ймовірність, виражена у відсотках, яка позначає впевненість алгоритму в тому, чи є даний піксель перешкодою, чи навпаки, вільним для пересування. Із часом роботи алгоритму, пікселі все ж таки встановлюються у певний клас,

в залежності від встановленого рівня. У даній роботі, рівень встановлений у значення 0.65, тобто піксель позначається як перешкода, тобто чорним, якщо впевненість алгоритму у цьому становить 65 відсотків, або більше. На рис. 7.2, можна також побачити, що деякі області поза очевидними межами позначені білим, тобто як придатні для пересування. Враховуючи планування приміщення, та розташування речей у ньому, таке уявлення не відповідає дійсності. Проте завдяки тому, що межі перешкод були розпізнані пристроєм правильно, це унеможливлює потрапляння туди робота, або неправильне його скерування методами навігації, адже через наявність меж, позначених чорним кольором, поміж них не існує достатнього місця для проїзду робота, яке б дало можливість скерувати його туди алгоритмам, які покладаються на таку мапу. Так, на рис. 7.3 наведений приклад, отримай під час тестування алгоритму, на якому відображена поведінка лідара у взаємодії зі склом:

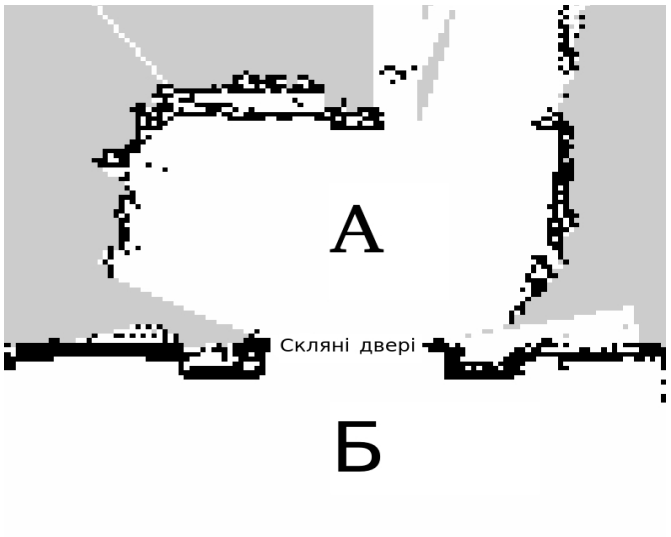

Рисунок 7.3 — Взаємодія лідара зі склом

Причиною появи таких ділянок є особливості роботи лідару, пов’язані із відбиттям світла та характеристиками поверхонь, які по різному відбивають світло.

Через те, що робот оснащений відео камерою, потік з якою також передається у програму rviz для спостереження, що було налаштовано у цьому проекті для зручноті роботи, ми маємо змогу, окрім припущень, побудованих на сканах мапи, отриманої в результаті роботи алгоритму, зробити також висновок, щодо ситуації, базуючись саме на візуальних даних, і таким чином підтвердити або спростувати версію, побудовану на припущенні. На рис. 7.4 наведений скріншот візуалізації у програмі rviz разом із камерою та лідаром:

