МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет

імені Петра Могили

Факультет комп'ютерних наук

Кафедра комп'ютерної інженерії

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри,
д-р техн. наук, проф.

_____ І. М. Журавська

« __ » _____ 2023 р.

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Апаратно-програмний комплекс 3D-сканування об'єктів на базі Arduino

Спеціальність 123 Комп'ютерна інженерія

123 – КБР.ПЗ.00 – 405.21910506

Студент _____Д. В. Варанкін
підпис
«__» _____ 202__ р.

Керівник ст. викладач _____ Є. С. Дарнапук
підпис
«__» _____ 202__ р.

Миколаїв – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет імені Петра Могили
Факультет комп'ютерних наук
Кафедра комп'ютерної інженерії

ЗАТВЕРДЖУЮ

Зав. кафедри _____ І. М. Журавська

«_____» _____ 2023 р.

ЗАВДАННЯ
на виконання кваліфікаційної бакалаврської роботи

Видано студенту групи 405 факультету комп'ютерних наук

Варанкін Денис Володимирович _____
(прізвище, ім'я, по батькові студента)

1. Тема кваліфікаційної роботи
__Апаратно-програмний комплекс 3D-сканування об'єктів на базі Arduino

Затверджена наказом по ЧНУ від «___»_____20___ р. №_____

2. Строк представлення кваліфікаційної роботи «____» _____ 20__ р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні
___Очікуваним результатом роботи є: апаратне та програмне забезпечення 3D-сканера. Вхідними даними роботи є специфікація вимог, що описує характеристики зазначеного апаратного та програмного забезпечення. _____
_____
_____
_____
_____

4. Перелік питань, що підлягають розробці
1) дослідження сучасного ринку 3D-сканерів;____ _____
2) аналітичний огляд літератури та технологій 3D-сканування;_____
3) вибір електронних компонентів;_____
4) розроблення програмного забезпечення для керування 3D-сканер;_____
5) розроблення апаратного забезпечення 3D-сканеру._____

5. Перелік графічних матеріалів

_слайди презентації, принципові схеми обраного обладнання, блок-схема_____
алгоритму роботи програмно-апаратного комплексу, приклади роботи 3D-__
сканерів, алгоритм калібрування камери, калібрувальні шаблони_____

6. Завдання до спеціальної частини

_____
_____
_____

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи
Спеціальна частина з охорони праці	Алексєєва Анна Олександрівна, канд. техн. наук, доцент кафедри екології Медичного інституту ЧНУ імені Петра Могили	

Керівник роботи

_ст. викладач Дарнапук Євген Сергійович_____

(посада, прізвище, ім'я, по батькові)

_____

(підпис)

Завдання прийнято до виконання

_Варанкін Денис Володимирович _____

(прізвище, ім'я, по батькові студента)

_____

(підпис)

Дата видачі завдання «____» _____ 20_____ р.

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи

Тема: Апаратно-програмний комплекс 3D-сканування об'єктів на базі Arduino

№	Найменування роботи	Початок	Закінчення	Примітки
1	Розробка та затвердження завдання на виконання КР	11.12.22	12.12.22	Виконав
2	Огляд літератури за темою роботи	15.01.23	18.02.23	Виконав
3	Складання календарного плану БКР	19.02.23	06.03.23	Виконав
4	Аналіз предметної області	19.02.23	04.03.23	Виконав
5	Розробка проєктних рішень	23.02.23	09.03.23	Виконав
6	Моделювання та конструювання АПЗ	20.02.23	27.02.23	Виконав
7	Перевірка працездатності, тестування та апробація розробленого АПЗ, аналіз результатів тестування	01.03.23	04.03.23	Виконав
8	Відгук керівника КР	09.03.23	07.04.23	Виконав
9	Оформлення БКР та презентації	15.03.23	21.04.23	Виконав
10	Попередній захист	13.06.23	13.06.23	Виконав
11	Рецензування	12.05.23	20.05.23	Виконав
12	Завершення оформлення КР та презентації	30.05.23	30.05.23	Виконав
13	Захист бакалаврської кваліфікаційної роботи	27.06.23		

Розробив здобувач ВО ____Варанкін Денис Володимирович _____ ___

(прізвище, ім'я, по батькові) *(підпис)*

«____» _____ 2023 р.

Керівник роботи _ст. викладач Дарнапук Є. С._____ _____

(посада, прізвище, ім'я, по батькові) *(підпис)*

«____» _____ 2023 р.

АНОТАЦІЯ

до кваліфікаційної бакалаврської роботи
«Апаратно-програмний комплекс 3D-сканування об'єктів на базі Arduino»
Студент 405 гр.: Варанкін Денис Володимирович
Керівник: ст. викладач Дарнапук Євген Сергійович

Об’єкт роботи – методи та технології 3D-сканування.

Предмет роботи – програмно-апаратний модуль 3D-сканеру на базі Arduino uno, крокового мотору NEMA 17 jk42hs34-1334ac, драйвера крокового двигуна l298n та лінійного лазеру.

Метою бакалаврської кваліфікаційної роботи є розроблення апаратно програмного комплексу 3D-сканування об’єктів на базі Arduino. Що зменшує загальну вартість, спрощує використання та покращує швидкість сканування.

Робота складається з фахового розділу і спеціальної частини з охорони праці. Пояснювальна записка складається зі вступу, трьох розділів, висновків та додатків.

У першому розділі проаналізовано об’єкт та предмет дослідження, і виконано аналітичний огляд аналогів та методів їх сканування та технологій.

У другому розділі описано алгоритм сканування та калібрування камери для 3D-сканера.

У третьому розділі було проведено вибір електронних компонентів, збірку пристрою та розроблення програмної частини пристрою. В результаті було прийнято рішення використовувати мову програмування Python та фреймворки cv2 open3d.

Результатом роботи став пристрій з програмною і апаратною частиною для 3D-сканування.

Бакалаврська кваліфікаційна робота викладена на n сторінках, містить n розділів, n ілюстрацій, n схем та n джерел у переліку посилань.

Ключові слова: *3D-сканування, безконтактне сканування, структуроване світло, лазерне сканування, поверхневий сканер, точковий хмара, моделювання об'єктів, геометричне відтворення, розпізнавання форми, реконструкція 3d-моделі*

ABSTRACT
of the Bachelor’s Thesis
"Hardware and software complex for 3D scanning of objects based on Arduino"
Student: Varankin Denys Volodymyrovych
Supervisor: Senior teacher Darnapuk Yevhen Serhiyovych

The object of work – methods and technologies of 3D scanning.

The subject of work – software-hardware module of a 3D scanner based on Arduino uno, NEMA 17 jk42hs34-1334ac stepper motor, l298n stepper motor driver and linear laser.

The purpose of the bachelor's qualification work is to develop a hardware and software complex for 3D scanning of objects based on Arduino. Which reduces the total cost, simplifies the use and improves the scanning speed.

The work consists of a professional section and a special part on labor protection. The explanatory note consists of an introduction, three sections, conclusions and appendices.

In the first section, the object and subject of the research is analyzed, and an analytical review of analogues and their scanning methods and technologies is performed.

The second section describes the scanning algorithm and camera calibration for the 3D scanner.

In the third section, the selection of electronic components, the assembly of the device and the development of the software part of the device were carried out. As a result, it was decided to use the Python programming language and cv2 open3d frameworks.

The result of the work was a device with a software and hardware part for 3D scanning.

Keywords: *3D scanning, non-contact scanning, structured light, laser scanning, surface scanner, point cloud, object modeling, geometric rendering, shape recognition, 3d model reconstruction*

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ПК – персональний комп’ютер

3D – 3-dimensional

CAD – Computer-aided design

DPI – Dots Per Inch

FOV – Field of View

GUI – Graphical user interface

LIDAR – Light identification

PC – Point cloud

STL – Stereolithography

USB – Universal Serial Bus

XYZ – Координати X, Y і Z

ВСТУП

У різних галузях людської діяльності технології 3D-друку здобувають популярність разом з 3D-сканерами, які дозволяють сканувати різноманітні фізичні об'єкти та отримувати їх тривимірні цифрові моделі з високою точністю. Перші 3D-сканери з'явилися наприкінці 20 століття, але мали обмежені можливості. Зараз 3D-сканери можуть швидко та якісно сканувати будь-який об'єкт.

Останнім часом актуальність створення складних тривимірних моделей постійно зростає, оскільки отримані 3D-моделі можуть бути використані у будівництві, архітектурі, дизайні, медицині, прототипуванні та реверсному інжинірингу, авіабудуванні.

Використання 3D-сканування на виробництві має кілька переваг, таких як низька вартість сканування, можливість використання при різному освітленні, бесконтактний процес сканування та відсутність необхідності встановлювати сітку на об'єкт.

Зараз можна знайти сканери різної форми, з різними камерами та мікроконтролерами та використовуваною керуючою програмою. Вартість сканерів залежить від раніше вказаних параметрів. Тому актуальною проблемою на сьогоднішній день є створення сканера, який складається з недорогих комплектуючих та має великий діапазон можливостей.

Мета: розроблення апаратно програмного комплексу 3D-сканування об’єктів на базі Arduino.

Об’єкт: методи та технології 3D-сканування.

Предмет: програмно-апаратний модуль 3D-сканеру на базі Arduino uno, крокового мотору NEMA 17 jk42hs34-1334ac, драйвера крокового двигуна l298n та лінійного лазеру.

Поставлена мета вимагає вирішеннях таких **завдань**:

1) дослідження сучасного ринку 3D-сканерів
2) аналітичний огляд літератури та технологій 3D-сканування;
3) вибір електронних компонентів;

4) розроблення програмного забезпечення для керування 3D-сканера;

5) розроблення апаратного забезпечення 3D-сканеру.

Практичне значення одержаних результатів полягає у створенні 3D-сканеру, що має високу точність сканування та низьку ціну.

1 ВИВЧЕННЯ ТЕХНОЛОГІЇ 3D-СКАНУВАННЯ

1.1 Опис 3D-сканера

3D-сканер [1], [17] є спеціальним технічним пристроєм, який аналізує фізичний об'єкт або простір з метою отримання інформації про його форму та в залежності від випадку – зовнішній вигляд. Отримані дані потім використовуються для створення цифрової 3D-моделі просканованого об'єкта або простору.

Існує кілька технологій, які дозволяють створити 3D-сканер з різними перевагами, недоліками та вартістю. Крім того, є певні обмеження на об'єкти, які можуть бути зісковані. Зокрема, виникають труднощі з блискучими, прозорими або дзеркальними поверхнями предметами.

3D-сканери діляться на декілька типів, а саме:

1) активні 3D сканери, які випромінюють на об'єкт пучок спрямованих хвиль і фіксують їх відображення. До можливих типів випромінювання відносяться рентгенівські промені, ультразвук і світло;

2) пасивні 3D сканери, які фіксують відбите від об'єкта навколишній випромінювання, найчастіше світло.

Кожен метод сканування 3D має свої переваги та недоліки, які впливають на вартість, швидкість та точність сканування, а також на доступність для сканування різних типів об'єктів. У цьому випадку потрібно як умога зменшити ціну та при цьому не втратити в якості сканування [4].

1.2 Функціональні можливості

Мета 3D-сканера полягає в створенні хмари геометричних точок на поверхні об'єкта [2]. Пізніше ці отримані точки можна використовувати для відтворення форми предмета. Цей процес називається реконструкцією. Якщо 3D-сканер може отримувати кольори об'єкта, то колір реконструйованої поверхні також можна визначити.

3D-сканери трохи нагадують звичайні камери. Наприклад, вони мають конусоподібне поле зору, а також отримують інформацію тільки з тих поверхонь, які не затемнені або не блищать. Основна відмінність полягає в тому, що камера передає лише кольори поверхні об'єкта, а 3D-сканер збирає інформацію про відстані на поверхні. "Зображення", отримане 3D-сканером, описує відстань до поверхні в кожній точці зображення. Завдяки цьому можна отримати дані про розташування в тривимірному просторі кожної точки на зображенні [7].

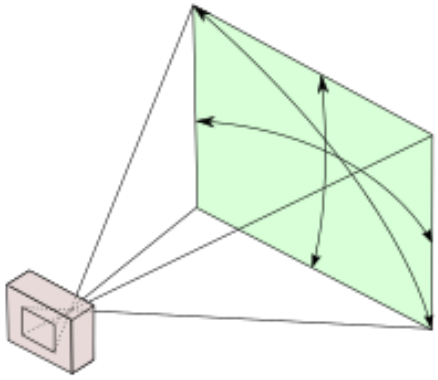


Рисунок 1.1 – Поле зору камери

При використанні 3D-сканерів часто виникає потреба в декількох скануваннях, щоб створити повноцінну модель об'єкта. Зазвичай для отримання інформації про всі сторони об'єкта необхідна велика кількість сканувань з різних напрямків. Отримані результати сканування повинні бути прив'язані до загальної системи координат, цей процес називається вирівнюванням. Після завершення вирівнювання створюється повна модель. Весь процес, починаючи зі створення простої карти з відстанями до повноцінної моделі, називається 3D-конвеєром сканування.

Таблиця 1.1 – Основні характеристики при 3D-скануванні

Характеристика	Опис	Від чого залежить
Роздільна здатність	Мінімальна відстань між сусідніми 3D-точками в міліметрах, еквівалентно дозволу фотографії	Залежить від розміру області сканування від дозволу камер та принципу роботи сканера
Деталізація	Можливість якісно передавати форму дрібних елементів поверхні.	Сканери різних виробників мають різну деталізацію. Ручні сканери зазвичай мають меншу деталізацію по порівняно зі стаціонарними
Рівень шуму	Випадкова складова загальної помилки виміру. Може бути оцінена при повторному скануванні одного і того ж об'єкта в тих же самих умовах та порівнянні з першим результатом.	Залежить від великої кількості факторів. Наприклад, якості поверхні об'єкта, зовнішнього освітлення, положення камер, напрями сканування тощо.
Точність	Загальна результуюча помилка вимірювання.	Містить як випадкову, так і систематичні складові.

1.3 Технологія

Перший етап при створенні будь-якої моделі – це огляд того, що вже створено на даний момент. Розглянемо можливі варіанти щодо 3D-сканерів [13].

Існує кілька технологій для цифрового сканування форми та створення 3D-моделі об'єкта. Однак була розроблена спеціальна класифікація, яка поділяє 3D-сканери на два типи:

– контактні [16];

– безконтактні.

Кожен тип має свої особливості, від яких залежить, де вони можуть застосовуватись.

1.4 Контактні 3D-сканери

Контактні 3D-сканери досліджують об'єкт безпосередньо за допомогою фізичного контакту, тоді як сам предмет знаходиться на поверхні для перевірки точності, яка була підготовлена до певної ступені шорсткості. Якщо об'єкт для сканування нерівний або не може стійко лежати на горизонтальній поверхні, його будуть утримувати спеціальні пристрої [14].

Існують кілька технологій для цифрового сканування форми та створення 3D-моделі об'єкта. 3D-сканери можна поділити на 2 типи:

– контактні 3d-сканери сканують об'єкт безпосередньо через фізичний контакт, якщо об'єкт складно підтримувати на плоскій поверхні, його можуть утримувати спеціальні тискання;

– безконтактні 3d-сканери можуть сканувати об'єкт за допомогою світла, радіохвиль або звуку, що відбивається від поверхні об'єкта, і збирається спеціальними сенсорами.

Технології контактних 3D-сканерів можуть використовувати каретку з фіксованою вимірювальною рукою, що розташована перпендикулярно, та маніпулятор з фіксованими складовими та високоточними кутовими датчиками [15]. Іноді можна використовувати комбінацію обох методів. Каретка з вимірювальною рукою може бути оптимальною для сканування плоских або звичайних випуклих кривих поверхонь, тоді як маніпулятор з кутовими датчиками можна використовувати для зондування виривів або внутрішніх просторів з невеликим вхідним отвором. Також можна поєднувати обидва методи, наприклад, використовуючи маніпулятор разом з кареткою для отримання 3D-даних від великих об'єктів з внутрішніми порожнинами або перекриваючими поверхнями.

КВМ (рис. 1.1) є яскравим прикладом контактного 3D-сканера.

