Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра автоматизації, комп'ютерних наук і технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеню вищої освіти – магістр
за освітньо-професійною програмою
«Кіберфізичні системи в промисловості, бізнесі та транспорті»

зі спеціальності
174 – Автоматизація, комп'ютерно-інтегровані технології та робототехніка

тема роботи:

«*Автоматизація процесу класифікації деталей на конвеєрі за габаритами та масою*»

Виконав ст. гр. АКІТР-23-1м.	_____	Прозоров О. П.
Керівник	_____	Савицький О. І.
Нормоконтроль	_____	Маринич І. А.
Завідувач кафедри	_____	Рубан С. А.

Кривий Ріг – 2024

КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет: *інформаційних технологій*

Кафедра: *автоматизації, комп'ютерних наук і технологій*

Ступінь вищої освіти: *Магістр*

Спеціальність: *174 – Автоматизація, комп'ютерно-інтегровані технології та робототехніка*

ЗАТВЕРДЖУЮ

Зав. кафедри: *к.т.н. Рубан С.А.*

_____

« *5* » *липня* 2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра

студентові групи АКІТР-23-1м. Прозорову Олексію Павловичу

1. Тема кваліфікаційної роботи: *«Автоматизація процесу класифікації деталей на конвеєрі за габаритами та масою і»*

затверджено наказом по університету № *595с* від *04.07.2024 р.*

2. Термін здачі кваліфікаційної роботи: *01.12.2024 р.*

3. Склад кваліфікаційної роботи: *Пояснювальна записка обсягом 78с., додатки, презентація у Microsoft PowerPoint (15 слайдів) в електронному та друкованому вигляді*

4. Консультанти кваліфікаційної роботи:

Розділ 1-3 доц. Савицький О. І.

Нормоконтроль доц. Маринич І. А.

5. Календарний план:

№	Етапи роботи	Термін виконання
1	*Вступ*	*10.07.24*
2	*Розділ 1*	*15.07.24*
3	*Розділ 2*	*18.08.24*
4	*Розділ 3*	*19.09.24*
5	*Висновки*	*15.10.24*
6	*Оформлення кваліфікаційної роботи*	*20.11.24*
7	*Підготовка презентації та графічного матеріалу*	*28.11.24*
8	*Підготовка доповіді до захисту*	*01.12.24*

6. Дата видачі завдання: *28.06.2024р.*

Керівник _____ /Савицький О. І./

7. Запевнення: *Я, Прозоров Олексій Павлович, запевняю, що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.*

Із чинним Положенням про академічну доброчесність Криворізького національного університету ознайомлений.

Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі умисних порушень робота не допускається до захисту або оцінюється незадовільно.

Здобувач _____ **/ Прозоров О. П.**

АНОТАЦІЯ

Прозоров О. П. Автоматизація процесу класифікації деталей на конвеєрі за габаритами та масою.

Кваліфікаційна робота на здобуття ступеню вищої освіти магістр за освітньо-професійною програмою «Кіберфізичні системи в промисловості, бізнесі та транспорті» зі спеціальності 174 – Автоматизація, комп'ютерно – інтегровані технології та робототехніка– Криворізький національний університет, Кривий Ріг, 2024.

Об'єктом керування є лабораторний стенд системи «маніпулятор-конвеєр» аудиторії №367 Siemens КНУ.

Метою роботи є розробка та реалізація системи автоматичного розпізнавання фізичних параметрів деталей, яка базується на методах комп'ютерного зору та зважування для забезпечення ефективного управління процесами на виробництві.

Зроблено аналіз створення слідкуючої системи, у якій реалізовано додатковий вплив, що в експлуатаційних умовах підвищує точність, практично гарантуючи рівність швидкодії вихідного і вхідного сигналів, та коректної роботи слідкуючої системи за фізичними характеристиками вантажу на конвеєрі, забезпечуючи умови ефективної роботи робота.

Представлено розрахунки визначення розмірів деталей на зображенні, корекції спотворень камери та кінематичних задач маніпулятора.

Також результатом виконання даної роботи є створення нових методичок для виконання лабораторних робіт, з використанням доданого обладнання, на стенді аудиторії Siemens №367.

Ключові слова:

СИСТЕМА АВТОМАТИЗАЦІЇ, КОМП'ЮТЕРНИЙ ЗІР, РОЗПІЗНАВАННЯ РОЗМІРІВ, ПРОГРАМУВАННЯ, ЗВАЖУВАННЯ, МОДЕРНІЗАЦІЯ.

ANNOTATION

Prozorov O. P. Automation of the process of classifying parts on the conveyor by size and weight.

Graduation master`s work for obtaining an educational degree «Master» for the educational and professional program « Cyber-physical systems in industry, business and transport» in specialty 174 – «Automation, computer-integrated technologies, and robotics». – Kryvyi Rih National University, Kryvyi Rih, 2024

The control object is the laboratory stand of the "manipulator-conveyor" system of the Siemens auditorium №367 of the KNU.

The purpose of the work is to develop and implement a system for automatic recognition of physical parameters of parts to be loaded onto the conveyor, which is based on computer vision and weighing methods to ensure accurate and effective control of production processes.

An analysis of the creation of a tracking system has been made, in which an additional effect is implemented, which in operating conditions increases accuracy, practically guaranteeing the equality of the speed of the output and input signals, and the correct operation of the tracking system according to the physical characteristics of the load on the conveyor, ensuring the conditions for effective operation of the robot.

Calculations for determining the dimensions of parts in the image, correction of camera distortions and kinematic tasks of the manipulator are presented.

Also, the result of this work is the creation of new methods for performing laboratory work, using the attached equipment, at the Siemens auditorium stand №367.

Keywords:

AUTOMATICS SYSTEM, COMPUTER VISION, DIMENSION RECOGNITION, PROGRAMMING, WEIGHING, MODERNIZATION.

ЗМІСТ

ВСТУП

Актуальність теми даної роботи зумовлена значним розвитком автоматизованих технологій на сучасних виробництвах, зокрема в галузі управління системами на основі комп'ютерного зору та зважування. Використання автоматизованих систем для розпізнавання та обробки даних про деталі, що переміщуються по конвеєру, дозволяє значно підвищити точність обробки та ефективність виробничих процесів. Це стає особливо важливим в умовах інтеграції виробничих систем у середовище Industry 4.0, де для забезпечення безперервної та високопродуктивної роботи необхідна точна синхронізація всіх етапів обробки та транспортування продукції. Технології комп'ютерного зору та зважування можуть вирішити низку проблем, таких як автоматичний контроль розмірів, ваги та якості продукції, що значно зменшує людський фактор та підвищує ефективність обробки.

Постановка та вирішення проблеми модернізації автоматизованих систем, зокрема системи розпізнавання деталей на конвеєрі за габаритами та масою, є важливим кроком на шляху до підвищення конкурентоспроможності виробництва. Однак, незважаючи на наявні досягнення в цій галузі, ряд аспектів потребує подальшого дослідження, зокрема вдосконалення алгоритмів розпізнавання та інтеграції з іншими виробничими системами. Це зумовило вибір теми роботи, яка охоплює як теоретичні, так і практичні аспекти реалізації таких систем.

Огляд літератури показав, що хоча існують окремі розробки в галузі автоматизованого розпізнавання та зважування деталей, багато аспектів цієї проблеми потребують глибшого вивчення, зокрема, в контексті інтеграції з іншими компонентами автоматизованих виробничих систем та удосконалення точності та швидкості обробки даних. Раніше отримані результати демонструють важливість таких підходів, але залишають відкритими питання щодо оптимізації програмних і апаратних рішень для досягнення максимальної ефективності системи.

Метою роботи є розробка та реалізація системи автоматичного розпізнавання фізичних параметрів деталей, що будуть завантажуватись на конвеєр, яка базується

на методах комп’ютерного зору та зважування для забезпечення точного і ефективного управління процесами на виробництві. Завданнями роботи є:

- Аналіз сучасних рішень у сфері автоматизованих систем розпізнавання та зважування;
- Розробка математичних моделей для обробки даних та розпізнавання характеристик деталей;
- Вибір і обґрунтування необхідного обладнання для реалізації системи;
- Створення програмного забезпечення для розпізнавання деталей та їх інтеграція в існуючу автоматизовану систему управління;
- Розробка інтерфейсу користувача для візуалізації процесу і керування системою.

Об'єктом дослідження є процеси автоматизованого розпізнавання та зважування деталей на конвеєрі, а предметом — алгоритми обробки зображень їх математичне моделювання та інтеграція цих процесів у систему автоматизованого управління.

Практичне значення одержаних результатів полягає в можливості впровадження розробленої системи на виробництвах, що дозволить підвищити ефективність контролю за продукцією, зменшити кількість помилок, а також забезпечити більш точне управління виробничими процесами. Розроблені алгоритми та програмне забезпечення можуть бути використані для модернізації існуючих автоматизованих ліній, а також як основа для створення нових систем на базі технологій комп’ютерного зору та зважування. Також це дає можливість для написання нових методичок лабораторних робіт для майбутніх студентів на базі встановленого обладнання та написаного програмного забепечення.

РОЗДІЛ 1
АНАЛІЗ ПІДХОДІВ ДО ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ПРОЦЕСУ

1.1 Аналіз технологічного процесу та огляд існуючих рішень інтегрованих автоматизованих систем управління виробництвом.

Технологічний процес виробництва на конвеєрній стрічці з маніпулятором є типовим прикладом автоматизованого виробництва, де всі етапи роботи з деталями відбуваються без втручання людини. Цей підхід забезпечує високу продуктивність, точність і надійність операцій. Проте, для підвищення ефективності і точності роботи, дана система потребує модернізації.

Сучасні автоматизовані системи управління виробництвом (АСУВ) спрямовані на максимальне використання можливостей інформаційних технологій для управління усіма аспектами виробництва. Вони включають в себе комплексні програмно-апаратні рішення, що забезпечують автоматизацію і оптимізацію виробничих процесів. Серед основних функцій таких систем можна виділити планування і диспетчеризацію виробництва, контроль якості, управління запасами, а також моніторинг і аналіз роботи обладнання [1].

В даному розділі розглядається декілька патентів за напрямком наукових досліджень. Цей етап є ключовим для визначення наявності аналогів та новизни у вибраному напрямку досліджень.

Система та метод визначення розмірів об'єкта (рис. 1.1)[2]

Номер патенту: US10937183B2

Анотація: Визначення розмірів об'єкта може включати в себе визначення відстані між об'єктом і пристроєм формування зображення, а також кута оптичної осі пристрою формування зображення. На зображенні об’єкта можна ідентифікувати одну чи більше ознак об’єкта. Розміри об’єкта можна визначити на основі відстані, кута й однієї чи кількох ідентифікованих особливостей.

Подано: 2019-01-28

Дата патенту: 2021-03-02

Правонаступник: Cognex Corp

Винахідники: José Fernandez-DoradoEmilio Pastor MiraFrancisco Azcona GuerreroIvan BachelderLaurens NunninkTorsten KempfSavithri VaidyanathanKyra MoedJohn Bryan Boatner

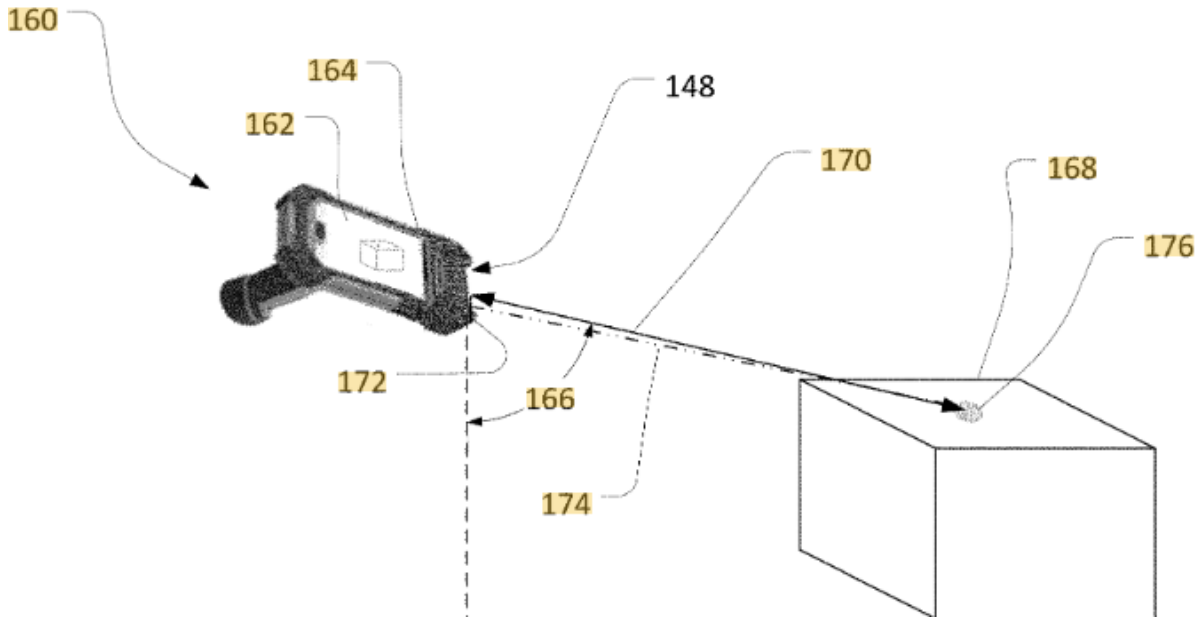


Рисунок 1.1 – Розпізнавання параметрів об’єкту

У багатьох контекстах може бути корисним визначати розміри об’єктів, наприклад паралелепіпедів або інших коробок. Наприклад, для оптимізації операцій пакування та доставки може бути корисним визначити тривимірний розмір окремих продуктів або упаковки. У деяких випадках об’єкти, для яких потрібно нанести розміри, можуть бути нерухомими. У деяких випадках предмети можуть рухатися.