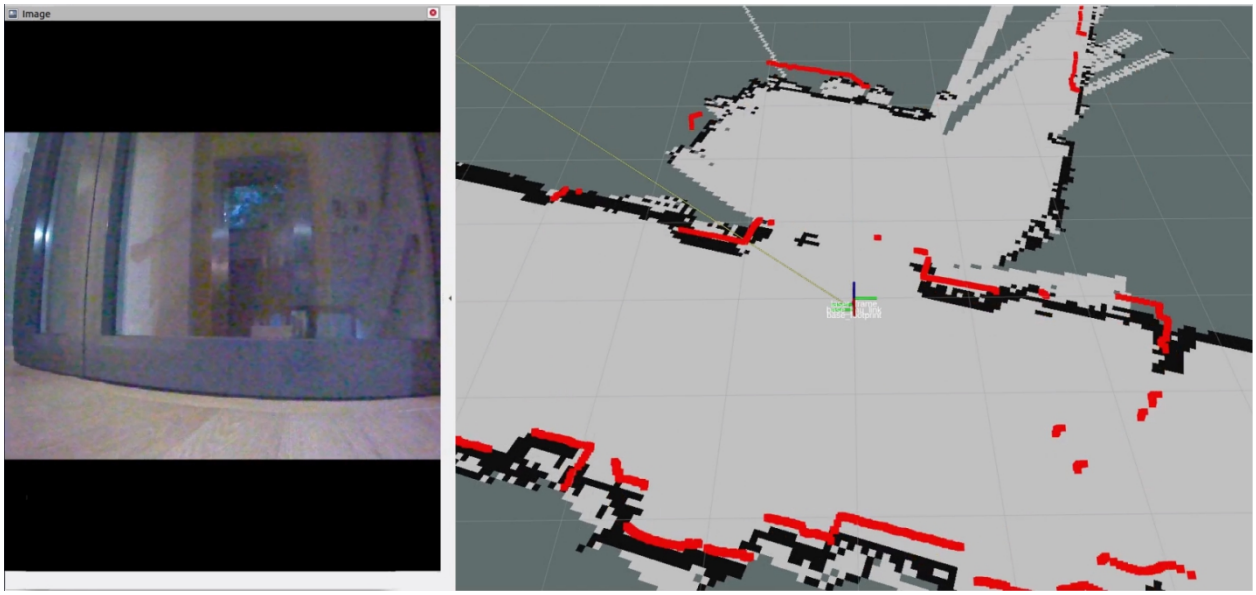

Рисунок 7.4 — Фрагмент візуалізацій сенсорів під час тестування алгоритму

І дійсно, уважно перевіривши дані з камери, ми можемо побачити на зображені скляні двері, які є зачинені під час тестування алгоритму, натомість на мапі, яка будується та судячи з даних лідаром, цей проміжок позначається як вільний для пересування. У даному випадку версія зпівпала і підтвердилась, проте це не завжди трапляється саме так, тому контроль та аналіз з боку людини, через відео потік або безпосередньо на місці залишається необхідним, як і саме завдання тестування подібних алгоритмів безпосередньо у фізичному середовищі.

Відтак, приміщення Б, у якому і проводилось тестування лідару відокремлене від приміщення А дверима, які побудовані зі скла, товщиною приблизно півтора сантиметри. Під час тестування двері були зачинені. Але особливість роботи лідару полягає в тому, що він може просвічувати і сканувати через скло, що може бути як недоліком, так і перевагою, проте у будь якому випадку потребує уваги та урахування цього факту, під час проектування подібних пристроїв для операцій у конкретних приміщеннях. Цей випадок також є прикладом, коли раціональним кроком було б замалювання чорним кольором отриманого отвору, якщо б планувалося подальше використання робота у цьому приміщенні. Повна мапа другого відсканованого приміщення виглядає наступним чином (рис. 7.5):