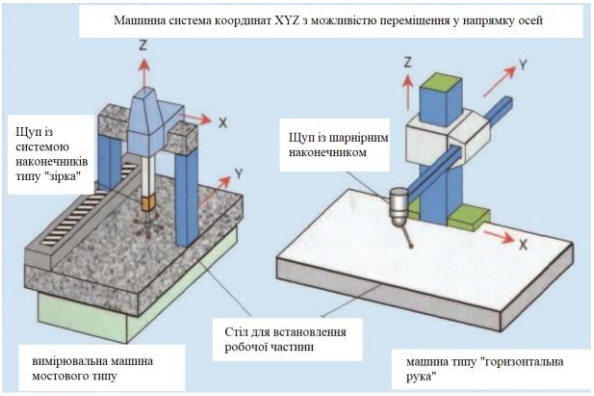


Рисунок 1.2 – Конструкція координатно-вимірювальної машини

Ці сканери переважно використовуються у виробництві і можуть бути надзвичайно точними. Контактні 3D-сканери прості у використанні та підходять для обробки об'єктів з простою геометрією. Тому вони успішно використовуються в промисловості. Часто виробники фрезерних та гравіювальних машин додають контактні сканери як додаткову опцію до верстатів.

Однак їх використання ускладнене, коли об'єкти мають багату деталізацію, а форма контурів замість чітко окреслених граней включає плавні лінії. У цьому випадку сканування може займати надто багато часу. До недоліків КВМ можна віднести необхідність безпосереднього контакту з поверхнею об'єкта. Тому існує можливість змінити предмет або навіть пошкодити його. Це дуже важливо у випадку, якщо сканується тонкий або цінний предмет, наприклад, історичні артефакти. Ще один недолік КВМ порівняно з іншими методами сканування – повільність. Переміщення вимірювальної руки з встановленим зондом може бути дуже повільним. Найшвидший результат роботи КВМ не перевищує декількох сотень герц. У той же час, оптичні системи, наприклад, лазерний сканер, може працювати від 10 до 500 кГц.

1.5 Контактні 3D-сканери

У сучасному світі найбільш популярними є сканери, які використовують безконтактний метод сканування. Цей варіант взаємодії вважається найбільш

перспективним, оскільки дозволяє створювати 3D-візуалізацію моделей, які знаходяться у важкодоступних місцях або є хрупкими. Існує кілька видів безконтактних 3D-сканерів, які відрізняються застосованою для їх роботи технологією:

– фотограмметричні;
– на основі структурованого білого кольору;
– лазерні.

1.5.1 Триангуляційні лазерні 3D-сканери

Триангуляційні лазерні 3D-сканери також відносяться до активних сканерів, які використовують лазерний промінь, щоб отримати інформацію про форму сканованого об'єкта [12, 13]. Так само, як і часові 3D-сканери, триангуляційні пристрої проектують лазерний промінь на об'єкт сканування, а окрема камера фіксує розташування точки лазера на поверхні об'єкта (рис. 1.3)

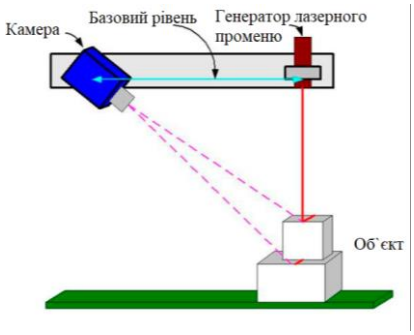


Рисунок 1.3 – Принцип роботи лазерної тріангуляції

Рисунок 1.4 – Приклад професійного ручного лазерного сканера

У більшості випадків для прискорення процесу отримання даних використовують лазерну смугу замість лазерної точки. Залежно від того, наскільки далеко лазер продовжується по поверхні, точка з'являється в різних місцях в полі зору камери. Ця технологія називається триангуляційною тому, що лазерна точка, камера та сам лазерний випромінювач утворюють свого роду трикутник. Довжина однієї сторони цього трикутника – відстань між камерою та лазерним випромінювачем – відома. Також відомий кут лазерного випромінювача. Кут камери можна визначити за місцезнаходженням лазерної точки в полі зору камери. Ці 3 компоненти повністю визначають розмір і форму трикутника та вказують на місцезнаходження кута лазерної точки.

Саме цей спосіб з невеликими доповненнями буде використовуватись для побудови власного 3D-сканеру.

1.6 Огляд аналогів

Були розглянуті найпопулярніші моделі 3D-сканерів, які мають різні зовнішній вигляд і можливості функціоналу. Основними параметрами під час огляду наявних моделей є:

– вартість;
– час та точність сканування;
– кількість та роздільна здатність камер;
– наявність платформи, що обертається;
– відкритість проекту;
– область сканування.

1. BQ Ciclop [26]

Сканер моделі BQ Ciclop є представником проектів з відкритим вихідним кодом, тобто проектів, код яких доступний для всіх. Він поставляється з програмним забезпеченням Horus, а вся інформація про його дизайн, програмне забезпечення та електронні компоненти є безкоштовною. Характеристики 3D-сканера наведені в таблиці 1.2, а реалізація 3D-сканера зображена на рисунку 1.5.

Таблиця 1.2 – Параметри 3D-сканера BQ Ciclop

Модель	BQ Ciclop
Метод сканування	Лазерний промінь
Джерело світла	Лазер класу 1
Точність	0.5-5мм від розмірів деталі
Область сканування	205 мм
Наявність платформи	Так
Час сканування	2-8 хв
Кількість камер	1
Кількість мегапікселів	1,3 Мп
OpenSource проєкт	Так
Середня ціна	8 000 грн

Рисунок 1.5 – 3D-сканер BQ Ciclop

2. David Laserscanner [25] – німецький виробник, який створив кілька моделей 3D-сканерів. Стартовий комплект V2 є початковою версією та базується на технології лазерної триангуляції. Сканер поставляється з програмним забезпеченням David-4, яке об'єднує різні грані об'єкта після його сканування. Параметри 3D-сканера наведені в таблиці 1.3, реалізація 3D-сканера показана на рисунку 1.6.

Таблиця 1.3 – Параметри 3D-сканера David Starter Kit V2

Модель	David Laserscanner
Метод сканування	Лазерний промінь
Джерело світла	Лазер класу 1
Точність	0,5 від розмірів деталі
Область сканування	10-600 мм
Наявність платформи	Ні
Час сканування	40 сек
Кількість камер	1
Кількість мегапікселів	2 Мп
OpenSource проєкт	Ні
Середня ціна	10 000 грн

Рисунок 1.6 – 3D-сканер David Starter Kit V2

3.Digitizer є одним з перших тривимірних сканерів для споживачів, що з'являються на ринку. Розроблений командою MakerBot, він базується на процесі лазерної триангуляції, який включає 1,3-мегапіксельний CMOS-сенсор [24]. Характеристики 3D-сканера представлені в таблиці 1.4, а реалізація – на рисунку 1.7.

Таблиця 1.4 – Параметри 3D-сканера MakerBot Digitizer

Модель	Digitizer
Метод сканування	Лазерний промінь
Джерело світла	Лазер класу 1
Точність	2 мм
Область сканування	205 мм
Наявність платформи	Так
Час сканування	12 хв
Кількість камер	1
Кількість мегапікселів	1,3 Мп
OpenSource проєкт	Ні
Середня ціна	15 000 грн

Рисунок 1.7 – 3D-сканер MakerBot Digitizer

При розмові про 3D-сканери не можна не згадати про триангуляційні лазерні датчики. Датчики призначені для безконтактного вимірювання та контролю положення, розмірів, профілю поверхні, деформацій, вібрацій, сортування, розпізнавання технологічних об'єктів; вимірювання рівня рідин та сипучих матеріалів

4. Лазерний датчик SK603

SK603 є лазерним тріангуляційним 2D-сканером з вбудованою мікропроцесорною системою управління. Технічні характеристики сканера наведені в таблиці 1.5, а сам сканер зображений на рис. 1.8.

Таблиця 1.5 – Параметри лазерного датчика SK603

Модель	SK603
Метод сканування	Лазерний промінь
Джерело світла	Лазер класу 2
Точність	0.05 мм
Область сканування	205 мм
Наявність платформи	Так
Час сканування	12 хв
OpenSource проєкт	Ні
Середня ціна	1500 грн

Рисунок 1.8 – Лазерний датчик SK603

5. Лазерный датчик LS2D

Сканери моделі LS2D призначені для виміру параметрів різноманітних об'єктів, включаючи профіль з розсіювальною поверхнею, ширину, товщину металопрокату, діаметри, розпізнавання дефектів, контроль зазорів та зварних швів, побудови 3D-моделей та використання в різних вимірювальних системах. Параметри лазерного датчика наведені в таблиці 1.6.

Таблиця 1.6 – Параметри лазерного датчика LS2D

Модель	LS2D
Метод сканування	Лазерний промінь
Джерело світла	Лазер класу 2
Точність	0.05 мм
Область сканування	205 мм
Наявність платформи	Так
Кількість камер	1
Час сканування	12 хв
OpenSource проєкт	Ні
Середня ціна	1500 грн

Після аналізу наявних 3D-сканерів та лазерних датчиків можна зробити висновок, що їх точність сканування зазвичай не перевищує 5 мм, тому прототип з точністю близько 3 мм буде вважатися добрим результатом. Область сканування розглянутих приладів зазвичай складає близько 200 мм, але було встановлено, що прийнятна область сканування повинна бути не менше 300 мм [5]. Прототип матиме одну камеру яка буде під’єднуватись до ПК, щоб забезпечити кращу якість сканування. Приблизний час сканування для прототипу очікується від 2 до 8 хвилин, залежно від області сканування. На основі аналізу були сформульовані технічні вимоги до створення прототипу 3D-сканера, які наведені у таблиці 1.7.

Таблиця 1.7 – Технічні вимоги до прототипу

Метод сканування	Лазерний промінь
Джерело світла	Лазер класу 1
Точність	3 мм
Область сканування	Не меньше 300 мм
Наявність платформи	Тільки для повороту лазера
Час сканування	2-10 хв
Кількість камер	1
Кількість мегапікселів	Залежить від камери
Середня ціна	Не повинна перевищити 2000 грн

Ці технічні вимоги до прототипу до прототипу є основними, тому що вони описують основні характеристики які потрібні для розробки

Висновки до розділу 1

З представленої інформації можна дійти висновку, що є кілька видів 3D-сканерів, кожен із яких використовує свою технологію для сканування об'єктів. Активні 3D-сканери, такі як лазерні та структурні світлові сканери, проектують на об'єкт сканування лазерний промінь або структуроване світло, а потім використовують камеру для вимірювання відстані між об'єктом та сканером. Безконтактні 3D-сканери використовують методи, такі як стереофотограмметрія, проміжні та фазові методи для створення точної 3D-моделі об'єкта.

Кожен вид 3D-сканера має свої переваги та недоліки, які можуть бути важливими для конкретних програм. У той час як активні 3D-сканери зазвичай більш точні, вони можуть бути більш складними у використанні і дорожчими. Безконтактні 3D-сканери, з іншого боку, можуть бути більш доступними та простими у використанні, але вони можуть не забезпечувати таку ж точність.

3D-сканери знайшли широке застосування у різних галузях, включаючи медицину, інженерну справу, архітектуру, дизайн, виробництво тощо.

У медицині 3D-сканування застосовується для створення точних моделей органів та тканин пацієнтів, що дозволяє краще планувати операції та зменшити ризики. Також 3D-сканери використовуються для створення протезів а також для дослідження деформацій скелета та м'язів.

В інженерній справі 3D-сканування допомагає створювати точні моделі деталей, що дозволяє скоротити час та витрати на проектування та виробництво. Також 3D-сканери використовуються для контролю якості та виявлення дефектів.

Таким чином, 3D-сканери є потужним інструментом для отримання точних і детальних моделей об'єктів і поверхонь, що дозволяє скоротити час і витрати на проектування, виробництво та контроль якості, а також покращити результати в медицині та інших галузях, але вони обмежено доступні для загальних мас користувачів.

2 МАТЕМАТИЧНІ МЕТОДИ

2.1 Калібрування камери

Калібрувальний шаблон на рисунку 2.1 містить шахову дошку з відсутніми плитками для визначення правильної орієнтації. Шахова дошка знаходиться всередині чорного прямокутника з контуром. Пізніше буде показано, як ці дані дозволяють визначити внутрішні параметри проекції камери та її спотворення.

Деякі підходи вже були запропоновані для сканування 3D поверхонь з використанням структурованого світла, такі як [5], [12], [3], [8], [9]. Підхід, представлений у [5], наприклад, схожий но той що був використаний, оскільки він використовує тінь, кинуту від ручного металевого стержня.

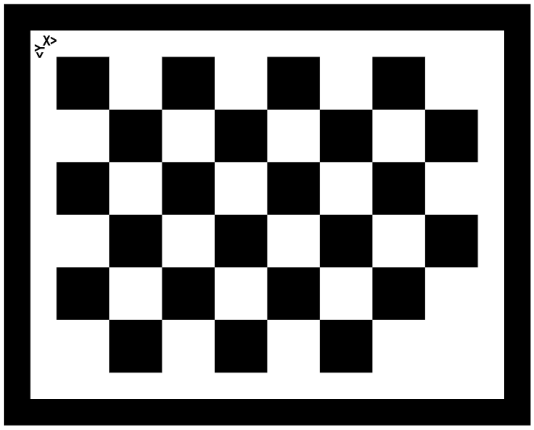


Рисунок 2.1 – Калібрувальний шаблон

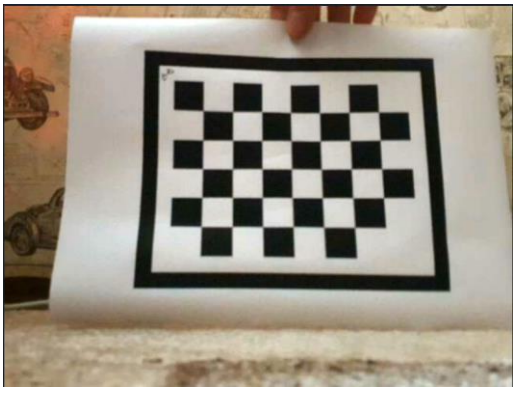


Рисунок 2.2 – Приклад 1 із 50 зображень
використовується для калібрування камери

Для отримання кадрів відео для 3D-реконструкції використовується наступне налаштування [10]:

– два перпендикулярні маркери з надрукованою паперовою площею розміром 23x13 см (файл з друкованим патерном можна знайти в репозиторії коду);

– фіксована камера rgb 1296 × 972 (модель невідома);

– червона лазерна лінійка, яка здатна перетинати обидва маркери та об’єкт;

– 3d-об’єкт для сканування, розміщений між двома маркерами.

Можна побачити, що всі компоненти обладнання дуже дешеві та легко знаходяться, на відміну від дорожчих стандартних рішень в промисловості. Крім того, система бачення є монокулярною: і тому не потрібна стереокалібрування, точне позиціонування та синхронізація відео.

Наданий набір даних складається з 4 відео загальною тривалістю 234 секунди (15 кадрів за секунду, загалом 990 кадрів). Усі відео використовують однакове обладнання та захоплюють кілька проходів лазером, зроблених за допомоги девайсу, після декількох секунд без лазера в полі зору. Буде видно, як перетин лазера з об'єктом та маркерами може допомогти відтворити 3D хмару точок данного об'єкта.

Рисунок 2.3 – Процес роботи лазера

– площина землі (δ), визначена чорною межею друкованого прямокутника;

– вертикальна площина (γ), визначена чорною межею іншого друкованого прямокутника;

– площина лазера (μ), визначена всіма точками, які освітлюються лазером.

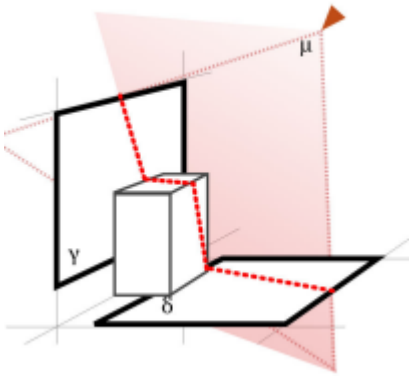


Рисунок 2.4 – Три площини, утворені маркерами та лазером

У відео лазерна площина перетинає дві маркерні прямокутники та сканований об’єкт у кількох точках [6]. Ми знаємо 2D проекцію цих точок перетину з відеокадрів, оскільки камера може зловити лазерне світло, а поверхні можуть відбивати його до сенсора. Ми можемо намагатися використовувати цю інформацію, щоб визначити 3D позицію кожної точки.