Відстань зображення об’єкта можна визначити як відстань між об’єктом і частиною лінзи. Кут осі можна визначити як кут оптичної осі відносно системи відліку. Зображення об'єкта можна отримати за допомогою системи візуалізації. За допомогою процесорного пристрою можна ідентифікувати одну або декілька ознак об’єкта на зображенні. Використовуючи пристрій процесора, можна ідентифікувати один або більше розмірів об’єкта на основі найменшої відстані зображення об’єкта, кута осі та однієї або кількох ідентифікованих особливостей об’єкта.

Спосіб параметризації освітлювального приладу для машинного зору [3]

Номер патенту: 11395382

Анотація: Спосіб параметризації приладу машинного зору. Спосіб включає в себе наступні послідовні етапи: a) вибір залежно від зони інтересу конфігурації підпорядкованого освітлення та запис конфігурації підпорядкованого освітлення в інтерфейсі керування та/або застосування конфігурації підпорядкованого освітлення; b) незалежно від етапу a), вибір конфігурації початкового освітлення та застосування конфігурації початкового освітлення; c) встановлення інформаційного зв'язку між згаданою вихідною та підлеглою конфігураціями освітлення та запис зазначеної залежності в інтерфейсі керування таким чином, щоб електронний блок наказував освітлювати зону інтересу в підпорядкованій конфігурації освітлення у відповідь на вихідну конфігурацію освітлення.

Подано: 1 листопада 2019 р

Дата патенту: 19 липня 2022 р

Правонаступник: TPL VISION UK LTD

Винахідники: Guillaume Mazeaud, Jack McKinley, NEdko Gatev

У винаході пропонується спосіб параметризації пристрою машинного зору, який містить:

Систему камери, що містить камеру та джерела світла першого та другого джерел, виконані з можливістю, у конфігурації джерела освітлення, випромінювати світлові сигнали першого та другого джерел відповідно;

Освітлювальний прилад, інтерфейс керування, що містить:

Перший і другий датчики світла, виконані з можливістю захоплення згаданих першого і другого вихідних світлових сигналів, відповідно, і випромінювання першого і другого сигналів датчиків, відповідно, у відповідь на захоплення згаданих першого і другого вихідних світлових сигналів відповідно;

Електронний блок для обробки згаданих першого і другого сигналів датчиків, причому згаданий електронний блок обробки виконаний з можливістю відправляти в результаті зазначеної обробки безліч керуючих сигналів на відповідні виходи електронного блоку обробки;

Безліч підлеглих джерел світла, кожне з яких підключено до одного відповідного виходу електронного блоку обробки, щоб керувати ними за допомогою керуючого сигналу, що подається на зазначений вихід, щоб освітлити зону інтересу.

Зазначений спосіб включає наступні послідовні етапи:

a) вибір, залежно від зони інтересу, конфігурації освітлення для підпорядкованих джерел світла, яка називається «конфігурація підпорядкованого освітлення», і запис конфігурації підпорядкованого освітлення в інтерфейсі керування та/або застосування конфігурації підпорядкованого освітлення;

b) незалежно від кроку a), вибір конфігурації освітлення для вихідних джерел світла, або «конфігурація початкового освітлення», і застосування конфігурації вихідного освітлення;

c) встановлення інформаційного зв’язку між згаданою вихідною та підпорядкованою конфігураціями освітлення та запис зазначеного інформаційного відношення в інтерфейсі керування таким чином, щоб електронний блок наказував освітлювати зону інтересу в конфігурації підпорядкованого освітлення у відповідь на прийом зазначеними датчиками вихідні світлові сигнали в конфігурації початкового освітлення.

Комп’ютерний зір і пошук характеристик об’єкту за зображенням (рис. 1.2)[4]

Номер патенту: US20190205962A1

Анотація: Описано комп'ютерний зір і пошук характеристик зображення. Описана система використовує методи візуального пошуку шляхом визначення візуальних характеристик об’єктів, зображених на зображеннях, і порівняння визначених характеристик з візуальними характеристиками інших зображень, наприклад, для ідентифікації подібних візуальних характеристик в інших зображеннях. У деяких аспектах описана система виконує пошук, який використовує цифрове зображення як частину пошукового запиту, щоб знайти цікавий цифровий вміст. Описана система має кілька інструментів інтерфейсу користувача, які включають зображення візерунків, текстур або матеріалів і які

можна вибрати для ініціювання візуального пошуку цифрового вмісту, що має подібний малюнок, текстуру або матеріал. Описані аспекти також включають аутентифікацію на основі шаблонів, у якій система визначає автентичність предмета на зображенні на основі подібності його візуальних характеристик до візуальних характеристик відомих автентичних предметів.

Подано: 2018-12-28

Дата патенту: 2019-07-04

Правонаступник: eBay Inc

Винахідники: Robinson PiramuthuTimothy Samuel KeeferKenneth Clark CrookstonAshmeet Singh RekhiNiaz Ahamed Khaja NazimudeenPadmapriya GudipatiShane LinJohn F. WeigelFujun ZhongSuchitra RameshMohammadhadi KiapourShuai ZhengAlberto Ordonez PereiraRavindra Surya LankaMd Atiq ul IslamNicholas Anthony WhyteGiridharan IyengarBryan Allen Plummer

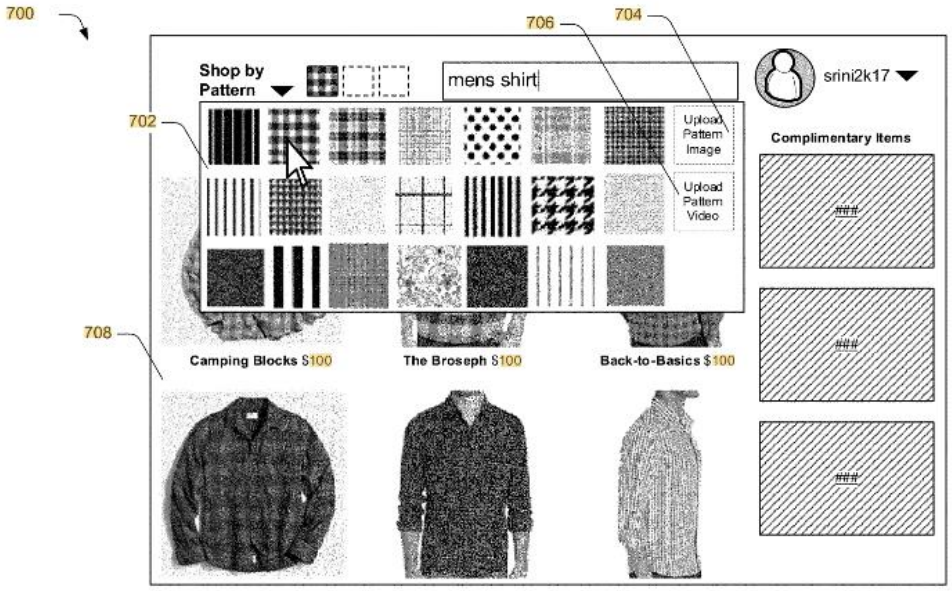


Рисунок 1.2 – Приклад розпізнавання форм

Традиційні системи текстового пошуку покладаються на те, що користувачі висловлюють цілі пошуку за допомогою тексту, що може працювати добре для простих запитів, таких як «знайти червоні кросівки». Однак ці системи мають труднощі, коли мету пошуку складно сформулювати в тексті. Крім того, необхідне спільне розуміння між користувачами, які перераховують елементи, і тими, хто їх шукає, що призводить до потенційних неточностей в описах елементів. Текстові

описи та зображення, надані користувачами, також можуть бути ненадійними, що призводить до подальшого розповсюдження неточностей у списках.

Для вирішення цих проблем цифрове середовище використовує комп'ютерний зір для пошуку характеристик зображення. Замість порівняння текстових запитів з текстовими даними, система використовує методи візуального пошуку, визначаючи візуальні характеристики об'єктів на зображеннях і порівнюючи їх для виявлення схожих візуальних характеристик на інших зображеннях. Система полегшує пошук, включаючи в запити цифрові зображення, що дозволяє ідентифікувати характеристики, що важко піддаються опису, такі як візерунки або конкретні форми. Система також пропонує кілька варіантів інтерфейсу користувача, що містять зображення візерунків, текстур або матеріалів для ініціювання візуального пошуку аналогічного цифрового контенту. Аутентифікація на основі шаблону є ще одним аспектом, при якому система перевіряє справжність предмета на основі візуальної схожості з відомими автентичними предметами, наприклад малюнків рядка та руху компонентів.

Системи та методи виявлення об'єктів, включаючи оцінку положення та розміру [5]

Номер патенту: 20240351724

Анотація: Даний винахід відноситься до систем і способів для виявлення об'єктів і оцінки положення в 3D на основі 2D-зображень. Виявлення об'єктів може виконуватися за допомогою моделі машинного навчання, налаштованої визначення різних властивостей об'єкта. Реалізації згідно з винаходом можуть використовувати ці властивості для оцінки положення та розміру об'єкта.

Подано: 2022-02-18

Дата патенту: 2024-11-02

Правонаступник: Google Llc

Винахідники: Robinson PiramuthuTimothy Samuel KeeferKenneth Clark CrookstonAshmeet Singh RekhiNiaz Ahamed Khaja NazimudeenPadmapriya GudipatiShane LinJohn F. WeigelFujun ZhongSuchitra RameshMohammadhadi

KiapourShuai ZhengAlberto Ordonez PereiraRavindra Surya LankaMd Atiq ul IslamNicholas Anthony WhyteGiridharan IyengarBryan Allen Plummer

Реалізований комп'ютером спосіб виявлення тривимірних (3D) об'єктів у двовимірних (2D) зображеннях, причому спосіб включає: отримання одним або більше обчислювальними пристроями двовимірного зображення, яке включає об'єкт, при цьому двовимірне зображення містить безліч пікселів; введення за допомогою одного або більше обчислювальних пристроїв двовимірного зображення модель виявлення об'єктів з машинним навчанням, що містить безліч головок, при цьому безліч головок містять: першу головку, сконфігуровану для створення теплової карти центроїду, при цьому теплова карта центроїда забезпечує відповідне значення тепла для кожного з множини пікселів, при цьому відповідне значення тепла для кожного пікселя описує ймовірність того, що центроїд об'єкта зображений таким пікселем; і другу головку, виконану з можливістю генерування безлічі полів відстаней, при цьому кожне поле відстаней забезпечує відповідне значення відстані для кожного з множини пікселів, при цьому відповідне значення відстані для кожного пікселя вказує відстань до однієї з безлічі вершин, пов'язаних з обмежує об'єм; генерацію одним або декількома обчислювальними пристроями теплової карти центроїду з першою головкою моделі виявлення об'єктів з машинним навчанням; генерацію одним або декількома обчислювальними пристроями безлічі полів відстаней з моделі виявлення об'єктів з машинним навчанням; та визначення за допомогою одного або більше обчислювальних пристроїв і на основі, щонайменше, теплової карти центроїду та безлічі полів відстаней, набору даних, що містить тривимірні координати для обмежуючої рамки, пов'язаної з об'єктом у двовимірному зображенні.

Система зважування з використанням конвеєрної стрічки з тензодатчиками (рис. 1.3) [6]

Номер патенту: 20240351724

Анотація: Модуль конвеєрної стрічки, конвеєрна стрічка, система зважування та спосіб зважування виробів, що транспортуються на конвеєрній стрічці. Конвеєрна стрічка включає набір датчиків навантаження, вбудованих в

модулі стрічки, для вимірювання сил, нормальних до транспортуючої поверхні стрічки. Вимірювання передаються зі стрічки на пульт дистанційного керування як частина системи технічного зору, яка ідентифікує окремі предмети, що транспортуються на стрічці, і притискає розташовані нижче тензодатчики. Система технічного зору визначає, які датчики ваги лежать в основі окремих предметів, та поєднує їх виміри для оперативного розрахунку ваги предметів.

Подано: 2012-08-11

Дата патенту: 2014-08-07

Правонаступник: Laitram LLC

Винахідники: Matthew L. Fourney

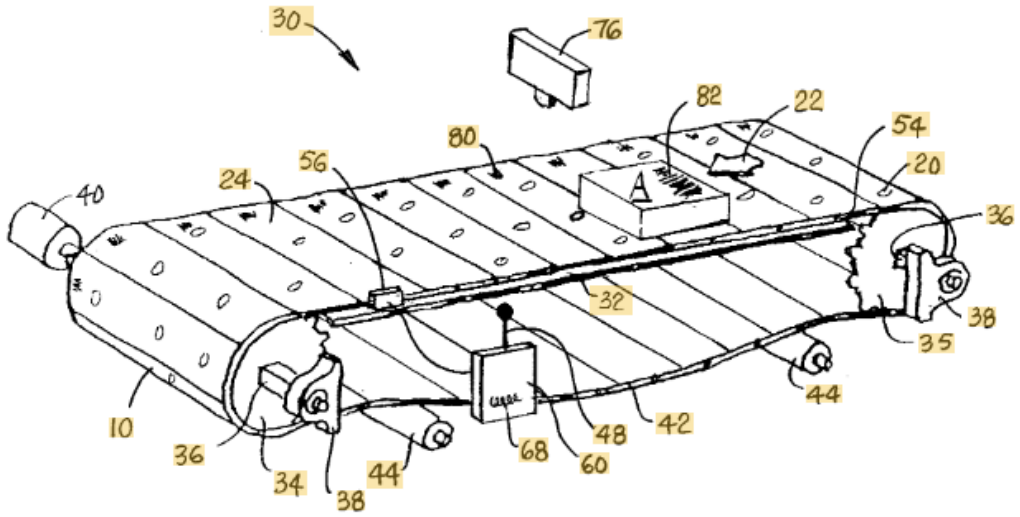


Рисунок 1.1 – Схема конвеєру з тензодатчиками

Даний винахід відноситься до конвеєрів з механічним приводом, зокрема, до конвеєрних систем, що включають тензодатчики всередині стрічки для індивідуального зважування предметів, що транспортуються в режимі реального часу.