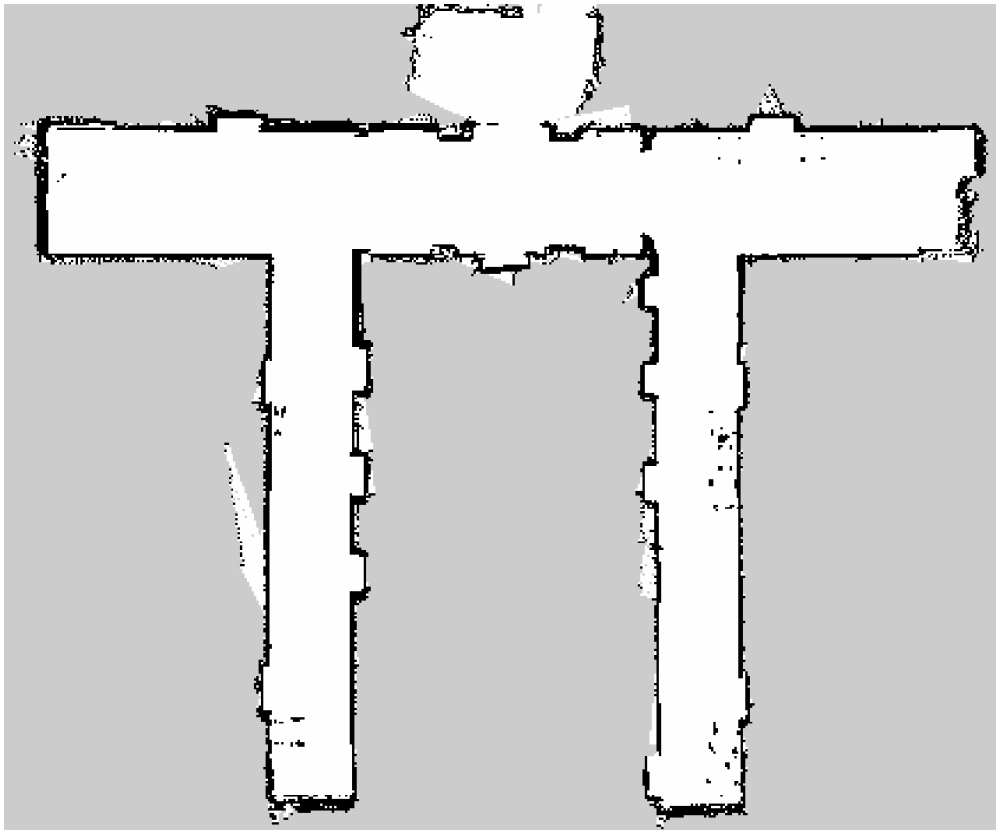

Рисунок 7.5 — Побудована мапа більшого приміщення

Слід також зазначити, що окрім пропорцій, мапа також зберігає в собі і реальні розміри. Це виражається за допомогою пікселів, тобто кожен піксель має певний розмір, який зазначений під час генерації.

При тестуванні алгоритму у даній роботі, була вказана роздільна здатність у 0.05 метрів на один піксель, або ж 5 см на один піксель. Таким чином, можна також виміряти розміри сканованого приміщення за допомогою мапи, з точністю до одиниці роздільної здатності, яка була застосована, із врахуванням похибки, яку може давати обраний алгоритм.

В результаті, ми отримали достатньо детальні мапи, із підходящою роздільною здатністю, яка здатна відображати необхідні перешкоди, при цьому не навантажуючи черезмірно систему великою кількістю даних.

ВИСНОВКИ

У результаті виконання роботи, були виконані наступні етапи:

- порівняні між собою доступні на ринку сенсори та обчислювальні модулі, та обрані таким чином, щоб їх характеристики давали змогу тестувати алгоритми одночасної локалізації та картографування;
- зроблений огляд популярних алгоритмів одночасної локалізації та картографування, а також описані принципи їх роботи;
- був спроектований та побудований мобільний автономний пристрій за допомогою обраних сенсорів та обчислювальних модулів;
- проведено встановлення та налаштування програмного забезпечення на пристрій, який був побудований;
- було проведено тестування алгоритму одночасної локалізації та картографування GMapping у реальному фізичному середовищі;
- проаналізовані отримані мапи приміщень.