Ми можемо описати рівняння для проекційної точки перетину роздільними координатами та рівнянням для камери з отвором ([18]). У цьому рівнянні K – матриця параметрів камери, RT – представляє жорсткий рух конкретний для точки зору камери (зовнішні параметри), а вектор (x, y, z) – це позиція точки проекції в глобальних координатах. Ми перетворюємо 3D точку в гомогенний 4D вектор, доповнюючи 1 в кінці, щоб дозволити нам виконувати проективні перетворення.

$$p = K(RT)\begin{pmatrix} x_W \\ y_W \\ z_W \\ 1 \end{pmatrix} \quad (2.1)$$

Щоб встановити відповідність між позиціями точок в реальному світі та їх позиціями на зображенні, отриманому від камери, можна перетворити координати 4D точок, отриманих в проекті, на координати в декартовій системі координат за допомогою ділення на останню компоненту w. Проте, для вирішення проблеми, потребується зворотне перетворення, оскільки ми хочемо відновити 3D координати точок з їх 2D позицій на зображенні, щоб побудувати хмару точок. У проекті потрібно ігнорувати частину перетворення R T, оскільки можна налаштувати камеру так, щоб вона спрямовувалася безпосередньо на об'єкт в наборі даних. Однак, навіть з таким наближенням, ми все ще потребуємо значення внутрішньої матриці K для реконструкції, включаючи фокусну відстань f_{xy} і центр зображення c_{uv}. (див. рис. 2.6).

$$K = \begin{pmatrix} f_x & 0 & c_u \\ 0 & f_y & c_v \\ 0 & 0 & 1 \end{pmatrix} \tag{2.2}$$

Навіть якщо ми ігноруємо фактор R T, нам все ще не вдасться знайти зворотний зв'язок, оскільки кожному пікселю на зображенні відповідає нескінченна лінія (рис. 2.6) відповідних точок. Якщо ми позначимо x як нормалізовану 2D-координату екрану, K^-1 як псевдоінверсію матриці внутрішніх параметрів K, а C – нуль-вектор, то ми не зможемо знайти відповідну 3D-точку без додаткової інформації

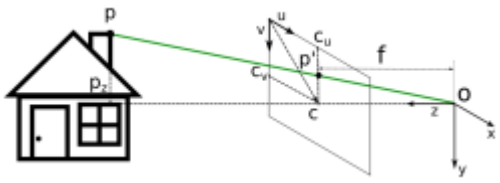


Рисунок 2.5 – Для кожного пікселя на зображенні маємо безліч точок, які відповідають йому на безкінечній прямій.

Для того, щоб вирішити задачу, потрібно переосмислити її, враховуючи три площини, які були визначені раніше. На практиці, бачимо, що всі 2D точки, які були проекцією на двовимірне зображення за допомогою внутрішньої матриці K, перетинають лазерну площину μ точно в одній

тривимірній точці (див. рис. 2.7). Тому, якщо на зображенні маємо точку, яку освітлює лазер, і ми знаємо параметри лазерної площини μ, ми можемо розв’язати систему рівнянь і визначити точне 3D положення цієї точки.

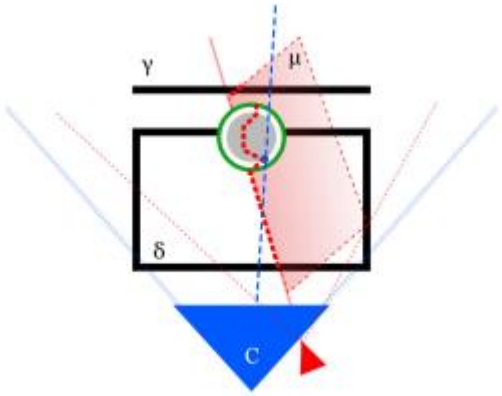


Рисунок 2.6 – Перетин між променем камери (синій) та лазерною площиною (червоний).

Оскільки маємо промінь точок для кожного пікселя, який був освітлений лазером, можемо визначити 3D-точку за допомогою перетину лінії та площини, використовуючи формулу:

$$d = \frac{(P_0 - l_0) * n}{l * n}. \quad (2.3)$$

Цей метод дозволяє отримувати зрізи поверхні об’єкта під час сканування лазером, який рухається вперед-назад, та перетворюємо їх на набір тривимірних точок. Повторюємо цей процес для кожного кадру сканування, що дозволяє нам реконструювати повну тривимірну хмару точок об’єкта.

Щоб визначити рівняння лазерної площини, нам потрібно знайти її перетин з двома опорними прямокутниками. Так як площину можна визначити за допомогою трьох точок, лазерний перетин з опорними прямокутниками, які відповідають площині, завжди створюватиме три невирівняні точки, оскільки два прямокутники перпендикулярні.

Як уже зазначалося раніше, для того, щоб відтворити 3D точки, необхідно спочатку відновити матрицю внутрішніх параметрів K камери. Попередньої інформації про цю матрицю ми не маємо, проте її можна відновити з калібрувальних зображень (див. рис. 2.2). Оскільки камера має

радіальне спотворення (див. рис. 2.7), необхідно спочатку виправити кадри перед вимірюваннями. Для цього ми використовуємо 5-параметричну поліноміальну модель спотворення [11].

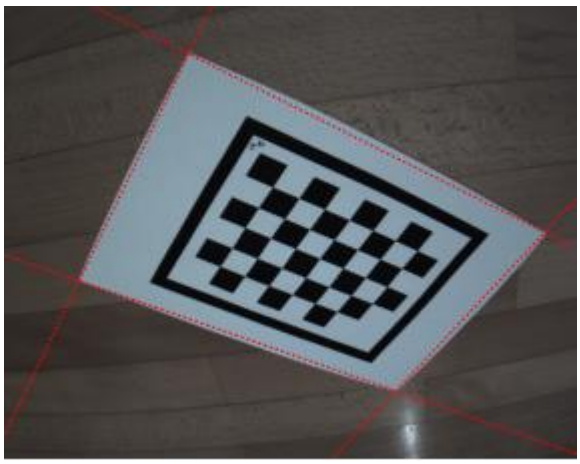


Рисунок 2.7 – Кадри набору даних представлені радіально спотворенні.

З використанням методу, описаного в " A Flexible New Technique for Camera Calibration" [16], можна вивести матрицю внутрішніх параметрів та 5 параметрів спотворення. У тестах було використано реалізацію алгоритму, надану бібліотекою OpenCV (*calibrateCamera()*).

Для отримання калібрування камери було використано всі 50 калібрувальних зображень [18, 20], наданих у наборі даних [3], і порівняно положення кутів шахової дошки з очікуваними координатами в просторі маркера-основи. Для пошуку кутів шахової дошки було успішно використано функцію *cornerSubPix()* з OpenCV.

Головною ідеєю є обрізання чорного прямокутника, використовуючи оригінальний шаблон і подібність зображення до оригіналу [21]. Було припущено, що точки кутів шахової дошки мають ті ж позиції, що й початковий шаблон, і використано ковзне вікно для оптимізації позицій точок зображення з підпіксельною точністю. Алгоритм підвищення точності підпіксельних кутів описаний в «A fast operator for detection and precise location of distinct points, corners and center of circular features» і має на меті знайти

сідлові точки сірого зображення. Після процесу калібрування було отримано наближену матрицю внутрішніх параметрів (K) та параметри спотворення d.

$$K = \begin{pmatrix} 1449.060927 & 0 & 611.9387293 \\ 0 & 1483.460784 & 392.2107774 \\ 0 & 0 & 1 \end{pmatrix} \quad (2.4)$$

$$d^T = \begin{pmatrix} -2.4568613490840346e-01 \\ -3.2570741561031419e-01 \\ 8.9675767947581340e-03 \\ 1.9515423269787597e-03 \\ 2.7000527738886521e-01 \end{pmatrix} \quad (2.5)$$

Для оцінки якості знайдених параметрів було використано середню помилку репроєкції, яка дорівнює 1,0079594964888223. З цим результатом можна проводити базову тривимірну реконструкцію.

Як було пояснено раніше, для обчислення тривимірних точок потрібно знайти двовимірні позиції точок, освітлені лазером, на зображенні, підібрати площину лазера та обчислити зворотно проектовані промені, які перетинають площину лазера. Було використано різні методи обробки зображень і порівняно їх результати, оцінюючи знайдені пікселі порівняно з очікуваними та кількість викидів. Мінімізація викидів є важливою, оскільки вони вносять шум у результат сканування та спричинюють помилки у підібранні параметрів площини лазера, точність якої є фундаментальною для отримання якісного скану.

Було знайдено найкращі результати (рис. 2.8), використовуючи простий пороговий метод визначення діапазону HSV, з наступним морфологічним відкриттям, щоб заповнити отвір з квадратним ядром 4 × 4. Цей метод ґрунтується на тому, що пікселі, які були освітлені лазером, мають специфічний постійний відтінок і насиченість, що відрізняє їх від інших пікселів на зображенні. Морфологічне відкриття дозволяє видалити деякі викиди для досягнення кращих результатів.

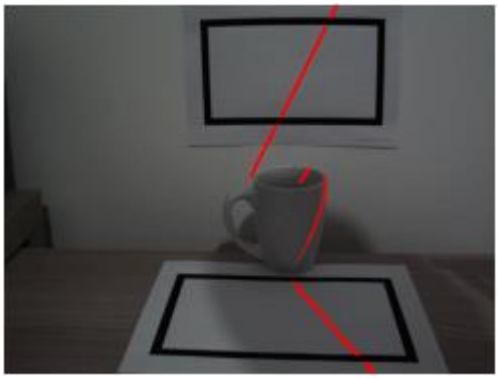


Рисунок 2.8 – Точки (червоним), знайдені лазером

результат алгоритму виявлення з використанням діапазону HSV

порогові та морфологічні оператори

Діапазони HSV, які були вибрані для цього набору даних, складаються з H: [150, 200], S: [20, 255] і V: [78, 255].

Оскільки алгоритм пропускає багато слабко освітлених точок, то було розглянуто два покращення для цього алгоритму:

Фонове віднімання: Було проведено спробу використання факту, що камера може збільшити контраст освітлених лазером точок. Для досягнення цього треба було віднімати перший кадр, в якому відсутні будь-які лазерні точки, від усіх наступних кадрів. Проте, на жаль, цей підхід погіршив виявлення, оскільки результат алгоритму виявлення використовує порогові значення в діапазоні HSV та морфологічні оператори.

Кластеризація: Для зменшення кількості викидів і розширення діапазону порогу HSV, було використано алгоритм кластеризації DBSCAN. Проте, було помічено, що цей підхід погіршив результати, оскільки багато лазерних точок не утворюють щільних скупчень.

Висновки до розділу 2

У данному дослідженні розглядається підхід до сканування 3D поверхонь, який базується на використанні структурованого світла та простого обладнання. Основною метою є відтворення тривимірної моделі об'єкта за допомогою відеозапису та аналізу його проекційних зображень.

Система складається з камери, маркерів та червоної лазерної лінійки. Маркери мають надруковану паперову площину та розміщуються перпендикулярно один до одного. Лазерна лінійка здатна перетинати обидва маркера та об'єкт. Камера фіксована та має високу роздільну здатність.

Під час відеозапису червона лазерна лінійка проходить через маркери та сканований об'єкт, створюючи проекційні зображення точок перетину. За допомогою камери ці проекційні зображення зафіксовуються. Загальна тривалість відеозаписів становить 234 секунди, що складається з 990 кадрів з частотою 15 кадрів на секунду.

Для відтворення 3D моделі об'єкта використовуються математичні моделі, такі як рівняння проекційної точки та рівняння для камери з отвором. Ці моделі дозволяють визначити тривимірну позицію кожної точки на поверхні об'єкта, використовуючи відомі 2D проекції та параметри камери.

Однією з ключових складнощів є визначення орієнтації маркерів та об'єкта в просторі. Для цього використовується калібрувальний шаблон з шаховою дошкою, який дозволяє встановити правильну орієнтацію маркерів та покращити точність реконструкції.

Загалом, запропонований підхід дозволяє отримати 3D реконструкцію поверхні об'єкта за допомогою простого та доступного обладнання. Використання структурованого світла та математичних моделей дозволяє досягти високої точності реконструкції, що відкриває широкі можливості для застосування в галузях, таких як комп'ютерне зорове сприйняття, виробництво та медицина.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Під час розробки прототипу вибір комплектуючих є важливим етапом, який може впливати на час розробки, габарити та вартість прототипу, а також на його функціональні можливості. Для забезпечення успішного вибору компонентів, було проведено дослідження ринку електронних компонентів та їх аналогів, використання яких можливе в рамках даного проекту, що дозволило вибрати оптимальні комплектуючі згідно з технічними вимогами, зазначеними в таблиці 2.7.

3.1 Плата

Arduino Uno може бути вибраний для лазерного 3D-сканера з кількох причин. По-перше, Arduino Uno – це недорога і легко доступна мікроконтролерна плата, що знижує вартість прототипу. По-друге, вона має достатньо потужний мікропроцесор для обробки даних з лазерного сканера і керування всіма компонентами пристрою. По-третє, Arduino Uno має багато доступних бібліотек і документації, що робить розробку програмного забезпечення для сканера швидкою та простою.

Рисунок 3.1 –Arduino uno виходи

Таблиця 3.1 – Характеристики Arduino uno

Мікроконтролер	ATmega328
Робоча напруга	5В
Напруга живлення (рекомендована)	7-12В
Напруга живлення (гранична)	6-20В
Цифрові входи/виходи	14 (з них 6 можуть використовуватися як ШІМ-виходи)
Аналогові входи	6
Максимальний струм одного висновку	40 мА
Максимальний вихідний струм виведення 3.3V	50 мА
Flash-пам'ять	32 КБ (ATmega328) з яких 0.5 КБ використовуються завантажувачем
SRAM	2 КБ (ATmega328)
EEPROM	1 КБ (ATmega328)
Тактова частота	16 МГц

Крім того, Arduino Uno має досить вхідних і вихідних портів, що дозволяє зручно підключати і керувати іншими компонентами сканера, такими як лазерний датчик і камера.

3.2 Драйвер крокового двигуна

L298N – це двигунний драйвер [22], який зазвичай використовується для керування кроковими двигунами або постійними двигунами в різних пристроях, включаючи 3D-принтери та лазерні 3D-сканери. Цей драйвер дозволяє керувати напругою і струмом, що подаються на двигун, що забезпечує точність та швидкість руху відповідно до вимог проекту.

Для лазерного 3D-сканера L298N використовується як драйвер для керування кроковими двигунами, які рухають лазерний сенсор по поверхні об'єкта для сканування. Використання крокових двигунів дозволяє досягти точності і повторюваності руху, що є критичним для точності 3D-сканування.

Крім того, L298N має вбудований захист від перевантаження і короткого замикання, що забезпечує безпеку роботи пристрою. Він також досить

доступний і популярний серед розробників, що дозволяє знайти необхідну документацію та ресурси для розв'язання будь-яких технічних проблем.

Отже, L298N є хорошим вибором для лазерного 3D-сканера, оскільки дозволяє керувати кроковими двигунами з точністю та швидкістю, забезпечує безпеку роботи та є доступним для розробників.

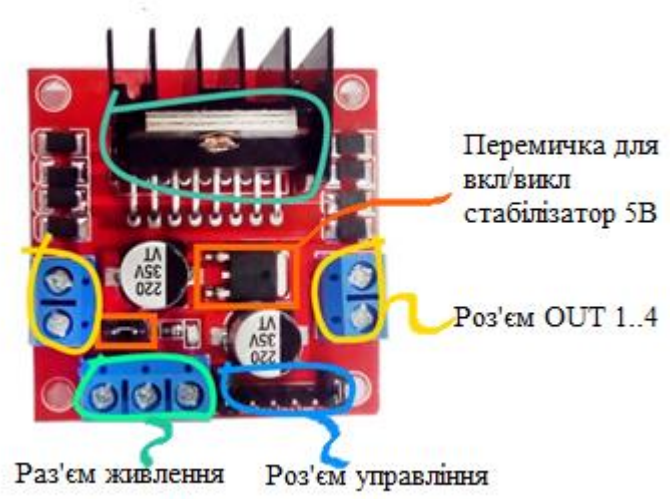


Рисунок 3.2 – L298N виходи

Таблиця 3.2 – L298N характеристики

Характеристика	Значення
Кількість каналів	2
Максимальна напруга живлення	46 В
Максимальний струм на канал	2 А
Максимальний струм при паралельному підключенні каналів	4 А
Вбудований захист від перевантаження, перегріву та короткого замикання	Так
Розміри	43 х 43 х 27 мм
Вага	33 г
Керування напругою живлення (PWM)	Так
Сумісність з різними мікроконтролерами	Ардуіно, Raspberry Pi, STM32 та інші
Максимальна частота керування	25 кГц

L298N – це двоканальний мостовий драйвер, який може керувати двома двигунами або одним кроковим двигуном. Цей модуль може забезпечити достатню потужність для використання у лазерному 3D-сканері, і він також має вбудований захист від перевантаження, перегріву та короткого замикання, що робить його безпечним використовувати. Крім того, L298N сумісний з різними мікроконтролерами, в тому числі з Ардуіно, що робить його популярним серед розробників.