У деяких випадках необхідно зважувати окремі предмети під час їх транспортування конвеєрною стрічкою. Традиційні конвеєрні системи вимагають поділу предметів перед тим, як вони потраплять на станцію зважування, де вони послідовно зважуються. Для цього перед станцією зважування додаються окремі секції конвеєра, щоб вирівняти товари в один рівень. Однак інтеграція розділових

секцій збільшує довжину конвеєра і зменшує корисну ширину поверхні, що транспортує.

Щоб подолати ці проблеми, даний винахід представляє конвеєрну систему з унікальними характеристиками. Один з варіантів цієї конвеєрної системи включає конвеєрну стрічку і систему комп’ютерного зору. Конвеєрна стрічка є двовимірним масивом датчиків навантаження, розподілених по її поверхні. Кожен тензодатчик вимірює силу, нормальну до поверхні стрічки у своєму певному положенні, генеруючи сигнал тензодатчика з вимірюванням відхилення прикладеної сили. Датчики, пов'язані з тензодатчиками, передають відповідні виміри тензодатчиків з ременя. Тим часом, система комп’ютерного зору ідентифікує окремі предмети на конвеєрній стрічці та створює відповідності між конкретними датчиками ваги та кожним ідентифікованим предметом. Окрім того, використання сучасних технологій, таких як Інтернет речей (IoT) та машинне навчання, дозволяє підвищити рівень автоматизації та інтелектуалізації виробничих процесів. Наприклад, платформи на базі IoT дозволяють збирати дані з різних сенсорів і пристроїв, аналізувати їх за допомогою алгоритмів машинного навчання і приймати оптимальні рішення для управління виробництвом.

Загалом, існуючі рішення для інтеграції камер та ваг у автоматизовані системи управління виробництвом дозволяють значно підвищити точність і ефективність виробничих процесів, забезпечуючи високий рівень контролю якості та оптимізацію витрат.

1.2 Архітектури систем на базі Industry 4.0.

Проектування загальної архітектури системи на базі еталонних архітектур Industry 4.0 передбачає створення інтегрованої і масштабованої системи, що забезпечує автоматизацію, контроль і оптимізацію виробничих процесів. Ця архітектура включає кілька основних компонентів: сенсори і виконавчі механізми, мережеву інфраструктуру для передачі даних, аналітичні інструменти для обробки великих даних, системи управління виробництвом (MES) та інтеграцію з

підприємницькими ресурсами (ERP) [7].

Загальна архітектура системи

1. Сенсори і виконавчі механізми

Система починається з набору сенсорів, що включають датчик ваги і камеру з машинним зором. Датчик ваги розташований під площадкою для завантаження деталей на конвеєр, що дозволяє точно зважувати кожну деталь до її потрапляння на конвеєр. Камера з машинним зором розміщується так, щоб охоплювати всю область завантаження деталей на конвеєр, забезпечуючи точну візуальну ідентифікацію розмірів кожної деталі. Ці сенсори забезпечують безперервний збір даних, які передаються до центральної системи обробки.

2. Мережева інфраструктура

Зібрані дані передаються через мережеву інфраструктуру, яка може бути побудована на основі локальних мереж (LAN) або бездротових мереж (Wi-Fi, IoT).

Мережа повинна забезпечувати високу швидкість передачі даних та надійність з’єднання для забезпечення безперервного моніторингу та управління виробничими процесами.

3. Аналітичні інструменти та обробка даних

Зібрані дані зберігаються в централізованій базі даних, яка може бути розміщена на хмарних сервісах для забезпечення масштабованості та доступності. Аналітичні інструменти використовуються для обробки великих обсягів даних, отриманих від сенсорів. Ці інструменти застосовують методи машинного навчання та штучного інтелекту для аналізу даних, прогнозування потреб у технічному обслуговуванні та виявлення аномалій у виробничому процесі.

4. Система управління виробництвом (MES)

Система управління виробництвом (MES) інтегрує всі компоненти системи і забезпечує централізоване управління виробничими процесами. MES відповідає за планування, моніторинг і контроль всіх виробничих операцій в режимі реального часу. Вона отримує дані з сенсорів і виконавчих механізмів, аналізує їх і приймає рішення щодо оптимізації виробничих процесів.

5. Інтеграція з підприємницькими ресурсами (ERP)

MES інтегрується з системами управління підприємницькими ресурсами (ERP), що дозволяє узгоджувати виробничі процеси з бізнес-процесами підприємства. ERP забезпечує управління ресурсами, постачанням, фінансами та іншими аспектами діяльності підприємства, що дозволяє оптимізувати виробництво з урахуванням загальної стратегії підприємства.

Деталізований опис процесу модернізації

Для реалізації зазначеної архітектури необхідно виконати кілька ключових кроків. Перший крок включає встановлення датчика ваги та камери з машинним зором. Датчик ваги обирається з урахуванням потреб виробництва, встановлюється під площадкою для завантаження деталей та калібрується для точного вимірювання. Камера з високою роздільною здатністю встановлюється так, щоб охоплювати область завантаження деталей на конвеєр, і підключається до системи обробки зображень для визначення розмірів деталей.

Наступний крок передбачає створення мережевої інфраструктури для передачі даних від сенсорів до центральної системи обробки. Для цього може бути використана локальна мережа або бездротові технології, залежно від умов виробництва. Важливо забезпечити високу швидкість передачі даних та надійність з’єднання.

Далі, зібрані дані зберігаються в централізованій базі даних, яка може бути розміщена на хмарних сервісах. Використання хмарних обчислень дозволяє масштабувати систему за потребою та забезпечувати доступність даних з будь-якої точки світу. Аналітичні інструменти застосовуються для обробки даних, виявлення закономірностей та аномалій, а також для прогнозування потреб у технічному обслуговуванні.

Інтеграція всіх компонентів здійснюється за допомогою системи управління виробництвом (MES), яка відповідає за моніторинг та контроль виробничих процесів в режимі реального часу. MES взаємодіє з системами управління підприємницькими ресурсами (ERP), що дозволяє оптимізувати виробництво з урахуванням бізнес-цілей підприємства.

Впровадження цієї архітектури забезпечує високий рівень автоматизації,

точності та ефективності виробничих процесів. Інтеграція IoT, великих даних, AI, хмарних обчислень, CPS та робототехніки створює умови для постійного вдосконалення та оптимізації виробництва, що дозволяє підприємству залишатися конкурентоспроможним у сучасному ринковому середовищі. Модернізація системи з маніпулятором і конвеєром через впровадження цих технологій дозволяє забезпечити високий рівень контролю якості та оперативно реагувати на будь-які зміни у виробничому процесі, сприяючи створенню ефективних і надійних виробничих систем [8].

1.3 Технічне завдання на розробку системи.

Модернізація системи з маніпулятором і конвеєром шляхом додавання та налаштування датчика ваги та камери з машинним зором для візуальної ідентифікації розмірів деталі може значно підвищити ефективність та точність системи. Опис процесу модернізації включає кілька ключових етапів.

Перший етап – встановлення та налаштування датчика ваги. Для цього потрібно обрати високоточний датчик, який відповідатиме конкретним потребам виробництва. Датчик необхідно розмістити під площадкою, з якої завантажуються деталі, що дозволяє точно зважувати кожну деталь до її потрапляння на конвеєр. Правильне калібрування датчика є важливим для забезпечення точних вимірювань, тому цей процес потребує уваги.

Другий етап – встановлення візуальної системи. Вибір відповідної камери з достатньо високою роздільною здатністю є ключовим для забезпечення точної візуальної ідентифікації розмірів деталей. Камеру має бути розміщено так, щоб вона охоплювала всю область з якої виконується завантаження деталей на конвеєр, що дозволяє отримувати чіткі зображення кожної деталі. Далі необхідно розробити програмне забезпечення для обробки зображень, яке зможе визначати розміри деталей з високою точністю.

Інтеграція цих компонентів з існуючою системою передбачає також налаштування програмного забезпечення для збору та аналізу даних з датчика ваги

і камери. Ці дані можуть використовуватися для автоматизації процесів контролю, що дозволяє швидко виявляти і реагувати на відхилення від встановлених стандартів. Це також сприяє оптимізації виробничих процесів за рахунок більш точного визначення ваги та розмірів деталей, що дозволяє зменшити кількість браку та підвищити загальну ефективність системи.

1.4 Розробка структури комплексу технічних засобів для реалізації системи.

Розробка структури комплексу технічних засобів для реалізації системи охоплює аналіз компонентів, їх взаємозв'язків і функціональних можливостей. Структура включає використання програмних засобів TIA Portal, Node-RED і Python у поєднанні з відповідним обладнанням. Сенсори та виконавчі механізми відіграють ключову роль у зборі даних і взаємодії із системою. Датчик ваги, встановлений під платформою завантаження деталей, забезпечує точне вимірювання маси, а камера з машинним зором, розташована для охоплення області завантаження деталей, відповідає за отримання візуальних даних для подальшого аналізу.

Мережева інфраструктура забезпечує передачу даних від сенсорів до центральної системи. Використовуються локальна мережа (LAN) або бездротові технології, такі як Wi-Fi чи IoT, що забезпечують гнучкість і надійність зв'язку. Центральна система обробки й керування реалізована на основі кількох програмних інструментів. TIA Portal використовується для програмування контролерів Siemens, налаштування мережевих з'єднань і моніторингу промислових процесів. Node-RED забезпечує побудову логіки управління й обробки даних через графічне середовище, що дозволяє швидко налаштовувати зв'язки між різними елементами системи. Python відповідає за розробку алгоритмів розпізнавання зображень і аналізу даних, що забезпечує гнучкість і точність роботи системи.

Виконавчі механізми, такі як маніпулятор і конвеєр, відповідають за фізичне

переміщення та обробку деталей у виробничому процесі. Ця структура створює комплексну систему, яка дозволяє контролювати, моніторити та оптимізувати виробничі процеси, забезпечуючи високу ефективність і інтеграцію з бізнес-процесами підприємства. Використання програмних засобів TIA Portal, Node-RED і Python у поєднанні з сучасним обладнанням забезпечує надійність, продуктивність і масштабованість системи, відповідаючи сучасним вимогам автоматизації виробництва

1.5 Обґрунтування вибору програмного забезпечення для реалізації системи.

Вибір програмного забезпечення для реалізації системи ґрунтується на кількох ключових аспектах, серед яких виділяються функціональність, можливість інтеграції з іншими системами, продуктивність, надійність і якість підтримки. Враховуючи ці критерії, для реалізації проекту було обрано програмні інструменти TIA Portal, Node-RED і Python, кожен з яких має свої переваги та функціональні особливості, що забезпечують ефективну розробку, управління та інтеграцію системи.

TIA Portal представляє собою інтегроване середовище програмування від Siemens, яке надає багатий набір можливостей для програмування промислових контролерів, конфігурації мережевих зв'язків і моніторингу технологічних процесів. Його обґрунтований вибір пояснюється широким функціоналом, який охоплює підтримку програмування контролерів, таких як SIMATIC S7-1200 та S7-1500, що є стандартом для сучасних промислових систем. Крім цього, TIA Portal забезпечує використання різних мов програмування, включно з Ladder Logic, Structured Text і SCL, що дозволяє реалізовувати складні алгоритми управління та обробки даних. Висока інтеграція з обладнанням Siemens, таким як контролери, датчики й приводи, значно полегшує процеси розробки, налаштування та подальшого обслуговування систем. Ще однією важливою перевагою TIA Portal є надійність і підтримка, оскільки Siemens відомий високою якістю своїх рішень,

активною підтримкою клієнтів та регулярними оновленнями програмного забезпечення [9].

Node-RED – це інший важливий компонент, що використовується в системі. Це візуальне середовище програмування, яке дозволяє створювати логіку управління й обробки даних через графічний інтерфейс. Його головною перевагою є висока гнучкість і швидкість розробки, що досягається завдяки інтуїтивному підходу до роботи з вузлами, які можна легко перетягувати й з'єднувати між собою. Це особливо важливо для зменшення часу розробки системи та спрощення налаштування її функціональності. Node-RED підтримує інтеграцію з багатьма пристроями й сервісами, що значно розширює можливості його використання у промислових системах. Крім цього, активна спільнота користувачів і наявність багатьох відкритих розширень роблять його універсальним і потужним інструментом для різних задач автоматизації та аналізу даних.

Python, як універсальна мова програмування, також займає важливе місце в архітектурі системи. Його популярність пояснюється зручністю використання, простим і зрозумілим синтаксисом, що дозволяє швидко розробляти програми навіть користувачам із базовими знаннями програмування. Багата екосистема бібліотек і модулів, доступна для Python, забезпечує широкий спектр можливостей, таких як обробка даних, машинне навчання, аналіз, візуалізація та багато іншого. Це робить Python оптимальним вибором для реалізації комплексних алгоритмів обробки інформації та інтеграції з іншими системами. Висока сумісність Python із різними мовами програмування й платформами дозволяє легко розширювати функціональність розроблюваної системи та забезпечувати її безперебійну взаємодію з існуючими компонентами.

Об'єднання можливостей TIA Portal, Node-RED і Python створює, гнучку та багатофункціональну систему управління й обробки даних, яка відповідає сучасним вимогам промислового виробництва. Така система забезпечує високу ефективність і надійність у виконанні своїх функцій, що сприяє оптимізації виробничих процесів, підвищенню їхньої продуктивності та якості [10].

Висновки до розділу:

Було проаналізовано технологічний процес та використання комп’ютерного зору на виробництвах.

Аналіз існуючих систем комп’ютерного зору та систем зважування дав змогу дослідити різноманітні пристрої, варіанти їх використання та фізичної реалізації, їх переваги та недоліки. На основі чого було прийнято рішення про проведення модернізації лабораторного стенду Siemens в КНУ, що також створить нові лабораторні роботи для майбутніх студентів закладу.