Результатом виконаної роботи є спроектований та побудований мобільний автономний пристрій, який здатен використовуючи лише наявне у нього обладнання виконувати такі задачі, як тестування алгоритмів одночасної локалізації та картографування. Головними сенсорами робота є лідар та камера. Пристрій оснащений потужним обчислювальним модулей NVidia Jetson Nano. Його пересування забезпечується за допомогою колесної платформи. Комунікація між роботом та комп’ютером реалізована з використанням бездротових мереж стандарту IEEE 802.11, тобто Wi-Fi, за допомогою захищеного віддаленого протоколу SSH. Керування переміщенням робота відбувається за допомогою клавіш стрілок на клавіатурі комп’ютера. Налаштована зручна та наочна візуалізація даних з камери та лідару, яка працює в режимі реального часу. Результатом роботи робота після тестування алгоритму одночасної локалізації та картографування є мапа приміщення у вигляді зображення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Correll Nikolaus Introduction to Autonomous Robots: Kinematics, Perception, Localization and Planning / Bretl Tim , Boots Byron — Magellan Scientific, 2nd edition, 2016. — 241 с.
2. Thrun Sebastian Probabilistic Robotics (Intelligent Robotics and Autonomous Agents series) / Dieter Fox — The MIT Press, 2005. — 672 с
3. Wang Zhan Simultaneous Localization and Mapping: Exactly Sparse Information Filters / — World Scientific Publishing Company, 2011 — 208 с.
4. Модуль лідару у ROS / Режим доступа: — http://wiki.ros.org/rplidar
5. Драйвери двигунів у фреймворку ROS / Режим доступа: — http://wiki.ros.org/Motor%20Controller%20Drivers
6. Офіційна документація з фреймворку ROS [Електронна версія] / Режим доступа: — https://docs.ros.org/
7. Lentin Joseph Mastering ROS for Robotics Programming. Design, build, and simulate complex robots using the Robot Operating System / — Packt Publishing; 2nd edition, 2018 — 580 с.
8. Koubaa Anis Robot Operating System (ROS): The Complete Reference / — Springer, 1st edition, 2018 — 741 с.
9. Документація з пакету rviz / Режим доступа: — http://wiki.ros.org/rviz/UserGuide
10. Документація з модулю ROS алгоритму GMapping / Режим доступа: — http://wiki.ros.org/gmapping#slam_gmapping
11. Дрейвера камери у фреймворку ROS / Режим доступа: — http://wiki.ros.org/gscam
12. Siegwart Roland Introduction to Autonomous Mobile Robots / Illah Reza Nourbakhsh, Scaramuzza Davide — The MIT Press, 2nd edition, 2011 — 472 с.

ДОДАТОК А

Перелік необхідних бібліотек та залежностей

```
sydo apt-get install ros-melodic-desktop-full
sudo apt-get install python-rosdep
sudo apt-get install ros-melodic-gmapping
sudo apt-get install ros-melodic-cartographer-ros
sudo apt-get install ros-melodic-navigation
sudo apt-get install ros-melodic-teb-local-planner
sudo apt-get install gstreamer1.0-tools libgstreamer1.0-dev libgstreamer-
plugins-base1.0-dev libgstreamer-plugins-good1.0-dev
```

Рисунок А.1 — Команди для встановлення залежностей робота

```
sudo apt install ros-melodic-desktop-full
sudo apt install python-rosdep
sudo apt-get install python-rosinstall python-rosinstall-generator python-
wstool build-essential
sudo apt-get install ros-melodic-joy
sudo apt-get install ros-melodic-camera-calibration
sudo apt-get install python-pip
python -m pip install imutils
```