3.3 Кроковий двигун

Jkongmotor NEMA17 JK42HS34-1334AC – це кроковий двигун [27] з кроковим кутом 1,8 градусів на крок, який може використовуватися для приводу механізму переміщення лазера у 3D-сканері.

Основні причини вибору даного двигуна можуть бути наступні:

– доступність та популярність: jk42hs34-1334ac є досить поширеним та доступним на ринку кроковим двигуном, що дозволяє швидко знайти запчастини та легко їх замінювати;

– потужність: jk42hs34-1334ac має високу потужність в порівнянні з іншими кроковими двигунами. це дозволяє забезпечити достатню силу для переміщення лазера та інших механізмів у 3d-сканері;

– висока точність кроку: кроковий кут 1,8 градусів на крок забезпечує достатню точність переміщення механізму, що важливо для створення точних 3d-моделей;

– сумісність з контролерами: jk42hs34-1334ac є сумісним з більшістю контролерів крокових двигунів, включаючи a4988 і drv8825, що дозволяє легко інтегрувати його в 3d-сканер.

Є два можливі варіанти підключення, які залежать від конкретної партії. В першому варіанті: червоний провід A+; зелений провід A-; жовтий провід B+; синій провід B-. У другому варіанті: зелений провід A+; чорний провід A-; синій провід B+; червоний провід B-.

У даному випадку у маємо 2 варіант підключення.

Рисунок 3.3 – Крокoвий двигун NEMA 17

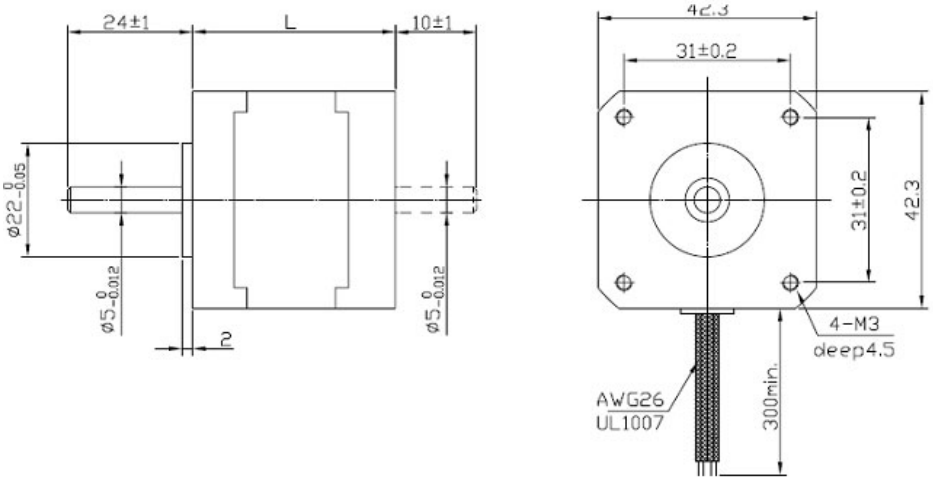


Рисунок 3.4 – Крокoвий двигун NEMA 17 габаритні розміри

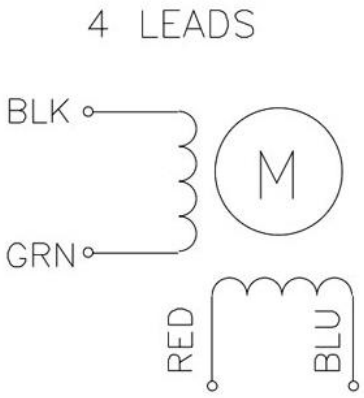


Рисунок 3.5 – NEMA 17 виходи для підключення

Крокoвий двигун Jkongmotor NEMA17 JK42HS34-1334AC має наступні характеристики(табл. 3.3)

Таблиця 3.3 – Характеристики Jkongmotor NEMA17 JK42HS34-1334AC

Характеристика	Значення
Тип	Крокoвий двигун
Модель	NEMA17 JK42HS34-1334AC
Крокoвий кут	1.8°
Кількість кроків на обертання	200
Діаметр корпусу	42 мм
Довжина корпусу	34 мм
Вихідна потужність	0.48 Вт
Напруга живлення	12 В
Ток живлення	1.33 А
Момент статичного утримання	4 кгсм
Максимальна швидкість	1000 об/хв

Основні переваги розпінування NEMA17 JK42HS34-1334AC для лазерного 3D сканера:

– висока точність кроку: цей крокoвий двигун має точність кроку 1,8 градуса, що дозволяє досягти високої точності позиціонування в просторі;

– висока крутний момент: цей крокoвий двигун має високий крутний момент 4,4 кгсм, що забезпечує достатню силу для приводу лазерної головки сканера та інших механізмів;

– надійність та довговічність: jk42hs34-1334ac має надійну конструкцію, що забезпечує стійкість та довговічність роботи;

– легка інтеграція: цей крокoвий двигун має стандартний розмір та розпинування, що дозволяє легко інтегрувати його в будь-яку систему;

– широкий діапазон застосування: jk42hs34-1334ac може бути використаний в різноманітних промислових та науково-технічних застосуваннях, включаючи лазерні 3D сканери, принтери, ЧПУ та багато іншого.

3.4 Лазерний модуль

Основним компонентом прототипу 3D-сканера є лазерний модуль S-3 [26], який використовується для вимірювання. В прототипі планується використовувати червоний лазерний модуль з проекцією променя в лінію з

кутом 90 градусів, який має основне призначення для позиціонування електроінструментів, верстатів та маркіраторів в промисловості та будівництві. Лазерні модулі також використовуються в приладобудуванні та мікроелектроніці. Технічні характеристики лазерного модуля S-3, зокрема довжина хвилі, потужність та розміри, наведені в таблиці 3.4. Позиційні модулі, з використанням лазерних компонентів, є незамінним елементом в сучасному виробництві та технологіях.

Таблиця 3.4 – Характеристики лазерного модуля s-3

Характеристика	**Значення**
Потужність	5 мВ
Довжина хвилі	640-660 нм
Напруги струму	3 В
Струм споживання	40 мА
Ресурс	10000 часов
Клас небезпеки лазера	перший

Рисунок 3.6 – Лінійний лазерний лазер

Лазери використовують для 3D сканування, оскільки вони дозволяють швидко та точно вимірювати відстань до об'єктів. Крім того, лазери мають високу точність і повторюваність вимірювань, що є необхідним для створення точної 3D-моделі об'єкта. Червоний лазерний модуль 650нм 5мВт 12*45мм з проекцією променя в лінію з кутом 90 градусів є ідеальним варіантом для 3D-сканування, оскільки він має достатню потужність для забезпечення достатньої яскравості променя та гарної видимості на поверхні об'єкта. Крім

того, такі модулі мають компактний розмір та відносно невисоку вартість, що робить їх ефективним вибором для 3D-сканування.

3.5 Камера

При аналізі аналогів на ринку було виявлено, що включення камери до складу обладнання значно підвищує загальну вартість розробки 3D-сканера. У багатьох випадках камера, що входить до комплекту, може бути обмежена функціональністю та не задовольняти потреби користувача. Застосування сучасних смартфонів, які мають високоякісні камери, є більш економічно доцільним варіантом. Крім того, багато користувачів вже мають смартфони з потрібною якістю камер у своїх кишенях, що дозволяє знизити вартість розробки та зробити 3D-сканер більш доступним для широкого кола користувачів.

В рамках проекту буде розглянуто використання звичайних камер смартфонів або веб-камер, які можуть бути додатково підключені до комп'ютера. Це дозволить знизити витрати на розробку та забезпечити більш широкий вибір засобів для захоплення зображень. На відміну від камер, що входять до комплекту, такі камери є більш універсальними та можуть забезпечити високу якість зображення. Також можливість використання різних типів камер дозволить користувачам підібрати оптимальний варіант для своїх потреб.

3.6 Підсумки вибору компонентів

Таблиця 3.5 містить детальні описи можливих компонентів, які були розглянуті при дослідженні створення 3D-сканера. Окрім того, у таблиці наведені найближчі аналоги для кожного компонента, що дозволяє оцінити їхню вартість та технічні характеристики.

Для створення лазерного модуля було розглянуто декілька варіантів, зокрема моделі від Adafruit, OSRAM, Keyence, та SainSmart. Кожна з цих

моделей має свої переваги та недоліки, які були враховані при виборі підходящої опції.

У таблиці також наведені різні варіанти кріплення для лазерного модуля, сервоприводів, та шагових двигунів, які можуть бути використані в сканері. Крім того, для керування рухом датчиків та моторів розглядалось використання мікроконтролерів, таких як Arduino Uno, Raspberry Pi, та BeagleBone Black.

У таблиці також було розглянуто декілька варіантів для електронної плати, зокрема L298N, DRV8825, та A4988. Кожен з цих контролерів має свої переваги та недоліки, які потрібно враховувати при виборі підходящої опції.

Крім того, у таблиці наведено можливі варіанти датчиків, таких як гіроскопи, акселерометри, та енкодери, які можуть допомогти підвищити точність сканування. Кожен з цих датчиків має свої технічні характеристики, які потрібно враховувати при виборі підходящої опції.

У таблиці також розглянуто можливі варіанти для блока живлення, включаючи адаптери постійного струму та літій-іонні акумулятори.

Можна використовувати звичайну веб-камеру або камеру смартфона для створення 3D-сканера. За допомогою відповідного програмного забезпечення можна обробити знімки, отримані з цих камер, та створити 3D-модель об'єкту. Крім того, знадобиться лазерний модуль, який допоможе визначити точну відстань між камерою та об'єктом. Для контролю руху лазерного модуля використовують кроковий двигун, такий як NEMA17 JK42HS34-1334AC. Щоб керувати крековим двигуном, можна використовувати драйвер крокового двигуна, такий як L298N. Окрім цього, вам знадобиться контролер Arduino Uno, який візьме на себе керування всіма компонентами 3D-сканера.

Таблиця 3.5 – Описи можливих компонентів

Компонент	Характеристики	Аналоги
Камера	Максимальна роздільна здатність: 1080p Максимальна швидкість кадрів: 30 fps Інтерфейс: USB 2.0	Logitech C920 HD Pro Webcam Microsoft LifeCam HD-3000
Лазерний модуль	Хвильова довжина: 650 нм Потужність: 5 мВт Розмір: 12x45 мм Проекція променя: лінія з кутом 90 градусів	Keyence LV-H32 Adafruit Laser Diode
Крокový двигун	Момент: 0.34 Нм Крокový кут: 1.8 градуси Напруга: 2.8 В Струм: 1.33 А	Wantai Stepper Motor NEMA 17 Bipolar Stepper Motor
Драйвер крокового двигуна	Вхідна напруга: 5-35 В Максимальний струм: 2А	A4988 Stepper Motor Driver TB6600 Stepper Motor Driver

3.7 Налаштування Arduino uno

Arduino Uno – це мікроконтролер, що використовується для розробки електронних пристроїв та систем з відкритим кодом. Його можна програмувати з використанням мов програмування, таких як C і C++, а також з використанням Arduino IDE – інтегрованого середовища розробки.

Для налаштування Arduino Uno необхідно встановити драйвери для зв'язку з комп'ютером. Це можна зробити, завантаживши драйвери з офіційного сайту Arduino. Після встановлення драйверів необхідно вибрати правильний порт, до якого підключена плата.

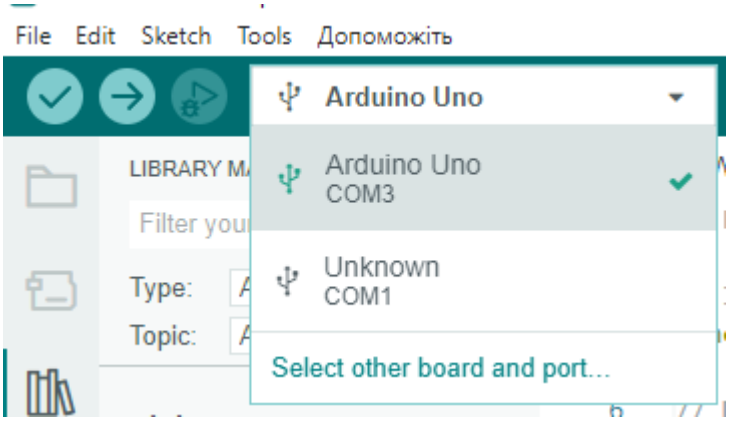


Рисунок 3.7 – Вибір порту підключення

Після цього необхідно завантажити прошивку на плату Arduino Uno. Для цього можна скористатися Arduino IDE, створивши новий проект та скомпілювавши його код. Після цього код можна завантажити на плату, підключивши її до комп'ютера за допомогою USB-кабелю.

Також для налаштування платформи можна використовувати додаткові бібліотеки та розширення, що дозволяють розширювати можливості платформи та забезпечувати зв'язок з іншими електронними пристроями та сенсорами.

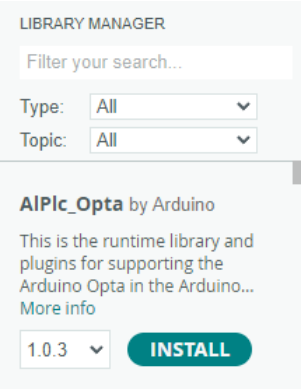


Рисунок 3.8 – Встановлення бібліотек в автоматичному режимі

Після налаштування платформи Arduino Uno можна розробляти програми, що взаємодіють з іншими компонентами, такими як лазерний модуль, камера, кроковий двигун та інші, що використовуються в 3D-сканері.

Кроковий двигун повинен бути прикріплений до лазера, щоб можна було його контролювати. Драйвер крокового двигуна служить для керування кроковим двигуном, дозволяючи точно контролювати рух лазера в заданому напрямку та куті.

Для усіх цих випадків потрібно використовувати лазер, кроковий двигун та драйвер крокового двигуна

Для налаштування Arduino uno для руху крокового двигуна використовуються спеціальні бібліотеки. Перш за все, необхідно встановити бібліотеку AccelStepper, яка дозволяє керувати кроковим двигуном. Далі, використовуючи функції бібліотеки, можна налаштувати режим роботи крокового двигуна, напрямок руху, швидкість та інші параметри.

Після налаштування крокового двигуна та драйвера, необхідно підключити камеру до Arduino uno. Для цього можна використовувати USB-порт, який забезпечує зв'язок між камерою та Arduino. Після підключення камери можна використовувати функції бібліотеки для керування камерою та отримання зображення.

Отримане зображення можна обробляти за допомогою програмного забезпечення, яке дозволяє відслідковувати рух лазера та створювати 3D-модель об'єкта. Таким чином, налаштування Arduino uno є важливим етапом у створенні 3D-сканера, оскільки вона дозволяє керувати рухом крокового двигуна та камерою, що є необхідним для сканування об'єктів.

3.8 Програмування крокового двигуна

На сам перед потрібно встановити бібліотеку AccelStepper, після її встановлення можна розпочинати роботу з двигуном.