Було поставлено конкретну задачу модернізації, розроблено структуру комплексу технічних засобів, що будуть для цього використані та обрано в яких середах розробки буде виконуватись програмна реалізація системи.

РОЗДІЛ 2
ОПИС АПАРАТНОЇ ЧАСТИНИ, МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ ТА СИНТЕЗ КЕРУВАННЯ ПРОЦЕСОМ

2.1 Вибір датчиків та необхідних пристроїв для виконання модернізації

Для виконання модернізації лабораторного стенду Siemens в якості датчику для розпізнавання розмірів деталей було прийнято рішення використовувати веб камеру (рис. 2.1). Використання веб-камери як датчика для зчитування розмірів об'єктів із зображення має низку переваг, що обґрунтовують її вибір у багатьох проектах. Веб-камери є доступними, недорогими та легко інтегруються з комп’ютерними системами, що робить їх економічно вигідним рішенням для проектів, де необхідно забезпечити функціонал вимірювання без значних витрат. Сучасні веб-камери забезпечують достатню роздільну здатність для отримання якісних зображень, які можна обробляти за допомогою алгоритмів комп'ютерного зору, таких як OpenCV. Завдяки таким алгоритмам можливо вимірювати розміри об'єктів, використовуючи визначення контурів, аналіз геометрії чи порівняння масштабів.

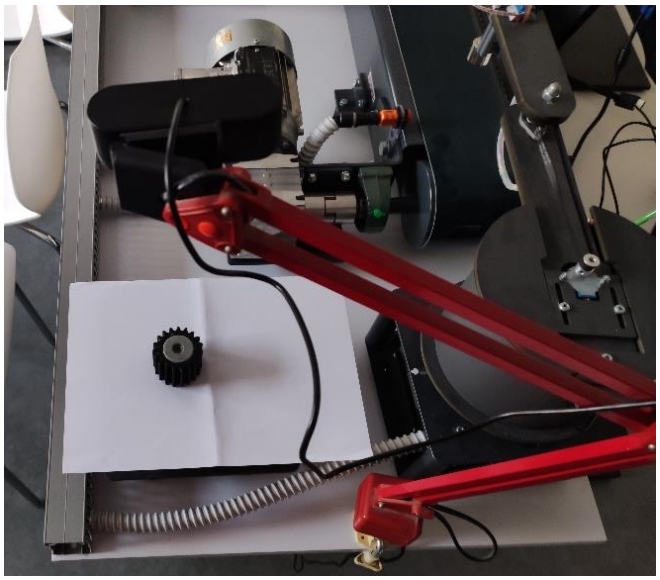


Рисунок 2.1 – Зовнішній вигляд встановлення камери на стенді

Універсальність веб-камер дозволяє застосовувати їх для аналізу об'єктів різної форми, розміру та розташування. Програмні інструменти забезпечують

автоматизацію процесу розпізнавання та вимірювання, а калібрування камери за допомогою еталонних об'єктів дає змогу досягти високої точності. Веб-камери також підтримують роботу в реальному часі, що робить їх зручними для використання в системах моніторингу, контролю або аналізу динамічних процесів. Їх легко інтегрувати з різними платформами, такими як комп’ютери, Raspberry Pi чи спеціалізовані контролери, що дозволяє створювати автоматизовані вимірювальні системи [11].

Однак є й виклики, які необхідно враховувати. Для точного вимірювання потрібно калібрувати камеру з урахуванням перспективних спотворень та фокусної відстані. Крім того, якість вимірювання залежить від стабільності освітлення, тому важливо забезпечити належні умови для роботи системи. Програмна реалізація вимірювань вимагає розробки або налаштування алгоритмів аналізу. Незважаючи на ці виклики, веб-камера є ефективним та практичним рішенням для задач вимірювання, особливо для низькобюджетних проектів чи створення прототипів.

Використання SIWAREX WP231, вагового модуля від Siemens, обґрунтовується його технічними характеристиками, функціональністю та адаптивністю до широкого спектру застосувань, а також тим що його вже було раніше встановлено на відповідний стенд аудиторії і в даному випадку виконується його калібрування та налаштування параметрів для досягнення необхідної точності зважування та коректної роботи. Цей модуль є частиною системи автоматизації SIMATIC S7 і розроблений спеціально для точного вимірювання маси, інтеграції з автоматизованими системами керування та обліку.

SIWAREX WP231 забезпечує високу точність вимірювань завдяки підтримці сучасних технологій обробки сигналів. Його роздільна здатність дозволяє визначати навіть незначні зміни у вазі, що є критично важливим у таких сферах, як контроль дозування, пакування чи моніторинг продукції в реальному часі. Модуль підтримує багатофункціональність, що включає вбудовані функції калібрування, компенсації похибок та тарування, що спрощує налаштування та експлуатацію системи.

Інтеграція SIWAREX WP231 із SIMATIC S7-1200 дозволяє реалізовувати складні алгоритми керування ваговими процесами в межах єдиної платформи автоматизації. Завдяки підтримці таких інтерфейсів, як Modbus TCP/IP і RS485, модуль можна легко інтегрувати в існуючі системи автоматизації, навіть якщо вони базуються на сторонньому обладнанні. Це робить його універсальним рішенням для різних галузей промисловості.

Крім того, SIWAREX WP231 має високу надійність та стабільність у роботі навіть за умов значних температурних коливань або електромагнітних перешкод, що є важливим для промислових умов. Він сертифікований для роботи у вибухонебезпечних середовищах, що розширює його сферу застосування.

Таким чином, використання SIWAREX WP231 є економічно та технічно виправданим вибором для завдань точного зважування, автоматизації вагових процесів та інтеграції в сучасні системи керування. Його функціональність, гнучкість та надійність роблять цей модуль ефективним рішенням для промислових підприємств різних галузей [12].

2.2 Опис об’єкта керування лабораторного стенду моделі системи автоматизації «маніпулятор-конвеєр»

Було проведено ознайомлення з апаратною частиною та документацією по експлуатації та програмуванню стенду №1 аудиторії №367. Стенд складається з контролера Simatic S7-1200 1215C (рис. 2.2), блоків живлення, комунікаційного модуля, перетворювача частоти (керує швидкістю роботи конвеєра), двигунів конвеєра та маніпулятора, енкодера конвеєра, оптичних датчиків, поворотної вежі-маніпулятора з магнітним захватом, конвеєра, майданчику розташування вантажу та HMI панелі оператора. Робота стенда (за замовчуванням) полягає в піднятті та перенесенні вантажу за допомогою маніпулятору з майданчику для вантажів на конвеєр та перевезення його до кінцевої точки.

Головним компонентом стенду є програмований контролер Siemens Simatic S7-1200 з процесорним модулем 1215C DC/DC/DC, який використовується для керування всіма іншими елементами стенду за допомогою програмного коду.

Даний контролер має ступінь захисту IP20, монтується на 35мм профільну DIN рейку, і може працювати в діапазоні температур від 0 до 60°C. Може використовувати від 2 до 51 аналогових та від 14 до 284 цифрових каналів. До контролера також може виконуватись підключення комунікаційного модуля, сигнальної плати або сигнального модуля вводу-виводу дискретних та аналогових сигналів. Також є Ethernet роз'єм, що можна використовувати як для підключення зовнішніх інтерфейсів користувача (HMI), так і для діагностики або зв'язку з іншими елементами [13].

В даному випадку спільно з контролером використовується блок живлення на 24В – PM 1207 та комутатор для зв'язку між деякими елементами стенду – CSM 1277.

Для підключення зовнішніх датчиків або інших керованих блоків до цифрових входів-виходів та аналогових входів використовуються контакти з гвинтовими затискачами. Для підключення сигнальних та комунікаційних модулів використовуються роз'єми з боків контролера, можна виконати підключення до 8 блоків по черзі.

Рисунок 2.2 – Зовнішній вигляд контролера

Для живлення контролера використовується блок живлення PM 1207 6EP1 332-1SH71. Це імпульсний блок живлення від компанії Siemens, що належить до

серії SITOP. Він призначений для стабільного живлення промислових систем автоматизації. Блок живлення підтримує вхідну напругу в діапазоні 85–264 В змінного струму або 110–300 В постійного струму, а його номінальна напруга становить 120-230 В змінного струму. На виході він забезпечує 24 В постійного струму при максимальному струмі 2.5 А, що відповідає потужності 60 Вт. Пристрій має активний коефіцієнт корекції потужності (PFC) і оснащений захистом від перевантаження, короткого замикання та перегріву. Компактні розміри дозволяють економити місце в шафах управління, а енергоефективність понад 89% робить його екологічно вигідним рішенням. Він легко монтується на DIN-рейку і здатний працювати в широкому діапазоні температур.

Для комунікації з деякими елементами стенду до контролера також додано комутатор – CSM 1277 (рис. 2.3). Це 4-канальний некерований комутатор від компанії Siemens, що належить до серії SIMATIC NET. Він розроблений для інтеграції в мережі промислової автоматизації, забезпечуючи надійне з'єднання між пристроями. Комутатор підтримує стандарт PROFINET та Ethernet, що робить його ідеальним для застосувань у середовищах з підвищеними вимогами до надійності передачі даних.

Пристрій оснащений чотирма портами RJ45 для Ethernet-з'єднань із підтримкою швидкості до 100 Мбіт/с. Некерований характер означає, що він не потребує конфігурації, що спрощує його використання. Комутатор легко монтується на DIN-рейку, має компактний дизайн і низьке енергоспоживання. Захищена конструкція забезпечує стійкість до вібрацій, пилу та перепадів температур, що є важливим для промислових умов. CSM 1277 підтримує функції автоматичного визначення швидкості підключення та діагностики з'єднання, що сприяє швидкому виявленню можливих несправностей у мережі [14].

Рисунок 2.3 – Комутатор CSM1277 6GK7277-1AA10-0AA0

Для керування кроковим двигуном, що приводить у рух конвеєр використовується частотний перетворювач Sinamics G120 (рис. 2.4) з силовим модулем PM240-2. Sinamics G120 – це модульний частотний перетворювач від компанії Siemens, призначений для точного керування швидкістю і моментом двигунів змінного струму. Він широко використовується в промислових системах, таких як конвеєри, насоси, вентилятори та верстати, завдяки своїй гнучкості, високій продуктивності та енергоефективності.

Перетворювач G120 має модульну конструкцію, що складається з трьох основних частин: силового модуля, контрольного модуля та панелі управління (опціонально). Така архітектура дозволяє легко адаптувати пристрій до конкретних завдань і забезпечує простоту обслуговування. Силові модулі доступні для діапазону потужностей від декількох сотень ват до десятків кіловат, що дозволяє використовувати G120 у різних масштабах проектів. У комбінації із силовим модулем PM240-2 – це потужне рішення для керування асинхронними та синхронними двигунами в промислових застосуваннях. Ця конфігурація поєднує гнучкість, енергоефективність і сучасні технології для забезпечення високої продуктивності обладнання.

Силовий модуль PM240-2 підтримує номінальну потужність від 0.37 до 250 кВт, залежно від моделі, та має вбудований функціонал динамічного гальмування, що забезпечує швидке зниження швидкості двигуна в аварійних чи технологічних

ситуаціях. Модуль оснащений покращеними системами захисту від перевантажень, перегріву та короткого замикання, що підвищує надійність роботи у важких умовах. Завдяки ступеню захисту IP20 (з можливістю покращення до IP55 залежно від шафи) модуль підходить для більшості промислових середовищ [15].

Рисунок 2.4 – частотний перетворювач Sinamics G120 з інтелектуальною панеллю оператора IOP-2

Для приведення у рух ланок маніпулятору використовуються 3 драйвери TB6560 V2 (рис. 2.5) та 3 крокові двигуни 28BJ48-12-300-01 (рис. 2.6).

Драйвер TB6560 V2 і кроковий двигун 28BJ48-12-300-01 є популярним вибором для побудови систем із крокovими двигунами в аматорських і професійних проектах, таких як 3D-принтери, ЧПК-станки або робототехніка.

TB6560 V2 – це драйвер крокового двигуна, розроблений для керування двигунами з напругою живлення від 12 до 36 В і максимальною силою струму до 3 А на фазу. Він підтримує мікрошагове керування (від повного кроку до 1/16 кроку), що дозволяє досягти більш плавного і точного руху. Драйвер оснащений захистом від перегріву, перевантаження та короткого замикання, що забезпечує безпеку

експлуатації. Для зручного підключення пристрій має інтерфейси для сигналів STEP і DIR, які легко інтегруються з контролерами, такими як Arduino або STM32.

28BJ48-12-300-01 – це невеликий біполярний кроковий двигун, розрахований на напругу 12 В із номінальним струмом приблизно 300 мА на фазу. Він має стандартну кількість кроків – 48 на оберт (7.5 градусів на крок), що підходить для застосувань із середньою точністю позиціонування. Його компактні розміри роблять його ідеальним для легких механізмів або проектів із обмеженим простором.

Рисунок 2.5 – Драйвер крокового двигуна TB6560 V2

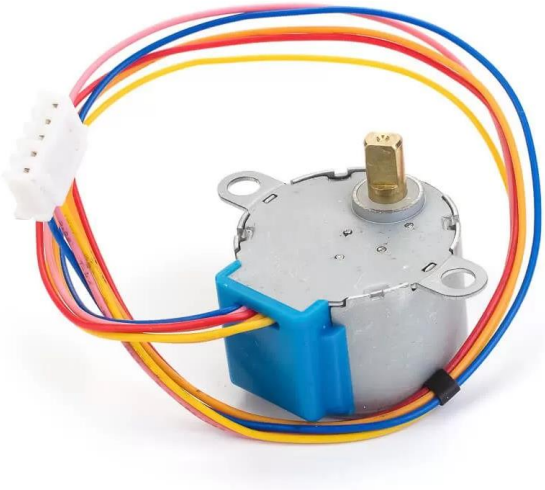


Рисунок 2.6 – Кроковий двигун 28BJ48-12-300-01

Для визначення наявності вантажу на початку та кінці конвеєра встановлено 2 оптичні датчики E3F-DS30C4 (рис. 2.7). В даному випадку використовуються

оптичні датчики бар’єрного типу, тобто датчик складається з випромінювача та відбиваючого світлоповертача.