Рисунок А.2 — Команди для встановлення залежностей основної машини

ДОДАТОК Б

Побудовані мапи приміщень

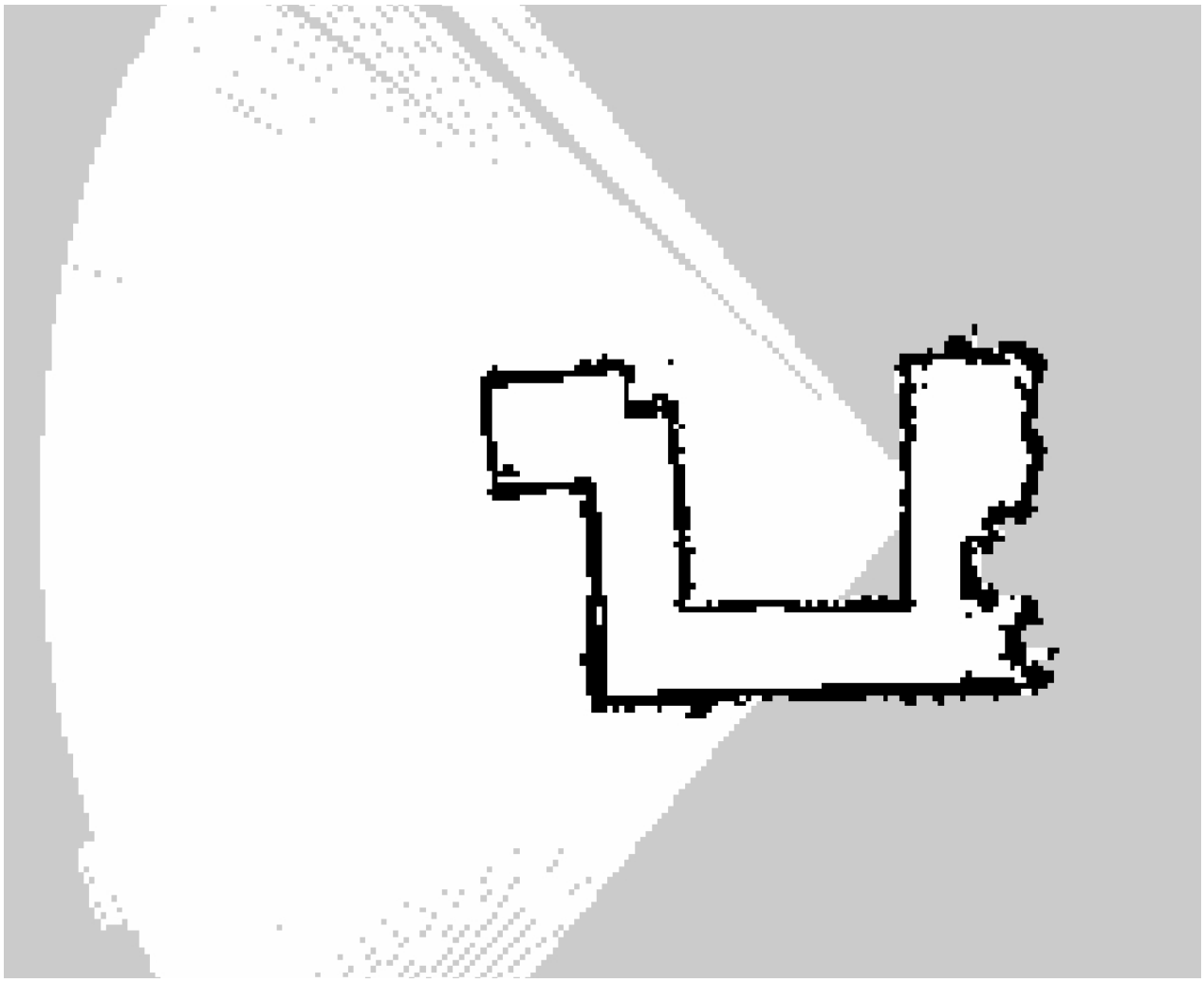

Рисунок Б.1 — Альтернативна мапа першого приміщення

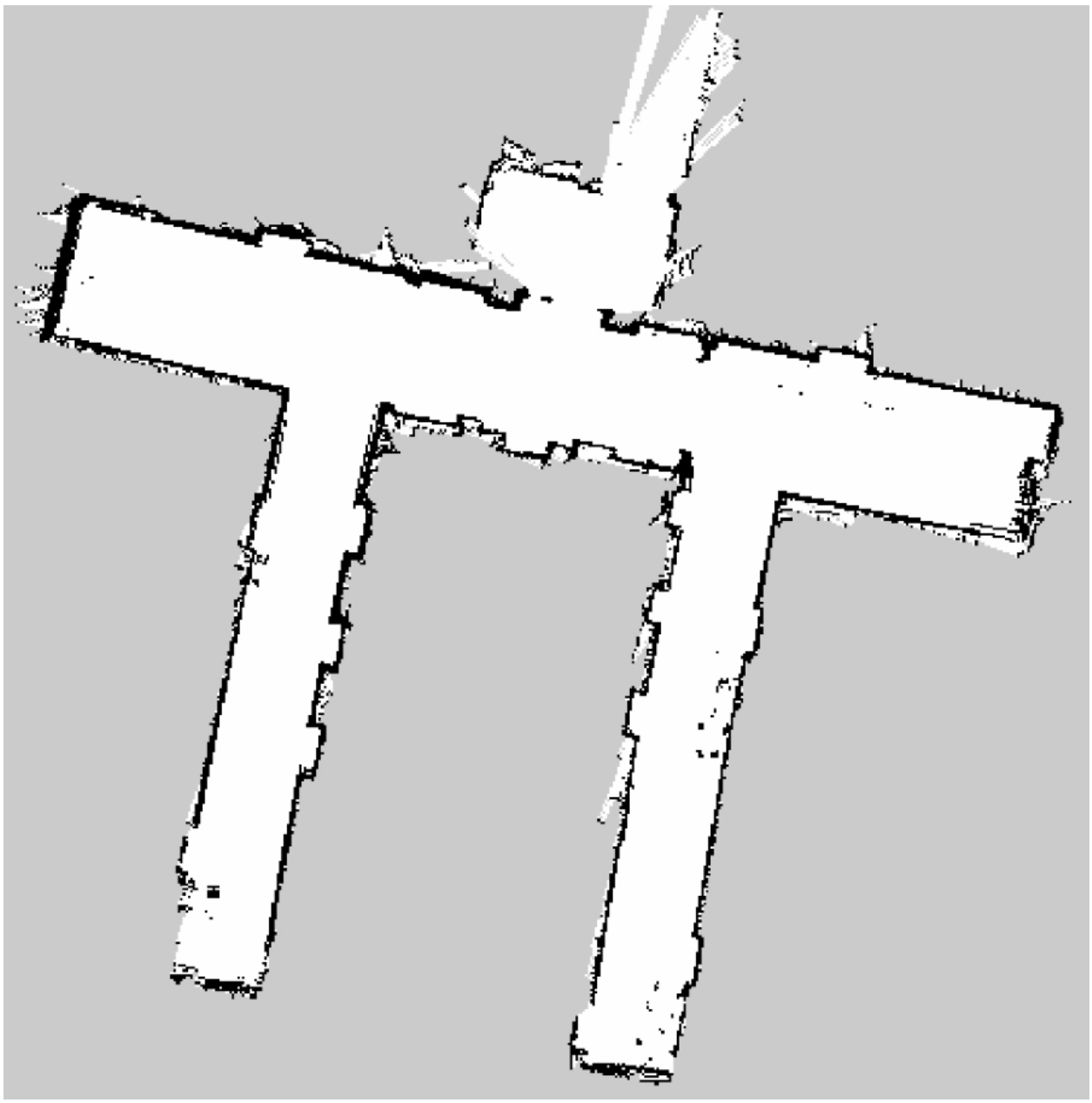

Рисунок Б.2 — Альтернативна мапа другого приміщення

ДОДАТОК В

Файли налаштувань для робота

```
<?xml version="1.0"?>
<launch>
  <param name="use_sim_time" value="false"/>
  <node name="jetbot" pkg="jetbot_pro" type="jetbot"
output="screen" respawn="true">
    <param name="port_name" value="/dev/ttyACM0"/>
    <param name="linear_correction" value="1.027"/>
    <param name="angular_correction" value="1.05"/>
    <param name="publish_odom_transform" value="false"/>
  </node>
     <node pkg="robot_pose_ekf" type="robot_pose_ekf"
name="robot_pose_ekf" output="screen">
    <param name="output_frame" value="odom"/>
          <param name="base_footprint_frame"
value="base_footprint"/>
    <param name="freq" value="30.0"/>
    <param name="sensor_timeout" value="0.5"/>
    <param name="odom_used" value="true"/>
    <param name="imu_used" value="true"/>
    <param name="vo_used" value="false"/>
    <param name="debug" value="false"/>
    <param name="self_diagnose" value="false"/>