AccelStepper – це бібліотека для Arduino, яка дозволяє просто та точно керувати кроковими двигунами. Дана бібліотека містить набір функцій, що дозволяють налаштовувати різні параметри руху крокового двигуна. Основні функції, які використовуються для керування кроковими двигунами виконують наступні операції:

– AccelStepper(steps, pin1, pin2) – конструктор класу, де steps – кількість кроків на обертання, pin1 та pin2 – піни для керування двигуном;

– setSpeed(speed) – встановлює швидкість руху крокового двигуна в кроках за секунду;

– setMaxSpeed(maxSpeed) – встановлює максимальну швидкість руху крокового двигуна в кроках за секунду;

– setAcceleration(acceleration) – встановлює прискорення руху крокового двигуна в кроках за секунду на другу;

– move(steps) – рухає кроковий двигун на задану кількість кроків steps;

– run() – виконує один крок крокового двигуна у напрямку, визначеному методом setDirection();

– runSpeed() – виконує кроки крокового двигуна з швидкістю, встановленою методом setSpeed();

– runToPosition() – рухає кроковий двигун до встановленої позиції.

Ці функції дозволяють здійснювати різноманітні операції з кроковими двигунами та налаштовувати їх параметри руху, швидкості та прискорення.

Для відскановання об’єктів потрібно повертати лазер за допомоги крокового двигуна на 180 градусів що б охопити усю поверхню

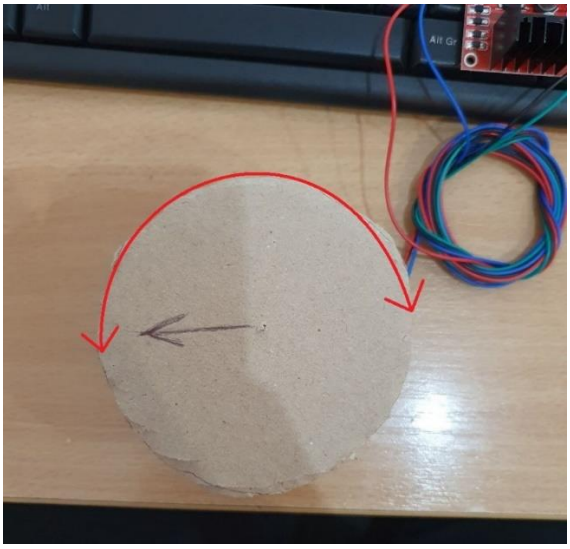


Рисунок 3.9 – Оборот лазеру

Для сканування об'єктів з використанням лазера та крокового двигуна було вирішено встановити невелику платформу на вал крокового двигуна. Ця платформа повільно обертається, а на ній знаходиться лазер, який буде сканувати об'єкти. Лазер може бути вмонтований в спеціальний кріплення, яке дозволяє регулювати його положення та орієнтацію.

Для того, щоб лазер сканував об'єкти, потрібно його рухати по определеній траєкторії, яку можна контролювати за допомогою крокового двигуна. Кроковий двигун можна відправляти команди на рух у певному напрямку та на певну відстань. Також можна налаштувати швидкість руху крокового двигуна.

Для контролю руху лазера за допомогою крокового двигуна можна використовувати певні бібліотеки в середовищі Arduino, наприклад, AccelStepper. З її допомогою можна встановити кількість кроків, які має виконати кроковий двигун для повного оберту платформи, і контролювати швидкість руху. Також можна встановити напрямок руху та налаштувати прискорення та сповільнення для більш плавного руху.

У результаті такої конфігурації, лазер буде повільно сканувати об'єкти, які розміщені на платформі, а кроковий двигун буде керувати її рухом. Результатом сканування буде набір координат точок, які можна використовувати для створення 3D-моделі об'єкта.

3.9 Підключення драйверу руху L298N

L298N – це інтегральна мікросхема, яка зазвичай використовується для управління двигунами в робототехніці та інших системах з електричними двигунами. За допомогою L298N можна керувати двома постійними або кроковими двигунами з максимальним струмом до 2 А на кожен з них. Мікросхема має декілька входів керування, які дозволяють керувати напрямом руху двигуна, швидкістю руху, а також можливість регулювати прискорення та сповільнення двигуна.

Для керування напрямом руху двигуна L298N має два входи: IN1 та IN2. При високому стані обох входів двигун рухається в одному напрямку, а при переведенні одного з входів в низький стан, а інший залишається високим, рух відбувається в зворотньому напрямку.

L298N має вихід для підключення внутрішнього резистора для обмеження струму, що проходить через кожен канал, що дозволяє контролювати максимальний струм, що подається на двигун.

Для підключення L298N до Arduino необхідно з'єднати вихідні піни L298N з відповідними пінами Arduino та підключити кроковий двигун до виходу L298N. Для керування напрямком руху двигуна та його швидкості можна використовувати відповідні піни Arduino, а для обмеження струму – відповідний резистор.

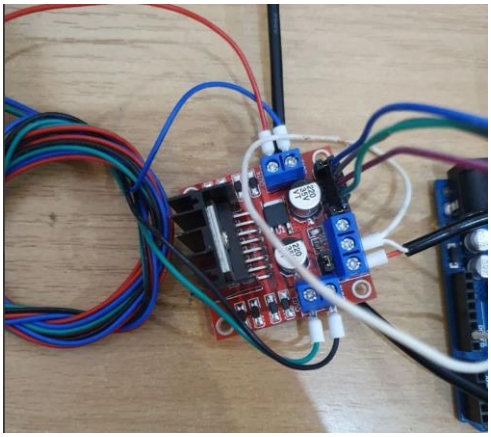


Рисунок 3.10 – Драйверу руху L298N

Для кращого з’єднання було використано обжимну гільзу(рис 3.11)

Рисунок 3.11 – Обжимна гільза

Обжимні гільзи є важливими компонентами для створення надійних з'єднань між проводами або кабелями. Зазвичай вони виготовляються з міді або іншого металу, і призначені для того, щоб затиснути провід між двома частинами гільзи. Для використання обжимних гільз необхідно мати спеціальний інструмент, який дозволить правильно затиснути гільзу. Інструмент зазвичай складається з матриці та кримпера. Матриця використовується для формування гільзи, а кримпер – для її затискування.

Правильний розмір гільзи є важливим для надійного з'єднання. Неправильно вибраний розмір може призвести до поганого затискування проводів, що може викликати проблеми з електричним з'єднанням. Якщо правильно вибрати розмір гільзи та затиснути її, то з'єднання буде надійним і стійким до розривів та інших впливів зовнішнього середовища.

Для драйверу руху L298N не потрібно писати додатковий програмний код, а просто потрібно проініціалізувати піни підключення в коді:

```
// Create instance of Stepper library
Stepper myStepper(stepsPerRevolution, 8, 9, 10, 11);
```

Модуль L298N має перемикач «5V enable», який дозволяє вибрати джерело живлення для чіпа. Якщо напруга на вході до модуля менше 12В, то перемичку потрібно залишити на місці, і він буде отримувати живлення через стабілізатор на модулі. Однак, якщо вхідна напруга вище 12В, то перемичку потрібно зняти, а живлення 5В для чіпа подавати на контакт «+ 5V power».

Для живлення моторів можна використовувати напругу від 5 до 35В, яку потрібно подавати на контакт «12V power». Не слід підключати живлення до контакту «+ 5V power», якщо перемичка «5V enable» встановлена на місці. Якщо перемичку знято, то напругу 5В для чіпа можна взяти з контакту «+ 5V power». Важливо пам'ятати, що якщо використовуються мотори з напругою вище 12В, перемичку «5V enable» необхідно зняти, щоб уникнути перегріву стабілізатора на модулі.

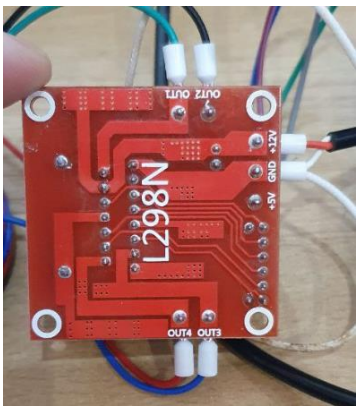


Рисунок 3.12 – Підключення L298N

До пінів Out1, Out2 потрібно під’єднати піни A+, A-. До пінів Out3, Out4 під’єднуються B+, B-.

ENA та ENB – ці контакти використовуються для регулювання швидкості обертання двигунів. При подачі логічної «1» на ENA, імпульсна ширина модулюється відповідно до вхідного сигналу PWM, що дозволяє змінювати швидкість обертання двигуна. Аналогічно, при подачі логічної «1» на ENB, можна контролювати швидкість другого двигуна.

VCC – це контакт живлення, до якого потрібно підключити джерело живлення з напругою від 5 до 35 В.

GND – цей контакт призначений для заземлення модуля та підключення до нього інших елементів схеми.

5V power – цей контакт призначений для подачі стабілізованої напруги 5 В на чіп L298N, якщо перемичка «5V enable» знята.

12V power – цей контакт призначений для подачі напруги від 5 до 35 В на двигуни.

5V enable – ця перемичка використовується для вибору джерела живлення чіпа L298N. При використанні моторів з напругою вище 12В необхідно зняти перемичку «5V enable» і подати на вхід «+ 5V power» живлення 5В для чіпа L298N. Якщо ж живлення моторі.

Рисунок 3.13 – Зовнішній блок живлення 12 В

Зовнішній блок живлення 12 В під’єднаний до 12 В роз’єму та виведено загальну землю на Arduino.

3.10 Підключення лазеру

Лінійний лазер – це електронний прилад, який випромінює тонку лазерну лінію, що використовується для сканування поверхонь в промислових та наукових застосуваннях, включаючи 3D сканування.

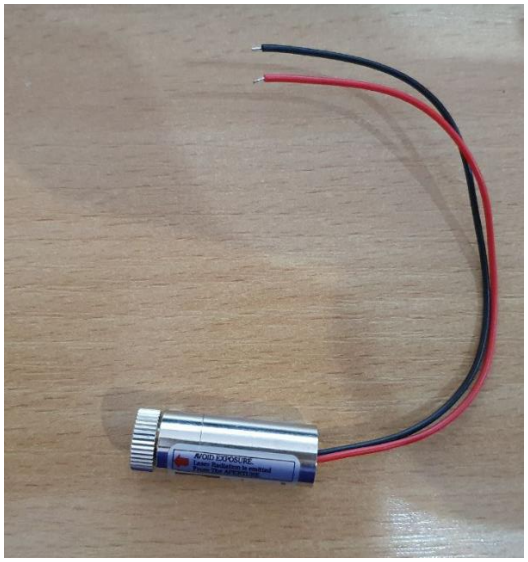


Рисунок 3.14 – Лінійний лазер

Для розробляємого 3D-сканера потрібно під’єднати лазер до ардуїно до цифрового виходу та мінус, додати до загального мінусу.

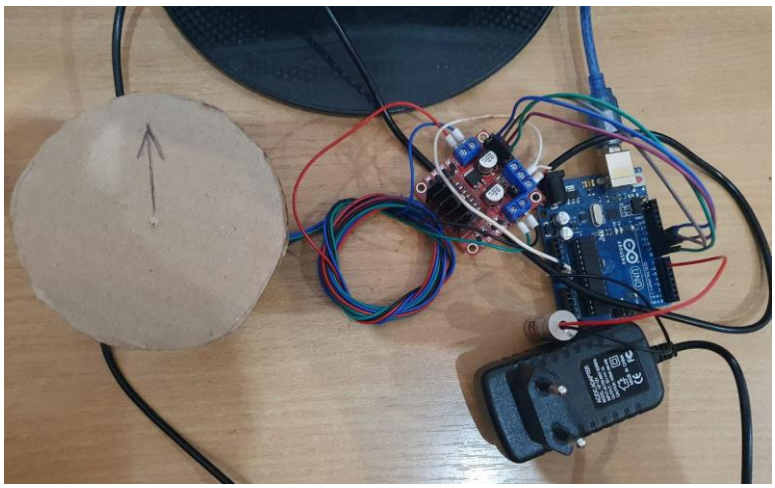


Рисунок 3.15 – Зібрана конструкція

Лазер ставиться на платформу, яку обертає кроковий двигун. Камера ставиться у будь-якому місці для калібрування.

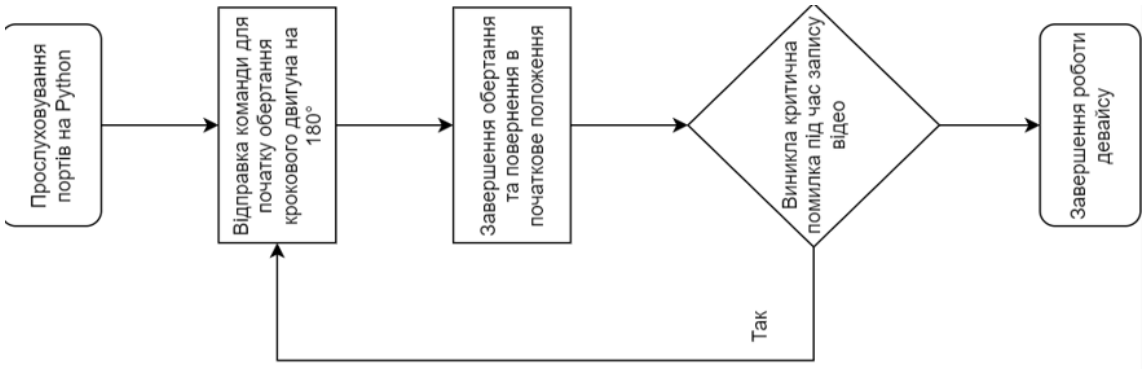


Рисунок 3.16 – Алгоритм роботи девайсу

Девайсу потрібно отримати команду про початок обертання крокового двигуна, повернути платформу на 180°, після завершення обертання повернутися в початкову точку, отримати сигнал що запис відео пройшов успішно та завершити роботу. У разі якогось збою в записі відео повторити алгоритм.

3.11 Розробка ПЗ

Під час розробки програмного забезпечення було прийнято рішення використовувати мову програмування Python. Це досить популярна мова в галузі розробки, оскільки вона має простий і зрозумілий синтаксис, який

дозволяє швидко розробляти та тестувати код. Python також має велику кількість сторонніх бібліотек, які допомагають значно зменшити час розробки та поліпшити функціональність програми. Крім того, Python є мовою з відкритим вихідним кодом, що дає можливість розробникам з усього світу співпрацювати над вдосконаленням мови та додавати нові функціональність. Загалом, використання Python є хорошим вибором для багатьох типів проектів, особливо тих, що пов'язані з обробкою даних, машинним навчанням та штучним інтелектом.

Бібліотеки, які використовуються у даному випадку, забезпечують можливість обробки зображень та роботу з файловою системою в мові програмування Python. Бібліотека *glob* дозволяє знайти всі шляхи, які відповідають заданому шаблону, тобто списки файлів у певному каталозі, які потрібно обробити. Бібліотека *os* надає функції для взаємодії з операційною системою, наприклад, для роботи з файловою системою. Бібліотека *sys* використовується для отримання аргументів командного рядка. Бібліотека *cv2* (*OpenCV*) містить функції для роботи з зображеннями та відео, що використовуються в області розпізнавання образів, машинного навчання та комп'ютерної графіки. Бібліотека *numpy* дозволяє ефективно працювати з великими об'ємами даних, зокрема з обробкою зображень у вигляді масивів.

Для зручного взаємодії з користувачем, було створено простий інтерфейс, який складається з чотирьох кнопок. Перша кнопка дозволяє зробити 50 фотографій для калібрування камери. Друга кнопка дозволяє провести калібрування камери для отримання точних даних про її параметри та відкоригувати можливі відхилення, що можуть виникати під час зйомки зображень. Третья кнопка дозволяє розпочати запис відео, що буде використано для подальшої обробки. Четверта кнопка запускає процес обробки записаного відео, що включає в себе виконання різноманітних операцій зображення для отримання більш точних та якісних результатів. За допомогою цих кнопок користувач може легко керувати процесом

калібрування камери, запису відео та його обробки без необхідності писати складні команди в терміналі.

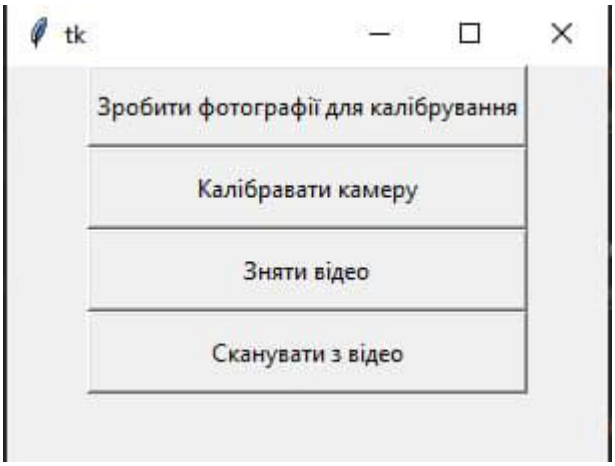


Рисунок 3.17 – Інтерфейс застосунку

Для початку нам потрібно зробити 50 зображень за допомогою веб-камери. Програма починається з захоплення відеопотоку за допомогою об'єкта *VideoCapture*. Потім встановлюються ширина і висота кадру. Далі створюється папка для збереження зображень.