Оптичний датчик E3F-DS30C4 – це інфрачервоний датчик відбиття, призначений для виявлення об'єктів без фізичного контакту. Він широко використовується в автоматизації промислових процесів, таких як конвеєрні лінії, сортувальні системи та контроль присутності об’єктів.

E3F-DS30C4 має діапазон виявлення до 30 см і працює за принципом відбиття інфрачервоного світла. У пристрої встановлений вбудований інфрачервоний випромінювач і приймач. Датчик активується, коли об’єкт потрапляє в зону виявлення і відбиває інфрачервоне світло назад до приймача. Завдяки цьому датчик підходить для виявлення об’єктів різних кольорів і форм.

Цей датчик має вихідний сигнал типу NPN NO (нормально відкритий), який активується при виявленні об’єкта. Він працює при напрузі живлення 6-36 В постійного струму, що дозволяє використовувати його з широким спектром пристроїв керування. Корпус виготовлений із міцного пластику або металу (залежно від модифікації), що забезпечує його надійність і тривалий термін служби.

Датчик оснащений гвинтом для налаштування чутливості, що дозволяє адаптувати його до різних умов освітлення і типів об'єктів. Завдяки компактному розміру і стандартному кріпленню його легко інтегрувати в існуючі системи.

Рисунок 2.7 – Оптичний датчик E3F-DS30C4 з відбивачем

Для визначення положення конвеєра на його валу встановлено оптичний інкрементальний енкодер 6FX2001-4DB00 (рис. 2.8).

Оптичний інкрементальний енкодер 6FX2001-4DB00 від Siemens є високоточним пристроєм для вимірювання кутового положення, швидкості обертання та напрямку руху. Він використовується у промислових системах автоматизації, наприклад, у приводах, станках ЧПК, роботизованих системах і конвеєрах.

Цей енкодер функціонує за принципом оптичного сканування кодованого диска. На диску нанесено велику кількість рівномірно розташованих штрихів, які оптичний сенсор зчитує у вигляді імпульсів. В результаті пристрій генерує інкрементальні сигнали, що пропорційні обертанню вала. Сигнали передаються через стандартні канали A, B і Z, де:

A і B визначають швидкість і напрямок руху.

Z (Zero Pulse) генерує один імпульс на оберт для визначення початкового положення.

Основні характеристики моделі 6FX2001-4DB00 включають високу роздільну здатність, яка може сягати кількох тисяч імпульсів на оберт, що дозволяє забезпечувати точне керування положенням і швидкістю. Енкодер підтримує стандартні вихідні інтерфейси, що робить його сумісним із більшістю промислових контролерів і приводів.

Пристрій має міцний корпус із високим ступенем захисту (IP64 або вищий,), що забезпечує його роботу в умовах пилу, вібрацій і перепадів температур. Це робить його надійним у складних промислових умовах. Кріплення та підключення є стандартними, що полегшує інтеграцію в існуючі системи.

Рисунок 2.8 – Енкодер 6FX2001-4DB00

Зовнішній вигляд стенду (рис. 2.9):

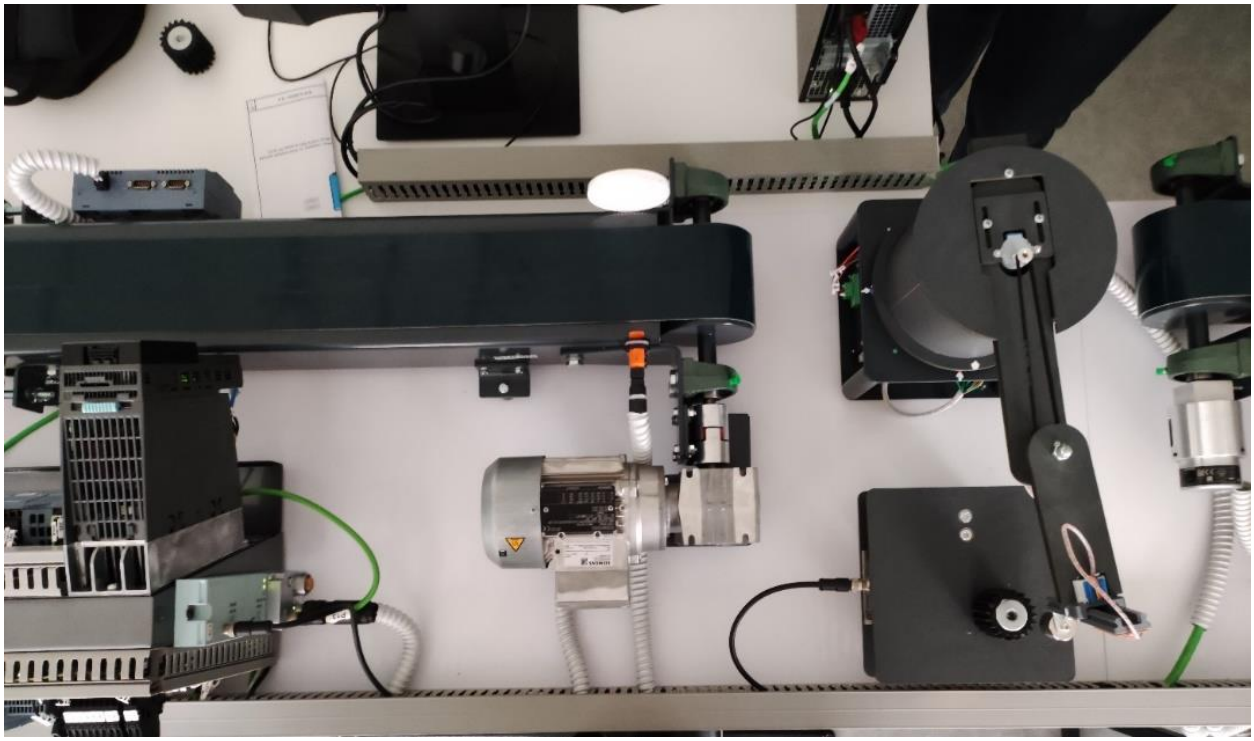


Рисунок 2.9 – Загальний вигляд стенду з усіма його складовими частинами

З використанням програмного пакету Eplan Education 2.9 SP1 виконаємо розмірне креслення параметрів стенду [16]. На рисунках (рис. 2.10 та 2.11) зображено креслення маніпулятору під різними кутами.

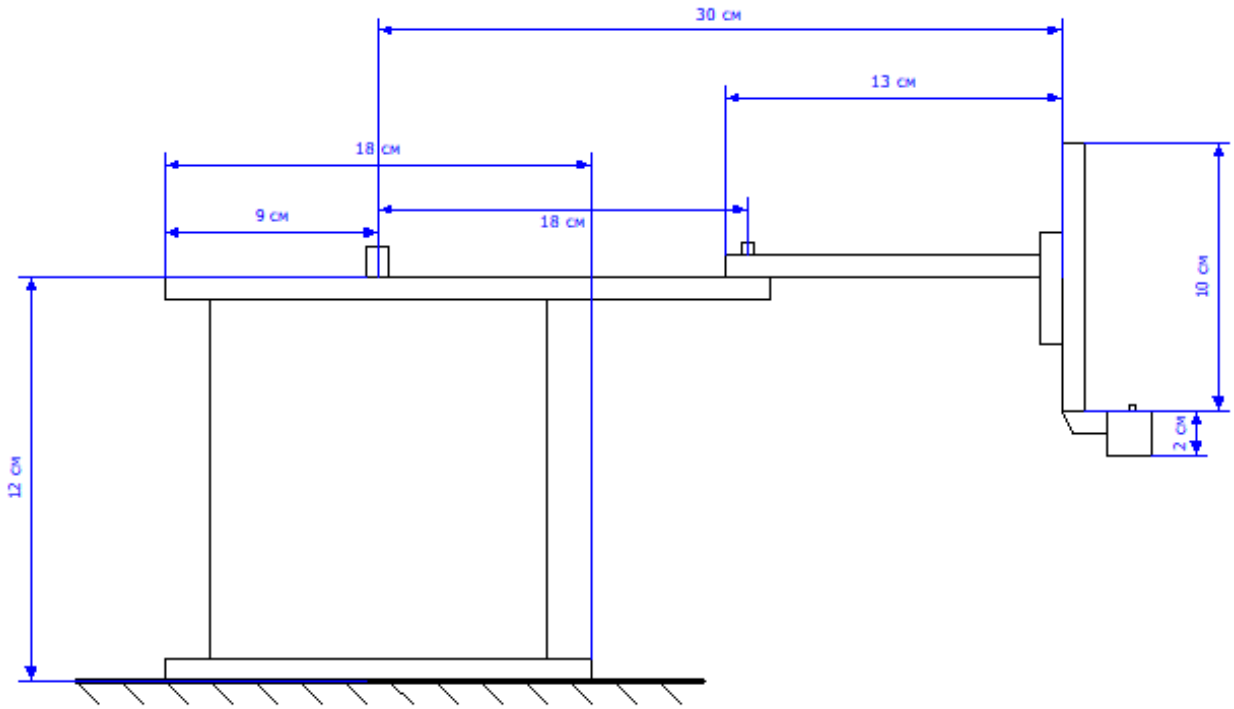


Рисунок 2.10 – Розмірності маніпулятору збоку

Розрахуємо та накреслимо максимально можливі кути повороту маніпулятору (рис. 2.11) які можна використати при проектуванні траєкторій його руху.

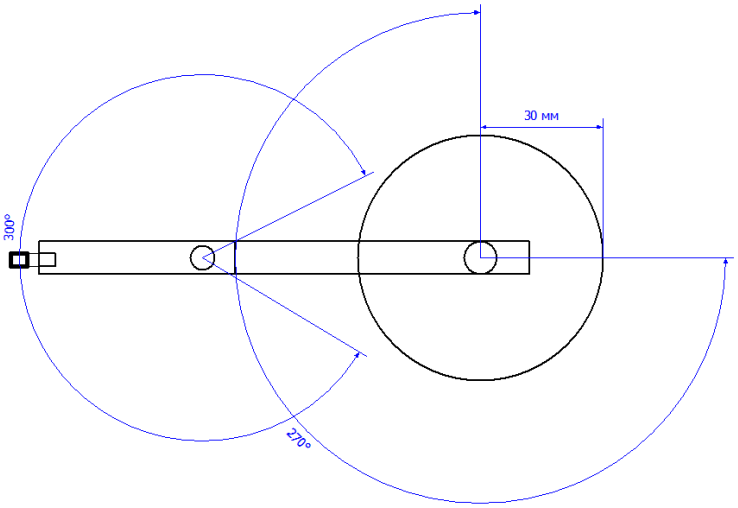


Рисунок 2.11 – Розмірності та максимально допустимі кути руху маніпулятору при вигляді зверху

На наступному рисунку (рис. 2.12) можна побачити реальне зображення маніпулятору.

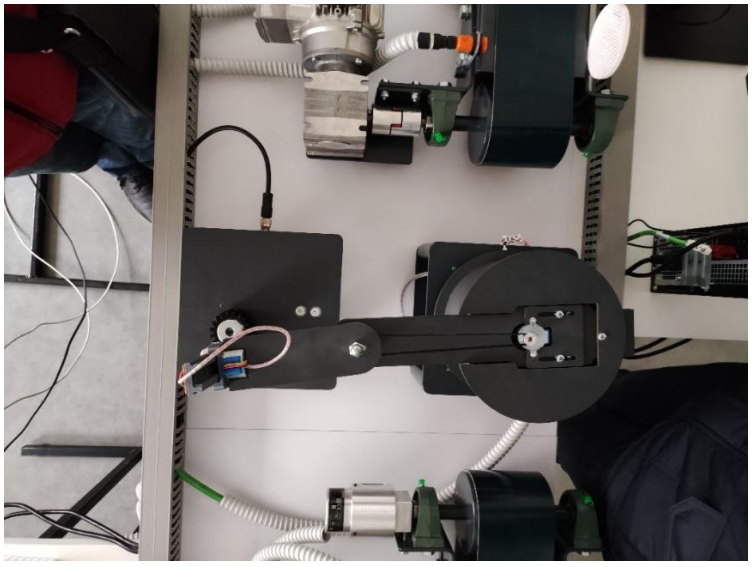


Рисунок 2.12 – Реальне зображення маніпулятору стенду

Вкажемо максимально допустимі границі в яких підйомник маніпулятору може знаходитись від підйомнику до поверхні майданчику для вантажів в двох його крайніх положеннях (рис. 2.13).