    <remap from="/imu_data" to="/imu"/>
     <remap from="/robot_pose_ekf/odom_combined"
to="/odom_combined"/>
  </node>
     <node pkg="tf" type="static_transform_publisher"
name="base_footprint_to_imu" args="0 0 0.07 0 0 0 base_footprint
base_imu_link 20"/>
</launch>
```

Рисунок В.1 — Головний файл robot.launch

```
<?xml version="1.0"?>
<launch>
  <include file="$(find jetbot_pro)/launch/jetbot.launch" />
  <node name="calibrate_linear"  pkg="jetbot_pro"
type="calibrate_linear.py" output="screen" />
</launch>
```

Рисунок В.2 — Файл налаштування лінійної калібрації

```
<?xml version="1.0"?>
<launch>
    <arg name="device_id" default="0"/>
    <arg name="frame_id" default="camera_link"/>
    <arg name="image_width" default="640"/>
    <arg name="image_height" default="480"/>
    <arg name="camera_info_url" default="file://$(find
jetbot_pro)/cfg/cam_480p.yaml"/>

    <node name="camera" pkg="cv_camera" type="cv_camera_node"
output="screen">
      <param name="device_id" value="$(arg device_id)" />
      <param name="frame_id" value="$(arg frame_id)" />
      <param name="image_width" value="$(arg image_width)" />
      <param name="image_height" value="$(arg image_height)" />