Далі створюється функція *on_mouse_click*, яка буде викликатися при кліку лівою кнопкою миші на кнопці «Take photo», яка з'явиться на екрані. Усередині цієї функції зображення, яке відображається в даний момент, зберігається в файл з унікальним ім'ям. Потім збільшується значення лічильника, і якщо значення досягає 50, то вікно програми закривається, інакше текст на кнопці оновлюється, і на екрані все ще відображається поточне зображення.

Далі створюється вікно з іменем «*frame*», на якому будуть відображатися зображення. Встановлюється обробник подій миші для вікна. У циклі while зображення захоплюється, на ньому відображається значення лічильника, а також кнопка «Take photo». Потім зображення відображається на екрані. Якщо користувач натискає клавішу "q", то програма закривається.

В кінці звільняються ресурси за допомогою методу *release()*, і всі вікна закриваються за допомогою методу *destroyAllWindows()*.

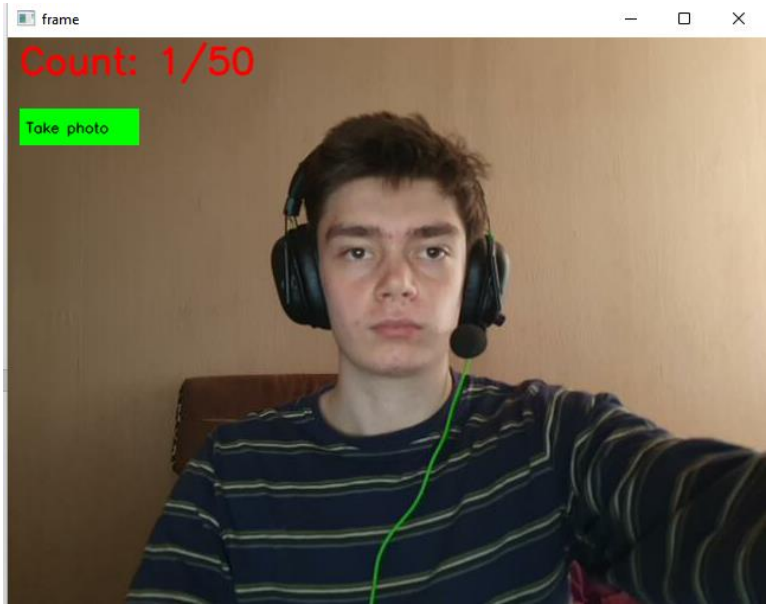


Рисунок 3.18 – Приклад роботи камери

Після того, як було зроблено 50 фотографій з патернами для калібрування, можна натискати кнопку на другу кнопку та починати процес налагодження камери. Під час налагодження камери буде виконуватися різні дії, такі як розрахунок параметрів камери, визначення положення об'єктів на зображенні та їхніх розмірів, а також визначення відстані до об'єктів. Для цього використовуються різні алгоритми та методи, такі як калібрування камери за допомогою *OpenCV*.

Далі потрібно коректно відкалібрувати зображення. Для цього використовується оптичне розпізнавання зображень (OCR) та геометричні трансформації.

У функції *checkBlankArea(warped)* перевіряється середнє значення яскравості підмножини зображення, яке відповідає пустим клітинам шахівниці. Ця функція використовується для вирівнювання шахівниці, так як на пустій клітинці очікується низький рівень яскравості.

У функції *removeXYSigns(warped)* білокутий прямокутник використовується для маскування позначень координат XY на зображенні, оскільки вони вводять шум у алгоритм вирівнювання.

Функція *findRectanglePatterns(gray)* знаходить всі можливі прямокутні шаблони на сірому зображенні, відсортовані за рейтингом. Для знаходження шаблонів використовується алгоритм порогової обробки Оцу (Otsu), а потім знаходяться всі можливі контури. Знайдені контури апроксимуються многокутниками, а потім фільтруються за довжиною та кутовою формою, щоб відбирати прямокутники.

Функція *findRectanglePatternHomography(gray)* знаходить прямокутний шаблон та оцінює матрицю гомографії. Першим кроком використовується функція *findRectanglePatterns(gray)* для знаходження найкращого прямокутного шаблону. Знайдений прямокутник використовується для обчислення матриці гомографії, яка застосовується для зміщення та зміни масштабу зображення. Після цього може відбуватися поворот зображення, тому що шахівниця може бути розташована в неправильному положені.

Функція *rotateAndCropImage(warped, pts)* обертає та обрізає зображення так, щоб шахівниця була вирівняна та знаходилася в центрі зображення. Для цього використовується матриця гомографії, яку було отримано у функції *findRectanglePatternHomography(gray).* Зображення обрізається до розміру шахівниці, який визначається на основі знайдених контурів клітинок.

Після вирівнювання та обрізки зображення, функція *recognizeChessBoard(warped)* використовує оптичне розпізнавання зображень (OCR) для визначення позиції фігур на шахівниці. Для цього зображення розбивається на клітинки, які проходять через центр кожної клітинки шахівниці.

Для кожної клітинки виконується OCR, щоб визначити, яка фігура зображена в цій клітинці. Результат розпізнавання фігур записується у вигляді двовимірного масиву, який представляє позиції фігур на шахівниці.

Функція *run()* зчитує набір зображень з вказаної директорії і для кожного зображення:

– знаходить прямокутний шаблон на ньому, який служить як калібрувальна основа;

– знаходить внутрішні кути шахівницевого шаблону на ньому, які служать точками калібрування;

– вдосконалює положення знайдених кутів;

– зберігає знайдені кути шахівниці в списку точок зображення та списку точок об'єкта.

```
Image 8/50

Image 1/50
Image 2/50
Image 3/50
Image 4/50
Image 5/50
Image 6/50
Image 7/50
Image 8/50
Image 9/50
Image 10/50

RMS: 9.686979135608168

Distortion parameters:
K: [[1.94255744e+04 0.08880000e+00 1.81905686e+82] [0.00000000e+00 3.78207561e+04 3.92078845e+82] [0.00000000e+00 0.00000000e+00 1.00000000e+00]]

[[-6.26654068e+01 4.59487921e+04 -2.53255039e-01 -2.15027532e-01 -5.15939852e+03]]
Images used for calibration: 11/58
Saved intrinsics in intrinsics.xml
```

Рисунок 3.19 – Тестова калібровка камери

Після обробки всіх зображень вона виконує калібрування камери, викликаючи *cv2.calibrateCamera()*, який повертає внутрішні параметри камери, такі як матриця камери K, параметри спотворення *dist*, вектори повороту та перекладу *rvecs* та *tvecs*, а також середньоквадратичну похибку калібрування.

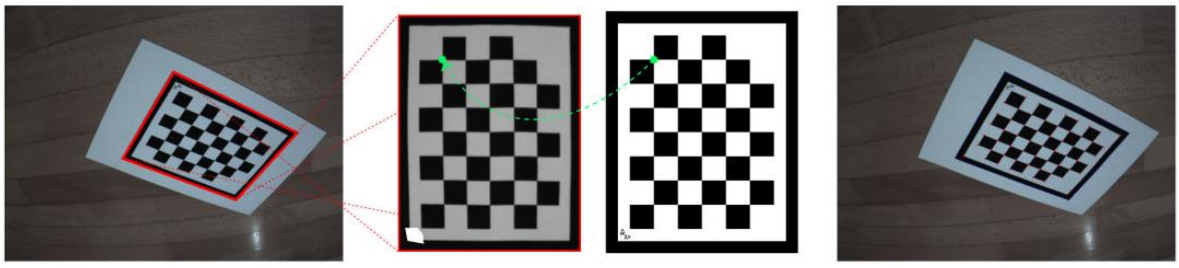


Рисунок 3.20 – Принцип калібрування

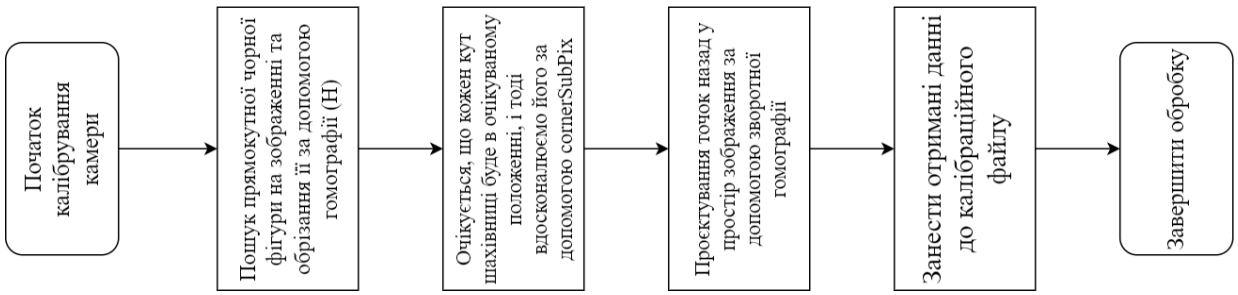


Рисунок 3.21 – Алгоритм калібрування камери з кожного зображення

Далі буде записано відео завдяки девайсу створеному раніше та мобільного телефону, який приєднаний до ПК

Команди на python прослуховують порт підключення до ардуіно та при натисканні кнопки с початком запису відео відправляють команду про запуск лазеру та початок обертання крокового двигуна.

Після запису відео можна приступити до його обробки.

То ж після запуску процесу сканування у відео запускається python скрипт який обробляє кадри з камери, виявляє лазерну лінію на зображенні, знаходить прямокутні малюнки на зображенні і за допомогою цих малюнків оцінює позицію лазерної лінії в тривимірному просторі. Потім підбирає площину до оцінених тривимірних точок і обчислює позицію всіх точок на лазерній лінії в тривимірному просторі.

На початку, зчитуються камерні параметри (матриця калібрування K та коефіцієнти спотворення *dist*) з файлів, створених під час калібрування камери.

Далі, зчитується перший кадр відео, який використовується для знаходження контурів прямокутників, які використовуються для визначення площини, на яку проектується 3D-хмара.

Потім, знаходяться матриці гомографії для площини стіни та столу, а також їх площини, які використовуються для проектування 3D-хмари на відповідні поверхні.

Далі, зчитується кожен кадр відео та обробляється. Кожен кадр спочатку пройде крізь алгоритм видалення фону, після чого проектується на 3D-хмару.

Знайдені точки та відповідні кольори зберігаються у відповідних масивах. Після того, як всі кадри оброблено, створюється 3D-хмара на основі знайдених точок та їх кольорів та зберігається у файл «output.py».

Останнім кроком є візуалізація 3D-хмари віджерелом відкритого програмного забезпечення Open3D.

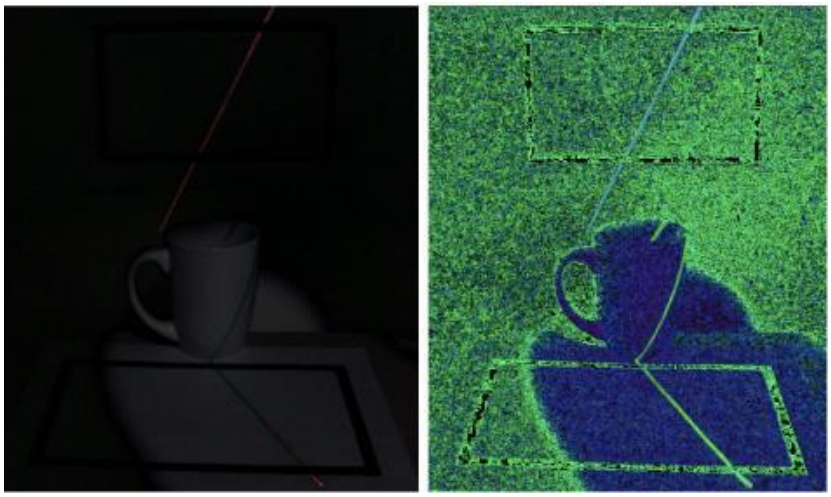


Рисунок 3.22 – Фоновий віднятий кадр та його відповідна конвертація в HSV (відтінок, насиченість, значення)

Функція *findRectanglePatterns* використовується для пошуку прямокутних малюнків на першому кадрі відео. Функція findReference3DPoints використовується для пошуку всіх тривимірних точок всередині прямокутника, що перетинає площину. Функція *processFrame* використовується для обробки окремого кадру відео, виявлення лазерної лінії, пошуку опорних точок на столі та стіні, підбирання площини до точок і обчислення позиції всіх точок на лазерній лінії.

Потік обробки складається з наступних кроків [19]:

– перетворення кадру на градації сірого та порогова обробка за допомогою методу отсу, щоб отримати двійкове зображення;

– пошук всіх можливих контурів на двійковому зображенні;

– наближення контурів до багатокутників та відсіювання всіх багатокутників, які не мають чотирьох сторін та не є опуклими;

– сортування залишених багатокутників за їх баллами (обчисленими за допомогою функції outercontour) у порядку спадання;

– використання першого багатокутника у відсортованому списку як опорного прямокутника для стіни та другого багатокутника як опорного прямокутника для столу.

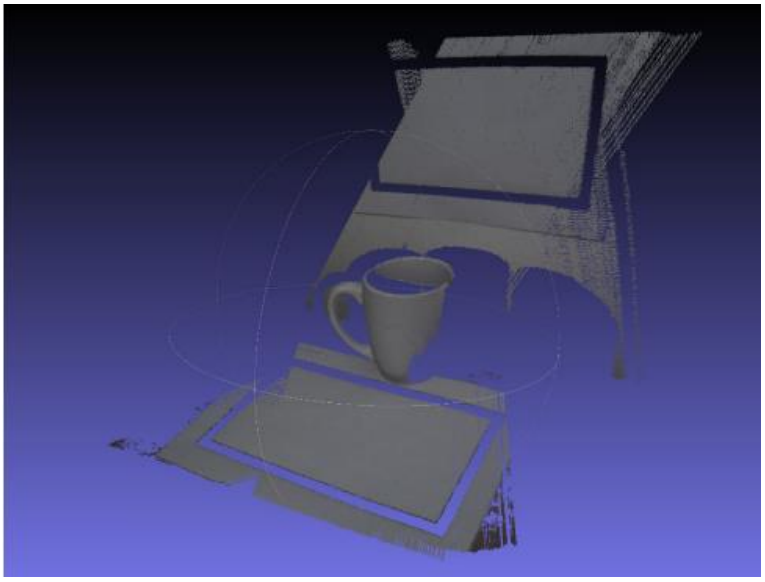


Рисунок 3.23 – Отриманий point cloud

Також було підраховано кількість часу яку займає сканування. Повний алгоритм роботи застосунку можна знайти в додатку Б, рис Б1-Б2.

Таблиця 3.6 – Швидкість сканування

Ім'я файлу	Тривалість відео, с	Загальний час обробки, с	Час обробки кадрів, с
cup1.mp4	62	101	79
toy.mp4	55	96	61
sneakers.mp4	74	121	105

Із таблиці 3.6 можна побачити, що тривалість сканування не перевищує 5 хвилин не залежно від розмірів об’єкта що є бажаним результатом.

Висновки до розділу 3

У даному розділі дипломної роботи було проведено детальний аналіз лазерного методу 3D-сканування. Для цього було створено простий 3D-сканер

з деталей загальною вартістю не більше 1000 тисяч гривень. Також було розроблено застосунок на мові Python, який спрощує взаємодію зі сканером для користувача. Для перевірки працездатності сканера було проведено ряд тестів, використовуючи розроблений застосунок та девайс.

Застосунок дозволяє легко взаємодіяти зі сканером та налаштовувати параметри для під’єднаної до нього камери. Також застосунок має можливість збереження результатів сканування у форматі PLY, що дозволяє легко імпортувати дані у програми для обробки 3D-моделей.

Після проведення тестів можна стверджувати, що розроблений сканер та застосунок працюють досить швидко та точно. Було проведено сканування об'єктів різної форми та розміру, і результати були задовільними. Також важливо відмітити, що тривалість сканування не перевищує 5 хвилин, що є бажаним результатом для даного типу сканерів.