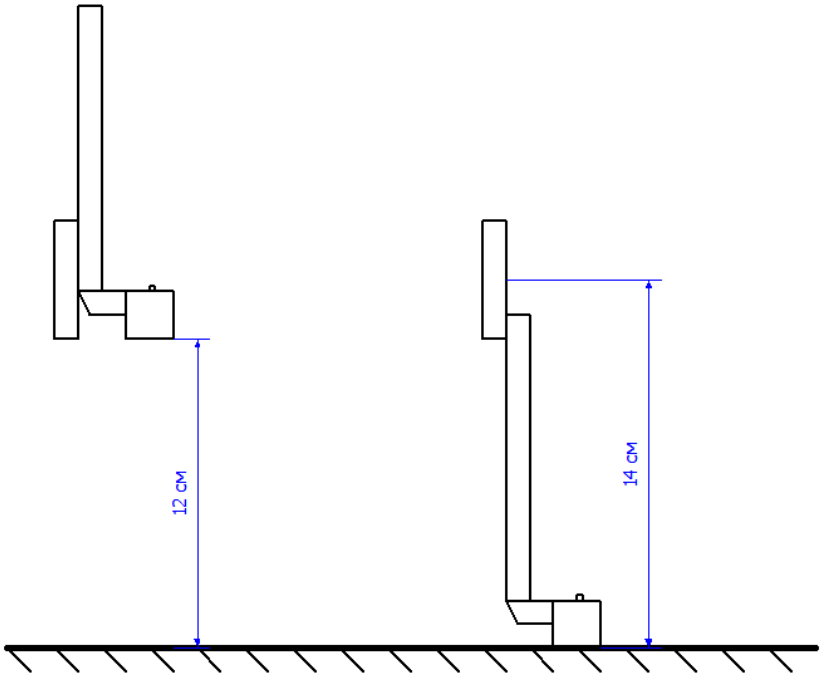


Рисунок 2.13 – Відстані в яких підйомник маніпулятору може знаходитись

Виміряємо розмірні параметри конвеєру та відстані між оптичними датчиками, що розташовані на ньому (рис. 2.14).

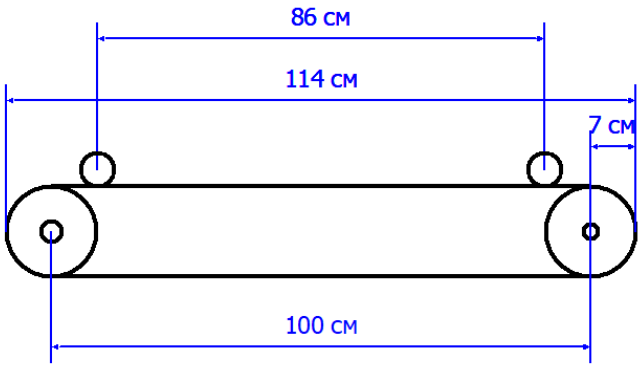


Рисунок 2.14 – Розмірності конвеєра та відстань між оптичними датчиками

Керування стендом здійснюється за допомогою HMI панелі в ручному або автоматичному режимах.

На рисунку (рис. 2.15) наведено інтерфейс HMI панелі оператору для керування стендом.

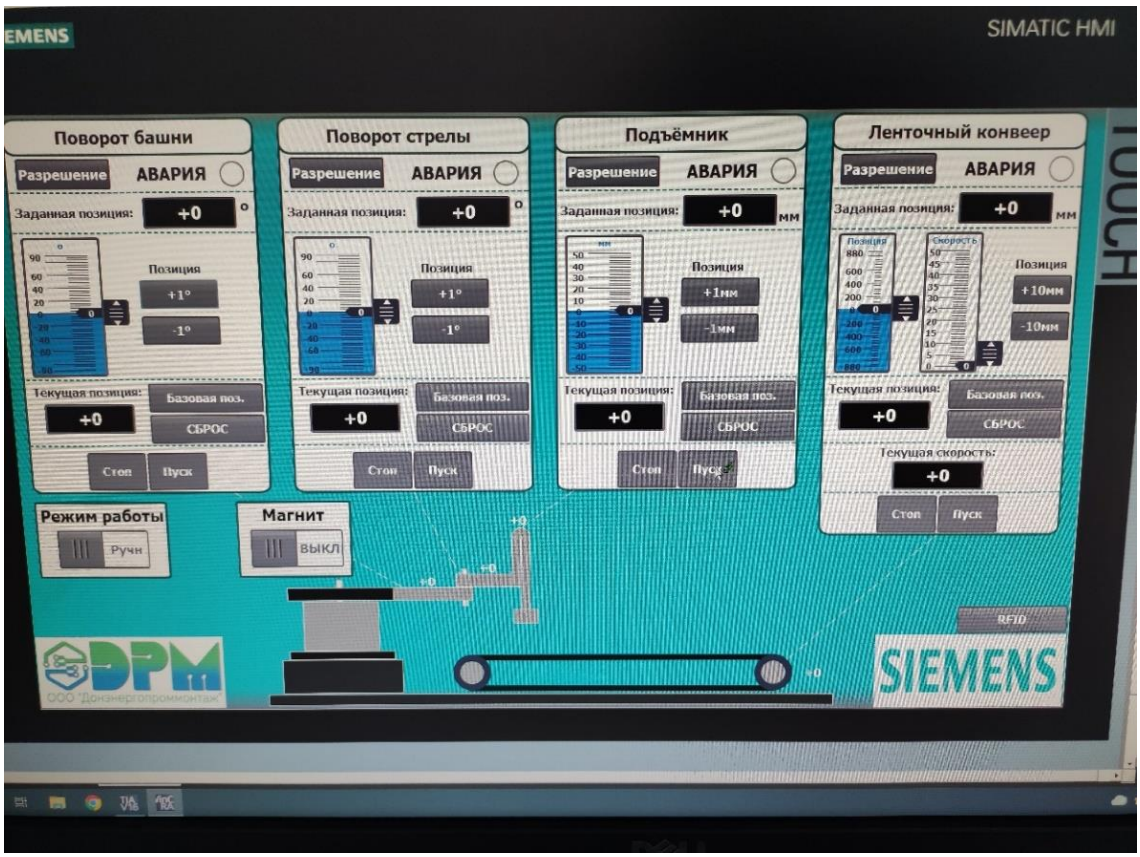


Рисунок 2.15 – Інтерфейс HMI панелі оператору

Для роботи стенду у ручному режимі керування потрібно спочатку надати дозвіл кожній рухомій частині на подальшу роботу шляхом натискання кнопки “Дозвіл”. Після надання необхідних дозволів на роботу можна перейти до встановлення необхідних значень параметрів позицій кожного елементу за допомогою повзунків або для більш точного введення можна використати кнопки “Позиція +1” та “Позиція -1”. Після введення необхідних параметрів необхідно натиснути кнопку “Пуск” після чого маніпулятор або конвеєр встановиться на задану позицію. Якщо виконання задачі потрібно перервати, то потрібно натиснути кнопку “Стоп”. Для захоплення металевого вантажу використовується магніт який вмикається перемикачем “Магніт ВКЛ/ВИКЛ”. В ручному режимі можна реалізувати будь-яку траєкторію руху маніпулятору (з врахуванням фізичних обмежень кутів повороту, що вказані на кресленні рисунку 2.11).

Також в ручному режимі відбувається налаштування базових позицій для подальшої коректної роботи стенду в автоматичному режимі. Це виконується

шляхом ручного встановлення елементів стенду по візуальних мітках в позицію 0, та натискання після цього кнопок “Базовая поз.”

Для роботи стенду в автоматичному режимі необхідно переставити перемикач “Режим работы” зі стану “РУЧН” в стан “АВТО”. Після цього стенд почне автоматично виконувати програму, що було записано до головного контролера S7-1200 1215C. При цьому програма автоматично встановить необхідні дозволи, параметри позицій після чого розпочне рух частин стенду. В автоматичному режимі можуть виконуватись різні траєкторії руху маніпулятору в залежності від поточної програми.

2.3 Математичне забезпечення системи

Описані нижче моделі відображають математичний опис процесів, що автоматизуються у системі з конвеєром, маніпулятором, камерою та датчиком ваги. Розглянемо ці моделі з урахуванням ключових характеристик системи, таких як ідентифікація розмірів деталей, зважування та керування рухом.

Модернізація автоматизованої системи, що включає конвеєр та маніпулятор з додаванням камер і датчиків ваги, вимагає розробки ефективної системи керування, яка здатна забезпечити точність, швидкодію та стабільність роботи всіх компонентів. При синтезі системи керування необхідно враховувати динамічні характеристики системи та визначити критерій оптимальності, за яким оцінюватиметься якість керування.

Для початку розглянемо основні параметри системи, що потребують оптимізації: точність визначення розмірів та ваги деталі, а також взаємодія камери, маніпулятора та контролера. Вибраним критерієм оптимальності є мінімізація часу перехідного процесу та забезпечення мінімальної похибки при позиціонуванні деталей для подальшої обробки маніпулятором.

2.1.1 Модель руху деталей на конвеєрі

Рух деталі по конвеєрній стрічці описується рівняннями кінематики. Для визначення положення деталі у момент часу використовується рівняння (2.1):

$$x(t) = x_0 + v_{conv} * t \quad (2.1)$$

де x(t) – положення деталі на конвеєрі у момент часу t, x0 – початкове положення, a – швидкість руху конвеєра.

Це рівняння дозволяє відстежувати деталі під час їх переміщення по стрічці [17].

2.1.2 Формули для визначення реальних розмірів деталі

Математична модель ідентифікації розмірів деталі за допомогою камери базується на основних принципах комп'ютерного зору та геометричної оптики. Основна мета цієї моделі — точне визначення габаритних розмірів деталі (ширини, висоти, глибини) на основі аналізу зображень, отриманих з камери.

Камера з машинним зором використовує концепцію перспективного перетворення, коли об'єкти, що знаходяться на різній відстані від камери, можуть виглядати меншими або більшими, ніж вони є насправді. Це відображається через відношення реальних розмірів об'єкта до його зображення на сенсорі камери.

Основні параметри, що впливають на вимірювання:

Фокусна відстань камери f — відстань від оптичного центру об'єктива до матриці камери, визначає, наскільки сильно камера "наближає" об'єкт.

Відстань до об'єкта D — відстань між камерою та деталлю на конвеєрі.

Розміри деталі на зображенні P_W і P_H — кількість пікселів, яку займає деталь на зображенні за шириною і висотою.

Фізичні розміри пікселя на матриці камери s_p — фізичний розмір одного пікселя на сенсорі камери.

Реальні розміри об'єкта (ширина W і висота H) пов'язані з параметрами камери наступними співвідношеннями (2.2):

$$W = \frac{P_w * D * s_p}{f}, \qquad H = \frac{P_H * D * s_p}{f} \tag{2.2}$$

де W – реальна ширина деталі, H – реальна висота деталі, P_W – ширина деталі у пікселях на зображенні, P_H – висота деталі у пікселях на зображенні, D – відстань від камери до деталі, S_p – розмір одного пікселя на матриці камери, f – фокусна відстань камери.

Ці рівняння показують [18], що реальні розміри деталі залежать від кількості пікселів, зайнятих деталлю на зображенні, відстані до об'єкта та оптичних параметрів камери.

Похибки у визначенні розмірів деталі можуть виникати через:

1. Роздільну здатність камери. Чим більша роздільна здатність камери, тим точніше можна визначити розміри об'єкта. Похибка вимірювання пов'язана з мінімальним кроком визначення розміру, який залежить від фізичного розміру пікселя на матриці.

2. Неточності калібрування. Якщо камера не калібрована правильно, можуть виникати похибки у визначенні відстаней або розмірів через спотворення зображення.

3. Освітлення. Варіації освітлення можуть вплинути на якість зображення та точність визначення контурів деталі, що призведе до похибок у вимірюваннях.

Загальна математична модель ідентифікації розмірів деталі за допомогою камери включає геометричні перетворення для точного визначення ширини, висоти та глибини об'єкта на основі даних зображень. Для підвищення точності використовуються корекція оптичних спотворень та стереоскопічний метод для оцінки глибини.

2.1.3 Корекція спотворень

У практичних системах виникають оптичні спотворення, зокрема бочкоподібне або подушкоподібне спотворення (рис. 2.16), які виникають через конструктивні особливості об'єктива. Для їх корекції використовуються

калібрувальні матриці, що дозволяють врахувати нерівномірність проекції зображення на сенсор камери [19].

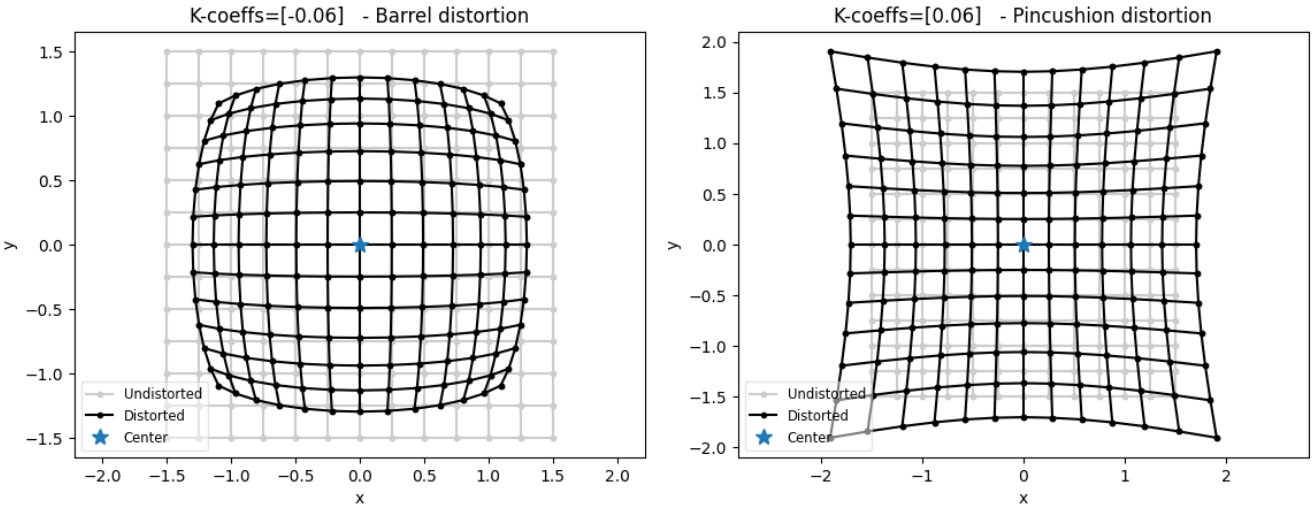


Рисунок 2.16 – Проекція спотворень зображення з камери

Модель спотворення описується нелінійними рівняннями, які можна застосовувати для корекції положення пікселів. Для цього зазвичай використовується наступна формула для радіального спотворення (2.3):

$$r' = r * (1 + k_1 * r^2 + k_2 * r^4 + \ldots) \quad (2.3)$$

де r – відстань від центру зображення до певної точки в ідеальному (неспотвореному) зображенні, r′ – відстань у спотвореному зображенні, k1, k2 – коефіцієнти спотворення, що визначаються під час калібрування камери.