      <param name="camera_info_url" type="string" value="$(arg
camera_info_url)"/>
      <remap from="/camera/image_raw/compressed"
to="/image_raw/compressed"/>
    </node>
</launch>
```

Рисунок В.3 — Файл налаштування камери

```
<?xml version="1.0"?>
<launch>
  <node name="rplidarNode"          pkg="rplidar_ros"
type="rplidarNode" output="screen">
    <param name="serial_port"         type="string"
value="/dev/ttyACM1"/>
    <param name="serial_baudrate"     type="int"
value="115200"/><!--A1/A2 -->
    <!--param name="serial_baudrate"     type="int"
value="256000"--><!--A3 -->
    <param name="frame_id"            type="string"
value="laser_frame"/>
    <param name="inverted"            type="bool"
value="false"/>
    <param name="angle_compensate"    type="bool"
value="true"/>
  </node>
  <node pkg="tf" type="static_transform_publisher"
name="base_link_to_laser"
    args="0 0.0 0.15 3.14 0.0 0.0 /base_footprint /laser_frame
20" />
</launch>
```

Рисунок В.4 — Файл налаштування лідару

```
<?xml version="1.0"?>
<launch>
  <include file="$(find jetbot_pro)/launch/jetbot.launch" />
  <node name="calibrate_angular"  pkg="jetbot_pro"
type="calibrate_angular.py" output="screen" />
</launch>
```