Отже, можна зробити висновок, що розроблений лазерний 3D-сканер та відповідний застосунок є ефективними та доступними засобами для створення 3D-моделей. Це може бути корисно для різних галузей, таких як архітектура, медицина, промисловість тощо.

ВИСНОВКИ

В даній дипломній роботі було проведено дослідження лазерного методу 3D-сканування та розроблено простий 3D-сканер із деталей загальна сума яких не перевищує 1000 тисячі гривень.

У роботі було проаналізовано різні методи 3D-сканування та обрано лазерний метод який дозволяє отримати точні та деталізовані моделі об'єктів. Також були розглянуті основні компоненти лазерного сканера та їхні функції.

Для реалізації лазерного сканера було використано компоненти які можна придбати у магазинах електроніки. Для обробки даних сканування було розроблено застосунок на мові програмування Python. Проведені тести сканера дозволили отримати точні та деталізовані 3D-моделі об'єктів. Також було визначено, що тривалість сканування не перевищує 5 хвилин що є бажаним результатом.

У цілому, розроблений лазерний 3D-сканер є простим та доступним для будь-якого користувача. Він дозволяє отримати точні та деталізовані 3D-моделі об'єктів за короткий час. Даний проект можна використовувати у різних галузях, наприклад, у промисловості, медицині, культурі та інших.

В майбутньому можна розробити більш складний та продуктивний лазерний 3D-сканер з використанням новітніх технологій та матеріалів. Також можна розширити функціональні можливості застосунку, наприклад, додати можливість збереження 3D-моделей у різних форматах, аналізувати отримані моделі для виявлення дефектів або аналізу їхніх характеристик.

Дана робота може бути використана як основа для подальшого дослідження в галузі 3D-сканування та розробки нових методів та технологій. Вона може бути корисною для науковців, інженерів, дизайнерів, медиків та інших спеціалістів, які займаються розробкою та виготовленням 3D-моделей.

Для покращення якості сканування можна застосувати різноманітні техніки, такі як автофокус, зменшення впливу шуму на дані сканування та оптимізація алгоритмів обробки даних. Також можна розглянути можливість

додаткової обробки даних сканування за допомогою програмного забезпечення для моделювання та аналізу 3D-моделей.

У цілому, дана дипломна робота дозволяє вирішити проблему доступності 3D-сканування для широкого кола користувачів. Розроблений лазерний 3D-сканер та застосунок на Python дозволяють швидко та ефективно створювати точні та деталізовані 3D-моделі об'єктів. Дані технології можуть бути використані у різних галузях, де потрібна висока точність та деталізація при створенні 3D-моделей. Загалом, дана дипломна робота відображає успішний проект з розробки лазерного 3D-сканера та відповідного застосунку на мові програмування Python. Результати дослідження та тестування свідчать про високу точність та ефективність розробленого 3D-сканера, що робить його досить привабливим для широкого кола користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Бевза І. О. 3D-сканер : bachelor's thesis. 2019. 63 с. URL: https://ela.kpi.ua/handle/123456789/28937 (дата звернення: 06.01.2023).

2. Покидько Л. М. Алгоритми побудови 3D-моделі об’єкту лазерного сканування. Збірник наукових праць Військового інституту Київського національного університету імені Тараса Шевченка. 2012. Вип. № 38. С. 86–92 (дата звернення: 20.02.2023).

3. Ram A., Jalal S., Jalal A., Kumar M. A density based algorithm for discovering density varied clusters in large spatial databases. *International journal of computer applications*. 2010. Vol. 3, No. 6. P. 1–4. URL: https://doi.org/10.5120/739-1038 (Last accessed: 17.02.2023).

4. Rocchini C. *A low cost 3D scanner based on structured light / Computer graphics forum*. 2001. Vol. 20, No 3. P. 299–308. URL: https://doi.org/10.1111/1467-8659.00522 (Last accessed: 05.05.2023).

5. Bleier M., Nüchter A. *Low-cost 3d laser scanning in air orwater using self-calibrating structured light. ISPRS – international archives of the photogrammetry, remote sensing and spatial information sciences. 2017*. XLII-2/W3. С. 105–112. URL: https://doi.org/10.5194/isprs-archives-xlii-2-w3-105-2017 (Last accessed: 07.05.2023).

6. Bouguet J. Y., Perona P. 3D *photography on your desk. IEEE 6th international conference on computer vision*, м. Bombay, India. URL: https://doi.org/10.1109/iccv.1998.710699 (Last accessed: 15.03.2023).

7. Bullinger S. *Image-Based 3D reconstruction of dynamic objects using instance-aware multibody structure from motion / sebastian bullinger*. 2020. URL: http://d-nb.info/1216243115/34 (Last accessed: 24.04.2023).

8. Daneshmand M., Traumann A., Anbarjafari G. *3D size-estimation based on the geodesic distance measured by photogrammetric scanning device. 6th international conference on 3D body scanning technologies, lugano, switzerland*, 27-28 october 2015, м. Lugano, Switzerland, 27–28 жовт. 2015 р. Ascona,

Switzerland, 2015. URL: https://doi.org/10.15221/15.221 (Last accessed: 23.02.2023).

9. Förstner W., Gülch E. *Automatic orientation and recognition in highly structured scenes. ISPRS journal of photogrammetry and remote sensing*. 1999. Vol. 54, No 1. P. 23–34. URL: https://doi.org/10.1016/s0924-2716(98)00022-7 (Last accessed: 15.03.2023).

10. Hirofumi Nakai, Daisuke Iwai, Kosuke Sato. *3D shape measurement using fixed camera and handheld laser scanner. SICE 2008 – 47th annual conference of the society of instrument and control engineers of japan,* м. Chofu, 20–22 серп. 2008 р. 2008. URL: https://doi.org/10.1109/sice.2008.4654904 (Last accessed: 01.05.2023).

11. J. Brünger *3D imaging, analysis and applications*. Cham, 2020. P. 1–36. URL: https://doi.org/10.1007/978-3-030-44070-1_1 (Last accessed: 01.05.2023).

12. *Least squares data fitting. Introduction to applied linear algebra*. 2018. P. 245–284. URL: https://doi.org/10.1017/9781108583664.014 (Last accessed: 04.05.2023).

13. *New compact 3D scanner. Metal powder report*. 2021. Vol. 76, No 4. P. 183. URL: https://doi.org/10.1016/j.mprp.2021.06.077 (Last accessed: 18.05.2023).

14. Rusinkiewicz S., Hall-Holt O., Levoy M. *Real-time 3D model acquisition. ACM transactions on graphics*. 2002. Vol. 21, No 3. P. 438–446. URL: https://doi.org/10.1145/566654.566600 (Last accessed: 13.04.2023).

15. *Scaning electron microscopes. Metal finishing*. 2003. Vol. 101, no. 12. P. 63. URL: https://doi.org/10.1016/s0026-0576(03)90123-1 (Last accessed: 14.01.2023).

16. Wang X. *Scanning probe contact printing* Langmuir. 2003. Vol. 19, No 21. P. 8951–8955. URL: https://doi.org/10.1021/la034858o (Last accessed: 18.03.2023).

17. Sousa J. F. C. *Scanner a 3D : master's thesis*. 2015. URL: http://hdl.handle.net/1822/52459 (Last accessed: 09.02.2023).

18. Dellaert F. *Structure from motion without correspondence*. *IEEE conference on computer vision and pattern recognition. CVPR 2000*, м. Hilton Head Island, SC, USA. URL: https://doi.org/10.1109/cvpr.2000.854916 (Last accessed: 11.05.2023).

19. Xiao J., Leng X. X., Li D. Y. *3D reconstruction of rock mass based on point cloud library. Applied mechanics and materials*. 2014. Vol. 543-547. P. 2920–2923. URL: https://doi.org/10.4028/www.scientific.net/amm.543-547.2920 (Last accessed: 12.04.2023).

20. Zhang Z. *A flexible new technique for camera calibration. IEEE transactions on pattern analysis and machine intelligence*. 2000. Vol. 22, No 11. P. 1330–1334. URL: https://doi.org/10.1109/34.888718 (Last accessed: 20.05.2023).

21. Zisserman A., Hartley R. *Multiple view geometry in computer vision*. 2-ге вид. Cambridge University Press, 2004. 672 с. (Last accessed: 20.05.2023).

22. Драйвер двигуна на L298N. URL: https://arduino.ua/prod406-draiver-dvyh-dvigatelei-na-l298n (дата звернення: 05.01.2023).

23. MakerBot Digitizer. URL: https://www.makerbot.com/3d-scanner/ (дата звернення: 05.01.2023).

24. David Starter Kit V2. URL: https://www.david-3d.com/en/products/hardware/starter-kit-2/ (дата звернення: 06.01.2023).

25. Ciclop / BQ. URL: https://www.bq.com/en/ciclop (дата звернення: 05.01.2023)..

26. Лазер червона лінія 5мВт з регулятором фокусу. URL: https://arduino.ua/prod617-lazer-krasnaya-liniya-5mvt-s-regyliryemim-fokysom (дата звернення: 05.01.2023).

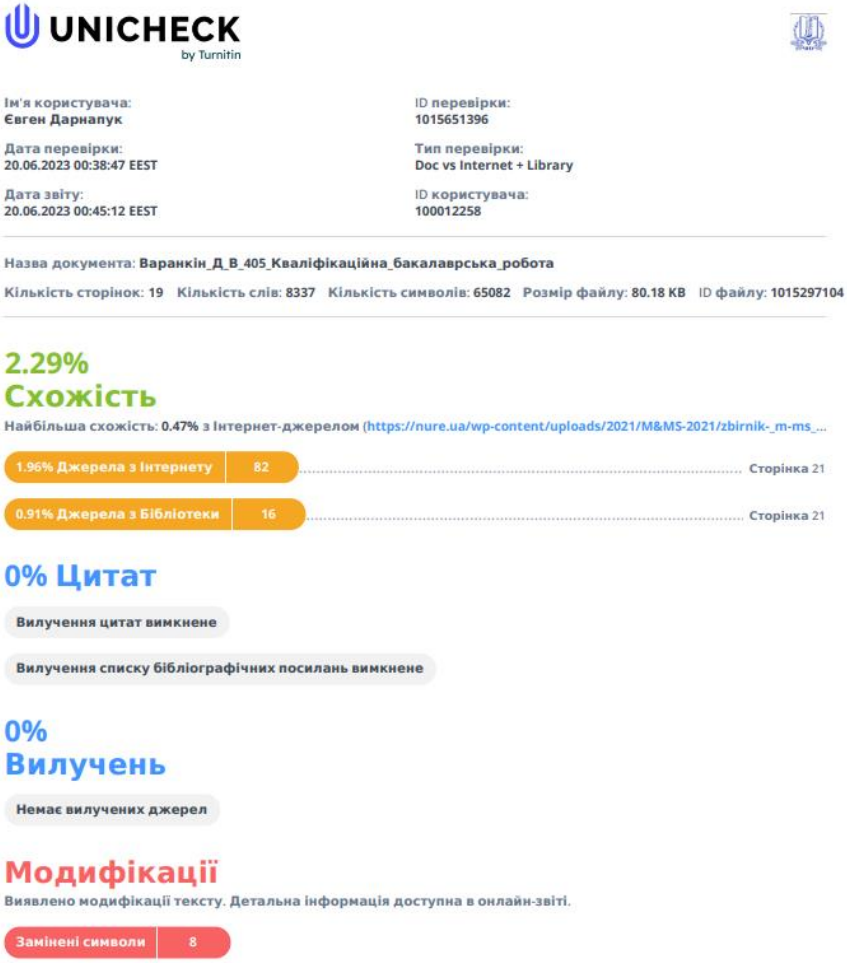
27. Крокoвий двигун JKongMotor NEMA17 JK42HS34-1334AC. Режим доступу: https://arduino.ua/prod2945-shagovii-dvigatel-jkongmotor-nema17-jk42hs34-1334ac (дата звернення: 05.01.2023).

ДОДАТОК А

ДОВІДКА

про перевірку на унікальність пояснювальної
записки кваліфікаційної бакалаврської роботи
на тему: «Апаратно-програмний комплекс 3D-сканування об'єктів на базі Arduino»
студента спеціальності 123 «Комп’ютерна інженерія»,
групи 405
Варанкіна Дениса Володимировича
прізвище, ім’я, по-батькові

Перевірку тексту здійснено сервісом: онлайн-сервіс Unicheck.
Результат перевірки тексту роботи: схожість складає 2.29 %.



Студент

_____Д. В. Варанкін
підпис ініціали, прізвище

Керівник
Ст. викладач

_____Є. С. Дарнапук
підпис ініціали, прізвище

Дата: «__»_____2023 р.

ДОДАТОК Б

Блок-схеми

Рисунок Б1 – Алгоритм роботи застосунку (частина 1)

Рисунок Б2 – Алгоритм роботи застосунку (частина 2)