Цей крок необхідний для підвищення точності вимірювання, особливо якщо камера має значні спотворення.

2.1.4 Визначення глибини

Для повної тривимірної ідентифікації деталі, окрім ширини та висоти, важливо визначити також її глибину. Для цього може бути застосовано стереоскопічний зір, коли дві камери використовуються для побудови тривимірної моделі об'єкта. Основний принцип — вимірювання диспаратності (різниці між

зображеннями, отриманими двома камерами), яка дозволяє визначити глибину за формулою (2.4):

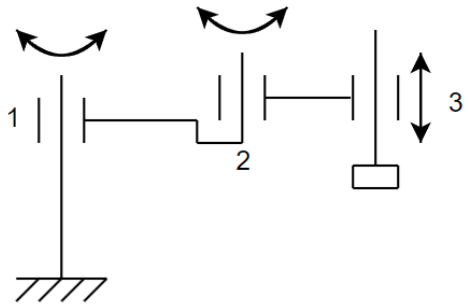
$$Z = \frac{B * f}{d} \tag{2.4}$$

де Z – глибина об'єкта (відстань від камери до деталі), B – базова відстань між двома камерами (основа стереосистеми), f – фокусна відстань камер, d – диспаратність (різниця у положенні однієї й тієї ж точки на зображеннях з двох камер).

Цей метод дозволяє отримувати повну тривимірну модель об'єкта, що є корисним для складних геометричних форм деталей.

2.1.5 Кінематична модель маніпулятора

Створимо математичну модель маніпулятора стенду з трьома степенями свободи. На рис. 2.17 наведено спрощену кінематичну схему маніпулятора.



1 – башта маніпулятора; 2 – стріла маніпулятора; 3 – підйомник маніпулятора

Рисунок 2.17 – Кінематична схема маніпулятора

За даною кінематичною схемою проведемо розрахунки змінних параметрів маніпулятора (табл. 2.1), для чого вирішимо обернену задачу кінематики методом матриць [20].

Задаємо координати кінцевого положення захвату:

X = 60; Y = -170; Z = 40.

Кінематична схема складається з кінематичних пар – обертальних та поступальної.

Таблиця 2.1 – Параметри маніпулятора

№ кінематичної пари	Значення параметрів маніпулятора			
	Θ	S	d	α
1	Θ_1	S_1	d_1	0
2	Θ_2	S_2	$-d_2$	0

З метою вирішення прямої задачі кінематики створимо матриці A_1 та A_2, що визначаються з розширеної матриці A_i і ці системи зв'язні як і-1 та і-та.

Розрахуємо розширені матриці переходу для даної кінематичної системи відповідно до таблиці з параметрами (табл. 1).

$$A_1 = \begin{bmatrix} \cos\Theta_1 & -\sin\Theta_1 * \cos 0 & \sin\Theta_1 * \sin 0 & S_1 * \cos\Theta_1 \\ \sin\Theta_1 & \cos\Theta_1 * \sin 0 & -\cos\Theta_1 * \sin 0 & S_1 * \sin\Theta_1 \\ 0 & \sin 0 & \cos 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\Theta_1 & -\sin\Theta_1 & 0 & S_1\cos\Theta_1 \\ \sin\Theta_1 & \cos\Theta_1 & 0 & S_1\sin\Theta_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.5)$$

$$A_2 = \begin{bmatrix} \cos\Theta_2 & -\sin\Theta_2 * \cos 0 & \sin\Theta_2 * \sin 0 & S_1 * \cos\Theta_2 \\ \sin\Theta_2 & \cos\Theta_2 * \sin 0 & -\cos\Theta_2 * \sin 0 & S_1 * \sin\Theta_2 \\ 0 & \sin 0 & \cos 0 & -d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\Theta_2 & -\sin\Theta_2 & 0 & S_2\cos\Theta_2 \\ \sin\Theta_2 & \cos\Theta_2 & 0 & S_2\sin\Theta_2 \\ 0 & 0 & 1 & -d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.6)$$

Пряма задача кінематики розв'язується шляхом перемноження матриць (2.5) та (2.6). В результаті маючи загальні координати можна визначити елементи матриці T_2 (2.7)і таким чином встановити положення захвату маніпулятора в просторі.

$$T_2 = \begin{bmatrix} \cos\Theta_1 & -\sin\Theta_1 & 0 & S_1\cos\Theta_1 \\ \sin\Theta_1 & \cos\Theta_1 & 0 & S_1\sin\Theta_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \quad (2.7)$$

$$\begin{bmatrix} \cos\Theta_2 & -\sin\Theta_2 & 0 & S_2\cos\Theta_2 \\ \sin\Theta_2 & \cos\Theta_2 & 0 & S_2\sin\Theta_2 \\ 0 & 0 & 1 & -d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos(\Theta_1+\Theta_2) & -\sin(\Theta_1+\Theta_2) & 0 & \cos(\Theta_1+\Theta_2)S_2 + S_1\cos\Theta_1 \\ \sin(\Theta_1+\Theta_2) & \cos(\Theta_1+\Theta_2) & 0 & \sin(\Theta_1+\Theta_2)\,S_2 + S_1\sin\Theta_1 \\ 0 & 0 & 1 & -d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

В результаті координати захвату можна знайти за формулами (2.8 – 2.10).

$$x_2^0 = \cos(\Theta_1+\Theta_2)S_2 + S_1\cos\Theta_1 \quad (2.8)$$

$$y_2^0 = \sin(\Theta_1+\Theta_2)\,S_2 + S_1\sin\Theta_1 \quad (2.9)$$

$$z_2^0 = -d_2 \quad (2.10)$$

Вирішимо зворотну задачу кінематики.

Прирівнявши перші три елементи з четвертого стовпчика результуючої матриці (2.7), до раніше вказаних координат захвату, можна одержати наступну систему:

$$\begin{cases} x_2^0 = \cos(\Theta_1+\Theta_2)S_2 + S_1\cos\Theta_1 \\ y_2^0 = \sin(\Theta_1+\Theta_2)\,S_2 + S_1\sin\Theta_1 \\ z_2^0 = -d_2 \end{cases} \quad (2.11)$$

$$\begin{cases} 60 = \cos(\Theta_1+\Theta_2)S_2 + S_1\cos\Theta_1 \\ -170 = \sin(\Theta_1+\Theta_2)\,S_2 + S_1\sin\Theta_1 \\ 40 = -d_2 \end{cases} \quad (2.12)$$

$$\begin{cases} 60 = (\cos(\Theta_1+\Theta_2) + \cos\Theta_1) * (S_2+S_1) \\ -170 = \sin(\Theta_1+\Theta_2)\,S_2 + S_1\sin\Theta_1 \\ 40 = -d_2 \end{cases} \quad (2.13)$$

Згідно теореми Піфагора визначимо S_1, S_2:

$$(S_1+S_2) = \pm\sqrt{60^2+(-170)^2} = \pm\sqrt{3600+28900} = \pm 180.3 \quad (2.14)$$

$$60 = \cos(\Theta_1 + \Theta_2) + \cos\Theta_1 * 180.3 \quad (2.15)$$

$$\cos(\Theta_1 + \Theta_2) + \cos\Theta_1 = \frac{60}{180.3} = 0.3328 \quad (2.16)$$

$$\cos(\Theta_1 + \Theta_2) + \cos\Theta_1 = \operatorname{acos} 0.3328 = 1.23 \text{ рад} = 70°33'$$

2.1.6 Розрахунки в Matlab

Розрахуємо пряму та зворотну кінематичну систему маніпулятора в програмному середовищі Matlab.

Визначимо довжини ланок маніпулятора (рис. 2.18).

```
>> syms L_1 L_2 theta_1 theta_2 XE YE
>> L1 = 190;
>> L2 = 140;
```

Рисунок 2.18 – Вказання довжин ланок

Позначимо X та Y координати як Θ_1 та Θ_2 (рис. 2.19).

```
>> XE_RHS = L_1*cos(theta_1) + L_2*cos(theta_1+theta_2)

XE_RHS =

L_2*cos(theta_1 + theta_2) + L_1*cos(theta_1)

>> YE_RHS = L_1*sin(theta_1) + L_2*sin(theta_1+theta_2)

YE_RHS =

L_2*sin(theta_1 + theta_2) + L_1*sin(theta_1)
```

Рисунок 2.19 – X та Y координати як Θ_1 та Θ_2

Перетворимо символьні вирази на функції (рис. 2.20).

```
>> XE_MLF = matlabFunction(XE_RHS,'Vars',[L_1 L_2 theta_1 theta_2]);
>> YE_MLF = matlabFunction(YE_RHS,'Vars',[L_1 L_2 theta_1 theta_2]);
```

Рисунок 2.20 – Перетворення виразів на функції

Розрахуємо пряму кінематику. Задаємо кути зони дії ланок маніпулятора та перетворюємо їх з градусів у радіани (рис. 2.21).

```
>> tl_degs_row = linspace(-180,90,100);
>> t2_degs_row = linspace(-150,150,100);
>> [tt1_degs,tt2_degs] = meshgrid(tl_degs_row,t2_degs_row);
>> tt1_rads = deg2rad(tt1_degs);
>> tt2_rads = deg2rad(tt2_degs);
```

Рисунок 2.21 – Введення значень кутів та їх перетворення у радіани

За допомогою функції Matlab розрахуємо X та Y координати (рис. 2.22).

```
>> X_mat = XE_MLF(L1,L2,tt1_rads,tt2_rads);
>> Y_mat = YE_MLF(L1,L2,tt1_rads,tt2_rads);
```

Рисунок 2.22 – Розрахунок X та Y координат

Знайдемо зворотну кінематику з рівнянь прямої кінематики. Задамо рівняння та вирішимо їх за Θ_1 та Θ_2 (рис. 2.23).

```
>> XE_EQ = XE == XE_RHS;
>> YE_EQ = YE == YE_RHS;
>> S = solve([XE_EQ YE_EQ], [theta_1 theta_2])

S =

  struct with fields:

    theta_1: [2×1 sym]
    theta_2: [2×1 sym]
```

Рисунок 2.23 – Вирішення рівнянь за Θ_1 та Θ_2

Виведемо рішення зворотної кінематики за Θ_1 з отриманої вище структури (рис. 2.24).

```
>> simplify(S.theta_1)

ans =

2*atan((2*L_1*YE + (- L_1^4 + 2*L_1^2*L_2^2 + 2*L_1^2*XE^2 + 2*L_1^2*YE^2 -
L_2^4 + 2*L_2^2*XE^2 + 2*L_2^2*YE^2 - XE^4 - 2*XE^2*YE^2 -
YE^4)^(1/2))/(L_1^2 + 2*L_1*XE - L_2^2 + XE^2 + YE^2))

2*atan((2*L_1*YE - (- L_1^4 + 2*L_1^2*L_2^2 + 2*L_1^2*XE^2 + 2*L_1^2*YE^2 -
L_2^4 + 2*L_2^2*XE^2 + 2*L_2^2*YE^2 - XE^4 - 2*XE^2*YE^2 -
YE^4)^(1/2))/(L_1^2 + 2*L_1*XE - L_2^2 + XE^2 + YE^2))
```

Рисунок 2.24 – Рішення за Θ_1

Виведемо рішення зворотної кінематики за Θ_2 з отриманої вище структури (рис. 2.25) [21].

```
>> simplify(S.theta_2)

ans =

-2*atan(((- L_1^2 + 2*L_1*L_2 - L_2^2 + XE^2 + YE^2)*(L_1^2 + 2*L_1*L_2 +
L_2^2 - XE^2 - YE^2))^(1/2)/(- L_1^2 + 2*L_1*L_2 - L_2^2 + XE^2 + YE^2))

 2*atan(((- L_1^2 + 2*L_1*L_2 - L_2^2 + XE^2 + YE^2)*(L_1^2 + 2*L_1*L_2 +
L_2^2 - XE^2 - YE^2))^(1/2)/(- L_1^2 + 2*L_1*L_2 - L_2^2 + XE^2 + YE^2))
```

Рисунок 2.25 – Рішення за Θ_2

2.1.7 Визначення динамічних характеристик системи

Першим етапом синтезу є визначення оптимальних динамічних характеристик системи. Для цього використовується передаточна функція, що описує поведінку системи від вхідного сигналу до вихідного. Конвеєр та маніпулятор з датчиками можна представити у вигляді системи з одним входом та кількома виходами, де вхідним сигналом є команда на зміну швидкості конвеєра, а вихідними — положення деталі, її вага та розміри.

Передаточна функція для конвеєра може бути записана у такій формі (2.17):

$$W(s) = \frac{K}{Ts + 1} \tag{2.17}$$

де W(s) – передаточна функція, K – коефіцієнт підсилення системи, T – стала часу, що характеризує динаміку системи, s – оператор Лапласа.

Ця функція показує, як швидкість зміни положення деталі залежить від вхідного керуючого сигналу. Завданням є мінімізація сталої часу T, щоб забезпечити швидкодію системи.

2.1.8 Оптимізація системи з урахуванням обмежень

На етапі синтезу необхідно враховувати певні обмеження, що накладаються на систему. Наприклад, система керування має враховувати обмежену потужність приводу конвеєра, межі точності вимірювальних приладів (камери та вагів), а також необхідність забезпечення стабільності системи при змінному навантаженні

на конвеєр. З огляду на ці обмеження, обирається критерій оптимальності, який дозволить досягти найбільш ефективної роботи системи.