Рисунок В.5 — Файл налаштування кутової калібрації

```
<?xml version="1.0"?> <launch>
  <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping"
output="screen" respawn="true" >
    <remap from="scan" to="scan"/>
    <param name="base_frame" value="base_footprint"/>
    <param name="odom_frame" value="odom"/>
    <param name="map_update_interval" value="2.0"/>
    <param name="maxUrange" value="8.0"/>
    <param name="maxRange" value="12.0"/>
    <param name="sigma" value="0.05"/>
    <param name="kernelSize" value="1"/>
    <param name="lstep" value="0.05"/>
    <param name="astep" value="0.05"/>
    <param name="iterations" value="5"/>
    <param name="lsigma" value="0.075"/>
    <param name="ogain" value="3.0"/>
    <param name="lskip" value="0"/>
    <param name="minimumScore" value="100"/>
    <param name="srr" value="0.01"/>
    <param name="srt" value="0.02"/>
    <param name="str" value="0.01"/>
    <param name="stt" value="0.02"/>
    <param name="linearUpdate" value="0.05"/>
    <param name="angularUpdate" value="0.0436"/>
    <param name="temporalUpdate" value="-1.0"/>
    <param name="resampleThreshold" value="0.5"/>
    <param name="particles" value="8"/>
    <param name="xmin" value="-50.0"/>
    <param name="ymin" value="-50.0"/>
    <param name="xmax" value="50.0"/>
    <param name="ymax" value="50.0"/>
    <param name="delta" value="0.05"/>
    <param name="llsamplerange" value="0.01"/>
    <param name="llsamplestep" value="0.01"/>
    <param name="lasamplerange" value="0.005"/>
    <param name="lasamplestep" value="0.005"/> </node> </launch>
```

Рисунок В.6 — Файл налаштувань алгоритму GMapping

```
image_width: 640
image_height: 480
camera_name: csi_cam_0
camera_matrix:
  rows: 3
  cols: 3
  data: [400.2333557174174, 0, 331.4549671721432, 0,
533.2837800786184, 263.3567321479512, 0, 0, 1]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [-0.3099359644894822, 0.08151662693197913,
0.0006660916041993709, 0.001461474111496217, 0]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1, 0, 0, 0, 1, 0, 0, 0, 1]
projection_matrix:
  rows: 3
  cols: 4
  data: [301.4978637695312, 0, 340.7424712103602, 0, 0,
494.3556213378906, 266.7389488050048, 0, 0, 0, 1, 0]
```

Рисунок В.7 — Файл з налаштуваннями параметрів камери

ДОДАТОК Г

Файли налаштувань для головної машини

```
<?xml version="1.0"?>
<launch>
  <param name="use_sim_time" value="false"/>
  <arg name="map_type" default="gmapping" doc="opt: gmapping"/>
  <include file="$(find jetbot_pro)/launch/joy.launch" />
  <group if="$(eval arg('map_type') != 'cartographer')">
     <node pkg="rviz" type="rviz" name="rviz" args="-d $(find
jetbot_pro)/rviz/slam.rviz" />
  </group>
  <group if="$(eval arg('map_type') == 'cartographer')">
     <node pkg="rviz" type="rviz" name="rviz" args="-d $(find
jetbot_pro)/rviz/carto_slam.rviz" />
  </group>
</launch>
```

Рисунок Г.1 — Файл налаштувань для перегляду роботи SLAM алгоритму

```
<?xml version="1.0"?><launch>
  <param name="use_sim_time" value="false"/>
  <node pkg="joy" type="joy_node" name="joy_node" />
  <node pkg="jetbot_pro" type="teleop_joy.py"
name="teleop_joy_node" output="screen">
     <param name="x_speed"  value="0.3" />
     <param name="y_speed"  value="0" />
     <param name="w_speed"  value="1" />
   </node></launch>
```

Рисунок Г.2 — Файл налаштувань для контролю робота через джойстик

```
<?xml version="1.0"?><launch>
    <param name="use_sim_time" value="false"/>
    <node pkg="jetbot_pro" type="teleop_key.py"
name="teleop_keyboard_node" output="screen">
     <param name="speed"  value="0.3" />
     <param name="turn"  value="1" />
    </node>
</launch>
```

Рисунок Г.3 — Файл налаштувань для контролю робота клавіатурою