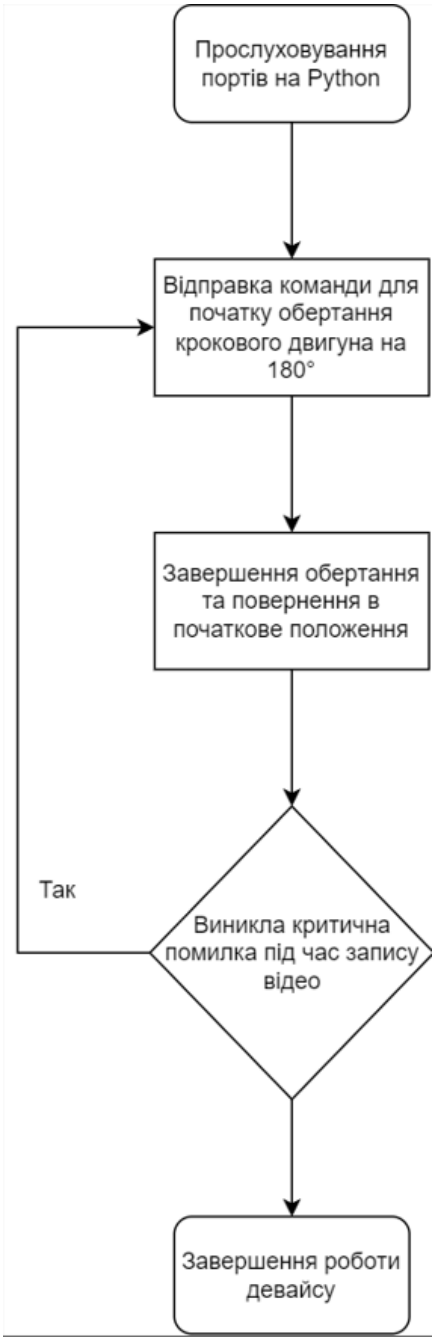


Рисунок Б3 – Алгоритм роботи девайсу

Рисунок Б4 – Алгоритм калібрування камери

ДОДАТОК В

Код для калібрування камери

```
import glob
import os
import sys
import getopt

import cv2
import numpy as np

from utils import outerContour

# Temp warped rectangle pattern size, proportional to the real size
warpedW = 700
warpedH = 900


def checkBlankArea(warped):
    """
    Check the mean of the area expected to be an empty chess.
    To align the chessboard image find a minimum of this value (white area).
    """
    roi = warped[75:160, 510:635]
    mean = cv2.mean(roi)
    return mean[0]


def removeXYSigns(warped):
    """
    Use a white rectangle to mask the XY signs, as they introduce noise
    """
    points = np.array(
        [[[97, 869], [36, 858], [27, 810], [74, 810], [94, 832]]])
    cv2.fillPoly(warped, points, (255, 255, 255))


def findRectanglePatterns(gray):
    """
    Find all the possible rectangle patterns in gray image, sorted by score.
    """

    # Threshold the image using Otsu algorithm
    thresh = cv2.threshold(
        gray, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)[1]

    # Find all the possible contours in thresholded image
    contours, hierarchy = cv2.findContours(
        thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)

    polys = []
    minContourLength = 300
    for contour in contours:
        if len(contour) >= minContourLength:
            # We approximate a polygon, we are only interested in rectangles (4
points, convex)
            epsilon = 0.05 * cv2.arcLength(contour, True)
            curve = cv2.approxPolyDP(contour, epsilon, True)
            if len(curve) == 4 and cv2.isContourConvex(curve):
                polys.append(curve)
```

```
    # We sort the found rectangles by score descending, using outerContour function
    # The function calculates the mean of the border inside the rectangle: it must be
around full black
    # It's safe to take the first entry as a valid pattern if it exists
    polys.sort(key=lambda x: outerContour(x, thresh), reverse=False)
    return polys


def findRectanglePatternHomography(gray):
    """
    Given a gray image, find the rectangle pattern and estimate homography matrix
    """

    # We use findRectanglePatterns and we keep the first (best) result
    polys = findRectanglePatterns(gray)
    biggerContour = polys[0]

    # We try estimating the homography and warping
    destPoints: np.ndarray = np.array(
        [[[0, warpedH]], [[0, 0]], [[warpedW, 0]], [[warpedW, warpedH]]])
    M = cv2.findHomography(biggerContour, destPoints)[0]
    warped = cv2.warpPerspective(gray, M, (warpedW, warpedH))

    # ...but it may be rotated, so we need to rectify our pattern.
    # To do this we iterate through all the possible 90 degrees rotations to find the
one with a blank tile (upper right).
    # We have the checkBlankArea function that returns the color of our check area,
we simply find the minimum.

    currMax = checkBlankArea(warped)
    for i in range(3):
        # Find homography and warped image with that rotation
        biggerContour = np.roll(biggerContour, shift=1, axis=0)
        M2 = cv2.findHomography(biggerContour, destPoints)[0]
        rotated = cv2.warpPerspective(gray, M2, (warpedW, warpedH))
        rotatedScore = checkBlankArea(rotated)
        if rotatedScore > currMax:
            M = M2
            warped = rotated
            currMax = rotatedScore

    removeXYSigns(warped)

    # We return the Homography, Corners and the Warped Image
    return M, biggerContour, warped


def genExpectedChessboardCorners(width=160, height=200, excludeTrickyPoints=True):
    """
    Generate the expected chessboard corners for our pattern
    """
    outerBorderPoints = [[0, 0], [160, 0], [160, 200], [0, 200]]
    innerBorderPoints = [[10, 10], [150, 10], [150, 190]]
    innerChessboardPoints = \
        [[20, j] for j in range(40, 180, 20)] + \
        [[i, j] for i in range(40, 120, 20) for j in range(20, 200, 20)] + \
        [[120, j] for j in range(40, 200, 20)] + \
        [[140, j] for j in range(60, 180, 20)]

    # We found out that the lower left corners (near the xy signs) are harder to find
    # As default we ignore them, unless the user wants to
```

```
    if excludeTrickyPoints == False:
        innerBorderPoints.append([10, 190])
        innerChessboardPoints.append([20, 180])

    innerPoints = innerBorderPoints + innerChessboardPoints
    transformedInnerPoints = [
        [[x / width * warpedW, y / height * warpedH]] for x, y in innerPoints]
    transformedOuterPoints = [
        [[x / width * warpedW, y / height * warpedH]] for x, y in outerBorderPoints]
    return np.array(transformedInnerPoints, dtype=np.float32),
np.array(transformedOuterPoints, dtype=np.float32)


def findChessboardCorners(H, rectangle, cropped, gray, useOuterPoints=False):
    """
    Given the homography, rectangle contour in the image and warped image,
    finds the inner chessboard corners, returning the image and object points
    """

    imagePoints = []
    objectPoints = []

    # We first find the expected corners positions
    innerTargetPoints, outerTargetPoints = genExpectedChessboardCorners()

    # ...then we refine the expected corners positions using cornerSubPix
    # This way the points are supposed to follow the corners precisely
    # We do this only with inner points
    zeroZone = (-1, -1)
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TermCriteria_COUNT, 2000, 0.01)
    refinedInnerPoints = cv2.cornerSubPix(
        cropped, innerTargetPoints, (32, 32), zeroZone, criteria)
    refinedOuterPoints = cv2.cornerSubPix(
        gray, outerTargetPoints, (4, 4), zeroZone, criteria)

    # We transform our found chessboard corners in the warped image
    # back to image points, inverting the homography matrix
    H_inv = np.linalg.inv(H)
    projectedInner = cv2.perspectiveTransform(refinedInnerPoints, H_inv)
    projectedOuter = cv2.perspectiveTransform(refinedOuterPoints, H_inv)

    # We had worst results with outer points, so maybe it's better to leave them
out...
    if useOuterPoints:
        innerTargetPoints.extend(outerTargetPoints)
        projectedInner.extend(projectedOuter)

    # We build our imagePoints and objectPoints arrays, using the found results
    for t, p in zip(innerTargetPoints, projectedInner):
        tx, ty = t[0]
        px, py = p[0]
        objectPoints.append([tx, ty, 0.0])
        imagePoints.append([px, py])

    return imagePoints, objectPoints


def run(debug=False):
    # From and to homography points
    imagePoints = []
    objectPoints = []
```

```
    i = 0
    for file in glob.glob("./calibration_images/*.jpg"):
        img = cv2.imread(file)
        gray = cv2.cvtColor(img, cv2.COLOR_RGBA2GRAY)

        # First we find our rectange pattern
        H, rectangle, cropped = findRectanglePatternHomography(gray)

        # Then we get our chessboard pattern corners
        imgImagePoints, imgObjectPoints = findChessboardCorners(
            H, rectangle, cropped, gray)
        imagePoints.append(imgImagePoints)
        objectPoints.append(imgObjectPoints)

        if debug:
            cv2.drawContours(img, [rectangle], -1, (255, 0, 0))
            for c in imgImagePoints:
                px, py = c
                cv2.circle(img, (px, py), 3, (0, 0, 255), -1)
            cv2.imshow(file, img)
            cv2.waitKey(1)

        print(f"Image {i}/50")
        i += 1

    imagePoints = np.array(imagePoints, dtype=np.float32)
    imagePoints = np.reshape(
        imagePoints, (imagePoints.shape[0], imagePoints.shape[1], 1, 2)
    )
    objectPoints = np.array(objectPoints, dtype=np.float32)

    # With our object and image points we can finally perform the calibration
    ret, K, dist, rvecs, tvecs = cv2.calibrateCamera(
        objectPoints, imagePoints, (img.shape[0], img.shape[1]), None, None
    )
    print(f"\n\nRMS: {ret}")
    print(f"\nK: {K}")
    print(f"Distortion parameters:\n{dist}")
    print(f"\nImages used for calibration: {imagePoints.shape[0]}/50")

    # We save our intrinsics parameters to file for later use
    Kfile = cv2.FileStorage("intrinsics.xml", cv2.FILE_STORAGE_WRITE)
    Kfile.write("RMS", ret)
    Kfile.write("K", K)
    Kfile.write("dist", dist)
    Kfile.release()
    print("Saved intrinsics in intrinsics.xml")

    if debug:
        cv2.waitKey(0)
        cv2.destroyAllWindows()


opts, args = getopt.getopt(sys.argv, "v")
debug = args.count("-v") > 0
run(debug)
```

ДОДАТОК Г

Код для сканування із відео

```
import cv2
import numpy as np
import open3d as o3d
import sys
import getopt

from utils import (
    loadIntrinsics,
    sortCorners,
    createRays,
    linePlaneIntersection,
    fitPlane,
    outerContour,
    findPlaneFromHomography,
    findPointsInsidePoly
)

# The reference fiducial rectangles sizes
rectWidth = 25
rectHeight = 15

# HSV ranges for laser line detection
hsvMin = (150, 20, 78)
hsvMax = (200, 255, 255)

# Kernels for opening-closing
kernel2 = np.ones((2, 2), np.uint8)
kernel4 = np.ones((4, 4), np.uint8)


def findRectanglePatterns(firstFrame):
    """
    Given the first frame, finds the 2D rectangular patterns in the image with float
precision.
    """

    # Threshold the image using Otsu algorithm
    gray = cv2.cvtColor(firstFrame, cv2.COLOR_RGBA2GRAY)
    thresh = cv2.threshold(
        gray, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)[1]

    # Find all the possible contours in thresholded image
    contours, hierarchy = cv2.findContours(
        thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)

    winSize = (16, 16)
    zeroZone = (-1, -1)
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TermCriteria_COUNT, 200, 0.1)
    minContourLength = 30
    polys = []
    for contour in contours:
        if len(contour) >= minContourLength:
            # We approximate a polygon, we are only interested in rectangles (4
points, convex)
            epsilon = 0.01 * cv2.arcLength(contour, True)
            curve = cv2.approxPolyDP(contour, epsilon, True)
            if len(curve) == 4 and cv2.isContourConvex(curve):
```

```
                # We use cornerSubPix for floating point refinement
                curve = cv2.cornerSubPix(gray, np.float32(
                    curve), winSize, zeroZone, criteria)
                sortedCurve = sortCorners(curve)
                score = outerContour(sortedCurve.astype(np.int32), gray)
                polys.append((sortedCurve, score))

    # We sort the found rectangles by score descending, using outerContour function
    # The function calculates the mean of the border inside the rectangle: it must be
around full black
    # It's safe to take the first entry as a valid pattern if it exists
    polys.sort(key=lambda x: x[1], reverse=False)
    return [p[0] for p in polys]


def findReference3DPoints(img, rect, plane, K_inv):
    """
    Given the thresholded laser, a rectangle for bounds, the rectangle plane and the
inverse
    camera matrix, finds all the 3D points inside the rectangle intersecting the
plane.
    This is used to find the reference 3D points inside the wall and desk rectangles.

              laser
     wall   /
       |  /
       x
    /  |_____ desk
    """

    # Find the 2D points inside the rectangle
    imgPoints = findPointsInsidePoly(img, rect.astype(np.int32))
    if imgPoints is None:
        return None, None

    # Create rays and find the intersections with plane
    homoImgPoints = np.hstack(
        (imgPoints[:, 0], np.ones(imgPoints.shape[0]).reshape(-1, 1),))
    rays = createRays(homoImgPoints, K_inv)
    points3D = [linePlaneIntersection(plane, ray) for ray in rays]
    return points3D, imgPoints


def processFrame(firstFrame, undistorted, K_inv, upperRect, lowerRect, upperPlane,
lowerPlane, debug=False):
    # Prepare the image for processing (thresholding) and find points
    hsv = cv2.cvtColor(undistorted, cv2.COLOR_BGR2HSV)
    inRange = cv2.inRange(hsv, hsvMin, hsvMax)
    final = cv2.morphologyEx(inRange, cv2.MORPH_OPEN, kernel4)
    laserPts = cv2.findNonZero(final)

    # DEBUG INFO
    if debug:
        if laserPts is not None:
            for p in laserPts:
                cv2.circle(undistorted, (p[0][0], p[0][1]), 1, (0, 0, 255))
        cv2.imshow('undistorted', undistorted)

    # Find reference points on desk and wall
    upper3DPoints, upperImgPoints = findReference3DPoints(
        final, upperRect, upperPlane, K_inv)
    lower3DPoints, lowerImgPoints = findReference3DPoints(
```

```
            final, lowerRect, lowerPlane, K_inv)

        # Then fit a plane if we have enough points
        if upper3DPoints is not None and lower3DPoints is not None:
            # Find the corrisponding laser plane
            referencePoints = np.array(upper3DPoints + lower3DPoints)
            laserPlane = fitPlane(referencePoints)

            # Find 3D points with line-plane intersection
            homoImgPoints = np.hstack(
                (laserPts[:, 0], np.ones(laserPts.shape[0]).reshape(-1, 1),))
            rays = createRays(homoImgPoints, K_inv)
            points3D = [linePlaneIntersection(laserPlane, ray) for ray in rays]

            # Recover colors for points from first frame
            x = laserPts.squeeze(1)
            colors = np.flip(firstFrame[x[:, 1], x[:, 0]].astype(
                np.float64) / 255.0, axis=1)
            return points3D, colors, laserPlane
        return None, None, None


def run(path, debug=False):
    # Load our camera parameters
    K, dist = loadIntrinsics()
    K_inv = np.linalg.inv(K)

    # Load our video and read the first frame
    # We will use the first frame to find the reference rectangles and colors for the
point cloud
    cap = cv2.VideoCapture(path)
    firstFrameDistorted = cap.read()[1]
    firstFrame = cv2.undistort(firstFrameDistorted, K, dist)

    # Finds the reference rectangles in the first frame
    polys = findRectanglePatterns(firstFrame)
    upperRect, lowerRect = polys[0:2]

    # DEBUG INFO
    if debug:
        firstFrameDbg = firstFrame.copy()
        cv2.drawContours(firstFrameDbg, [upperRect.astype(
            np.int32), lowerRect.astype(np.int32)], -1, (0, 0, 255))
        cv2.imshow("debug rect contours", firstFrameDbg)
        cv2.waitKey(1)

    # Find the WALL rectangle homography and plane
    upperDestPoints: np.ndarray = np.array(
        [[[0, rectHeight]], [[0, 0]], [[rectWidth, 0]], [[rectWidth, rectHeight]]])
    upperRectHomo = cv2.findHomography(
        sortCorners(upperDestPoints), upperRect)[0]
    upperPlane = findPlaneFromHomography(upperRectHomo, K_inv)

    # Find the DESK rectangle homography and plane
    lowerDestPoints: np.ndarray = np.array(
        [[[0, rectHeight]], [[0, 0]], [[rectWidth, 0]], [[rectWidth, rectHeight]]])
    lowerRectHomo = cv2.findHomography(
        sortCorners(lowerDestPoints), lowerRect)[0]
    lowerPlane = findPlaneFromHomography(lowerRectHomo, K_inv)

    # Here we store our found 3D cloud
    objPoints = []
```

```
    objColors = []

    isRecording = True
    while cap.isOpened():
        if isRecording:
            ret, frame = cap.read()
            if not ret:
                break

        undistorted = cv2.undistort(frame, K, dist)
        framePts, frameColors, laserPlane = processFrame(firstFrame, undistorted,
K_inv, upperRect, lowerRect,
                                                         upperPlane,
                                                         lowerPlane, debug)
        if framePts is not None and frameColors is not None:
            objPoints.extend(framePts)
            objColors.extend(frameColors)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
        if cv2.waitKey(1) & 0xFF == ord('p'):  # Pause
            isRecording = False
        if cv2.waitKey(1) & 0xFF == ord('c'):  # Continue
            isRecording = True

    # Finally, save our point cloud
    pcd = o3d.geometry.PointCloud()
    pcd.points = o3d.utility.Vector3dVector(
        np.vstack(objPoints).astype(np.float64))
    pcd.colors = o3d.utility.Vector3dVector(np.vstack(objColors))
    o3d.io.write_point_cloud("output.ply", pcd)
    o3d.visualization.draw_geometries([pcd])

    cap.release()
    cv2.destroyAllWindows()


opts, args = getopt.getopt(sys.argv, "v")
debug = args.count("-v") > 0
path = args[-1]
run(path, debug)
```

ДОДАТОК Д

Код для меню користувача

```
import tkinter as tk
import subprocess
import os

def btn1_click():
    os.startfile('cameraCalibrator.py')

def btn2_click():
    print("Кнопка 2 нажата")

def btn3_click():
    os.startfile('scanner.py')

window = tk.Tk()
window.geometry("300x200")  # установка размеров окна

btn4 = tk.Button(window, text="Зробити фотографії для калібрування",
command=btn1_click)
btn4.configure(width=30, height=2)  # установка размеров кнопки
btn4.pack()

btn1 = tk.Button(window, text="Калібравати камеру", command=btn1_click)
btn1.configure(width=30, height=2)  # установка размеров кнопки
btn1.pack()

btn2 = tk.Button(window, text="Зняти відео", command=btn2_click)
btn2.configure(width=30, height=2)  # установка размеров кнопки
btn2.pack()

btn3 = tk.Button(window, text="Сканувати з відео", command=btn3_click)
btn3.configure(width=30, height=2)  # установка размеров кнопки
btn3.pack()

window.mainloop()
```

ДОДАТОК Е

Код для управління девайсом

```
import serial
import time


arduino = serial.Serial('COM3', 9600)
time.sleep(2)


def send_command(command):
    arduino.write(command.encode())


def move_stepper(steps):
    send_command('M' + str(steps))


def control_laser(state):
    if state == 'on':
        send_command('L1')
    elif state == 'off':
        send_command('L0')


move_stepper(200)
time.sleep(1)
control_laser('on')
time.sleep(2)
control_laser('off')


arduino.close()
```