Одним із таких критеріїв є інтегральний квадратичний критерій, що мінімізує середньоквадратичне відхилення вихідного сигналу від бажаного (2.18) [22]:

$$J = \int_0^\infty (x_{set}(t) - x(t))^2 dt \tag{2.18}$$

де J – функціонал оптимальності,x_{set}(t) – бажане положення деталі, x(t) – фактичне положення деталі на конвеєрі.

Мета полягає у мінімізації цього функціонала, що забезпечить високу точність і мінімальні коливання при зміні швидкості конвеєра або при маніпуляціях із деталями.

2.1.9 Апроксимація оптимальних характеристик

Наступним етапом є апроксимація отриманих оптимальних характеристик для забезпечення їх реалізації в реальних умовах. У системі керування технологічними процесами необхідно знайти баланс між складністю керуючого алгоритму та точністю, яку він забезпечує. Використовуючи пропорційно-інтегрально-диференційний (ПІД) регулятор, можна контролювати поведінку конвеєра і коригувати швидкість його руху, базуючись на відхиленні фактичного положення деталі від заданого.

Передаточна функція ПІД-регулятора для керування швидкістю конвеєра може бути виражена так (2.19):

$$G_{PID}(s) = K_p + \frac{K_i}{s} + K_d s \tag{2.19}$$

де Kp – коефіцієнт пропорційної складової, Ki – коефіцієнт інтегральної складової, Kd – коефіцієнт диференційної складової.

Цей регулятор дозволяє досягти бажаних характеристик системи, зокрема швидкого й точного позиціонування деталей. У процесі апроксимації необхідно

підібрати значення коефіцієнтів Kp, Ki, Kd, щоб система залишалася стабільною і відповідала критеріям якості.

2.4 Аналіз отриманих характеристик

Одним з ключових показників є максимальне динамічне відхилення та залишкове відхилення після стабілізації системи.

Також проведено аналіз квадратичного інтегрального критерію, який характеризує площу під кривою перехідного процесу. Цей показник дозволяє оцінити точність та плавність роботи системи. За результатами аналізу можна скоригувати параметри ПІД-регулятора для досягнення кращих характеристик.

Розроблена система керування для модернізованої автоматизованої виробничої лінії з конвеєром та маніпулятором дозволить забезпечити високу точність і швидкість роботи. За допомогою критерію оптимальності було досягнуто мінімізації перехідних процесів та забезпечено ефективне керування положенням і швидкістю руху деталей на конвеєрі.

Висновки до розділу:

Виконано вибір та дослідження необхідного обладнання для виконання поставленої задачі модернізації.

Було досліджено об’єкт керування – лабораторний стенд, на базі якого буде проводитись модернізація.

Було математично розраховано моделі та отримано формули для визначення розмірів деталей, їх положень, корекції спотворень зображення з камери, визначення глибини зображення.

Також проведено математичні розрахунки прямої та зворотної кінематичних моделей руху ланок маніпулятора з використанням пакету Matlab. Визначено динамічні характеристики системи та оптимальні параметри її роботи.

РОЗДІЛ 3
ПРОГРАМНО-ТЕХНІЧНА РЕАЛІЗАЦІЯ СИСТЕМИ КЕРУВАННЯ ПРОЦЕСОМ

3.1 Розробка схеми мережних інформаційних потоків інтегрованої автоматизованої системи.

Інтегрована автоматизована система побудована на основі кількох ключових модулів, які забезпечують ефективний обмін інформацією та виконання виробничих завдань. Основу системи складає веб-камера, яка використовується для отримання візуальних даних про деталі. Веб-камера є початковим джерелом інформації, вона фіксує зображення деталей у реальному часі та передає ці дані на комп’ютер для подальшої обробки. На комп’ютері функціонує програмне забезпечення, розроблене мовою Python, яке спеціалізується на аналізі зображень. Цей програмний модуль виконує завдання ідентифікації та вимірювання геометричних параметрів деталей, таких як довжина, ширина, діаметр чи інші розміри, що є важливими для виробничих процесів.

Результати роботи Python-програми передаються далі через платформу Node-RED. Node-RED забезпечує маршрутизацію та інтеграцію даних у загальну інформаційну мережу. Ця платформа дозволяє зв’язувати різнорідні джерела даних, трансформувати інформацію в стандартизований формат і передавати її до наступного рівня системи. Передача даних відбувається за допомогою протоколу OPC UA, який забезпечує надійну, безпечну та стандартизовану комунікацію між вузлами системи. Завдяки OPC UA дані з обчислювального модуля стають доступними для TIA Portal, де створено програму управління.

У TIA Portal реалізована програма для промислового контролера (PLC). Контролер є центральним елементом прийняття рішень у системі. Він отримує дані про розміри деталей, оброблені на комп’ютері, а також дані з вагового модуля.

Ваговий модуль підключений до вагів, які здійснюють зважування деталей. Ці дані ваги служать додатковими параметрами, що впливають на рішення контролера. Наприклад, вагові дані можуть використовуватися для класифікації деталей за категоріями або перевірки відповідності заданим критеріям.

Контролер виконує обробку отриманих даних у реальному часі та реалізує алгоритми, які приймають рішення щодо подальшої обробки деталей. Наприклад, на основі розмірів або ваги може виконуватися сортування деталей, їх пакування або перенаправлення до іншого етапу виробничого процесу. Усі ці функції реалізовані через програмні алгоритми, завантажені до контролера з TIA Portal.

Вся система працює як єдиний інформаційний комплекс, де кожен модуль відіграє свою роль. Веб-камера забезпечує початковий збір візуальних даних, Python-програма виконує їхню аналітичну обробку, Node-RED і OPC UA відповідають за інтеграцію та передачу даних, а контролер приймає остаточні рішення на основі отриманої інформації (рис. 3.1). Така інтеграція забезпечує високу точність, надійність і оперативність у виконанні виробничих завдань.

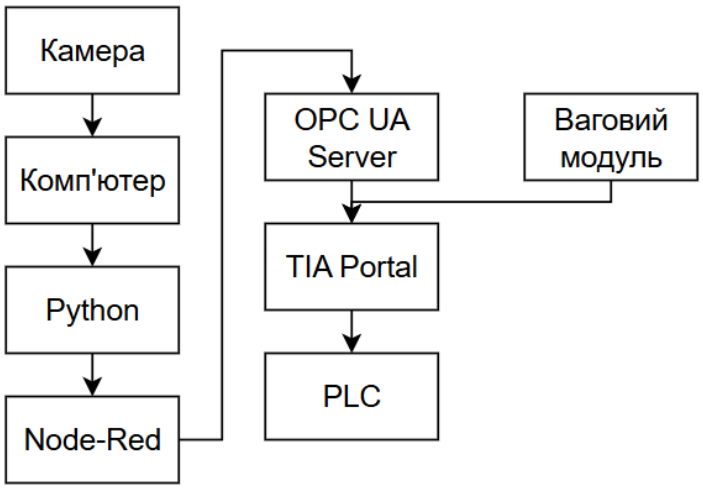


Рисунок 3.1 – Схема мережевих з’єднань

3.2 Структурна схема роботи системи.

Використовуючи середовище креслення діаграм draw.io було створено структурну схему роботи системи (рис. 3.2), яка наочно демонструє всі основні

компоненти, що беруть участь у функціонуванні системи, та їх взаємодію між собою. На схемі відображено, як виконано з’єднання різних елементів системи для досягнення необхідного результату. Систему керування стенду поділено на 4 основні складові – модулі керування, маніпулятор, конвеєр та ваги, а також до нього під’єднано зовнішню систему, що займається візуальним аналізом розмірів об’єктів.

У структуру модулів керування входить контролер (ПЛК), який виконує роль основного елемента керування стендом і забезпечує зв’язок між усіма компонентами системи. Через ПЛК здійснюється передача керуючих сигналів до інших елементів, що дозволяє налаштувати їх функціонування, а також зворотний зв’язок, який надходить від датчиків, таких як енкодер та ультразвуковий далекомір, для обробки отриманої інформації. Таким чином, ПЛК виступає як центральний компонент, який інтегрує всі інші елементи в єдину систему [23].

Для управління окремими елементами системи використовуються додаткові проміжні пристрої. Наприклад, панель оператора підключена до контролера через спеціальний комунікаційний модуль, що дозволяє здійснювати зручне та ефективне введення команд і налаштувань, а ваги підключено з використанням вагового модуля. Кроковий двигун конвеєра, який використовується для переміщення об’єктів, підключений до окремого від контролера модуля перетворювача частоти, що дає можливість точно регулювати його швидкість і напрямок руху.

Окремо варто зазначити, що для керування кроковими двигунами башти, ПЛК підключено безпосередньо до драйверів двигунів, що забезпечує оперативне і точне подавання сигналів. Вони, на відміну від крокового двигуна конвеєра, отримують сигнали від внутрішнього частотного перетворювача, що інтегрований у сам контролер. Це дозволяє підвищити ефективність керування та забезпечити синхронність усіх елементів системи в реальному часі.

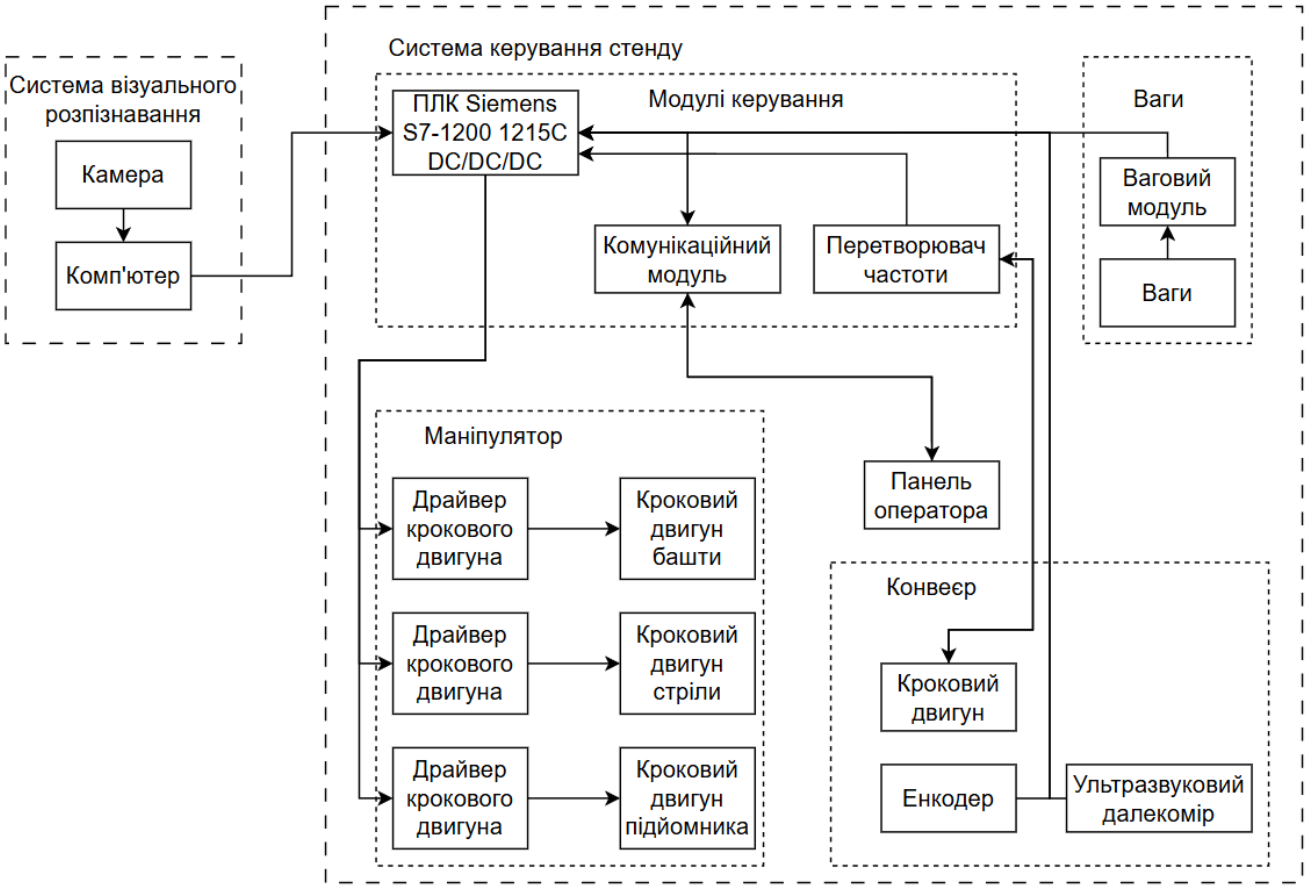


Рисунок 3.2 – Структурна схема

3.3 Блок-схема алгоритму роботи системи

Програма автоматизує процес вимірювання розмірів об'єктів за допомогою камери, яка знімає відео. Спочатку камера фіксує кадри, які обробляються програмним забезпеченням для визначення розмірів об'єкта. Ці дані, що містять розміри, передаються через Node-RED, який служить інтерфейсом між програмним забезпеченням і іншими системами. Використовуючи протокол OPC UA, ці розміри надсилаються до TIA Portal — середовища програмування для автоматизації. У TIA Portal на панелі HMI (Human-Machine Interface) відображаються значення виміряних розмірів [24].

Після запуску HMI панелі в TIA Portal оператор може натискати кнопку "авто-режим", що дозволяє програмі зберігати виміряні розміри і передавати їх в систему для подальшої обробки. Ці дані записуються в пам'ять HMI і зберігаються

там протягом циклу роботи програми контролера. Програма працює в циклі, збираючи нові вимірювання, поки не завершиться робота контролера (рис. 3.3). Після завершення циклу значення розмірів зберігаються в пам'яті або скидаються, залежно від налаштувань програми. Таким чином, програма дозволяє автоматизувати процес вимірювання, обробки даних і їх інтеграцію в систему автоматизації для подальшого використання в виробничих процесах.

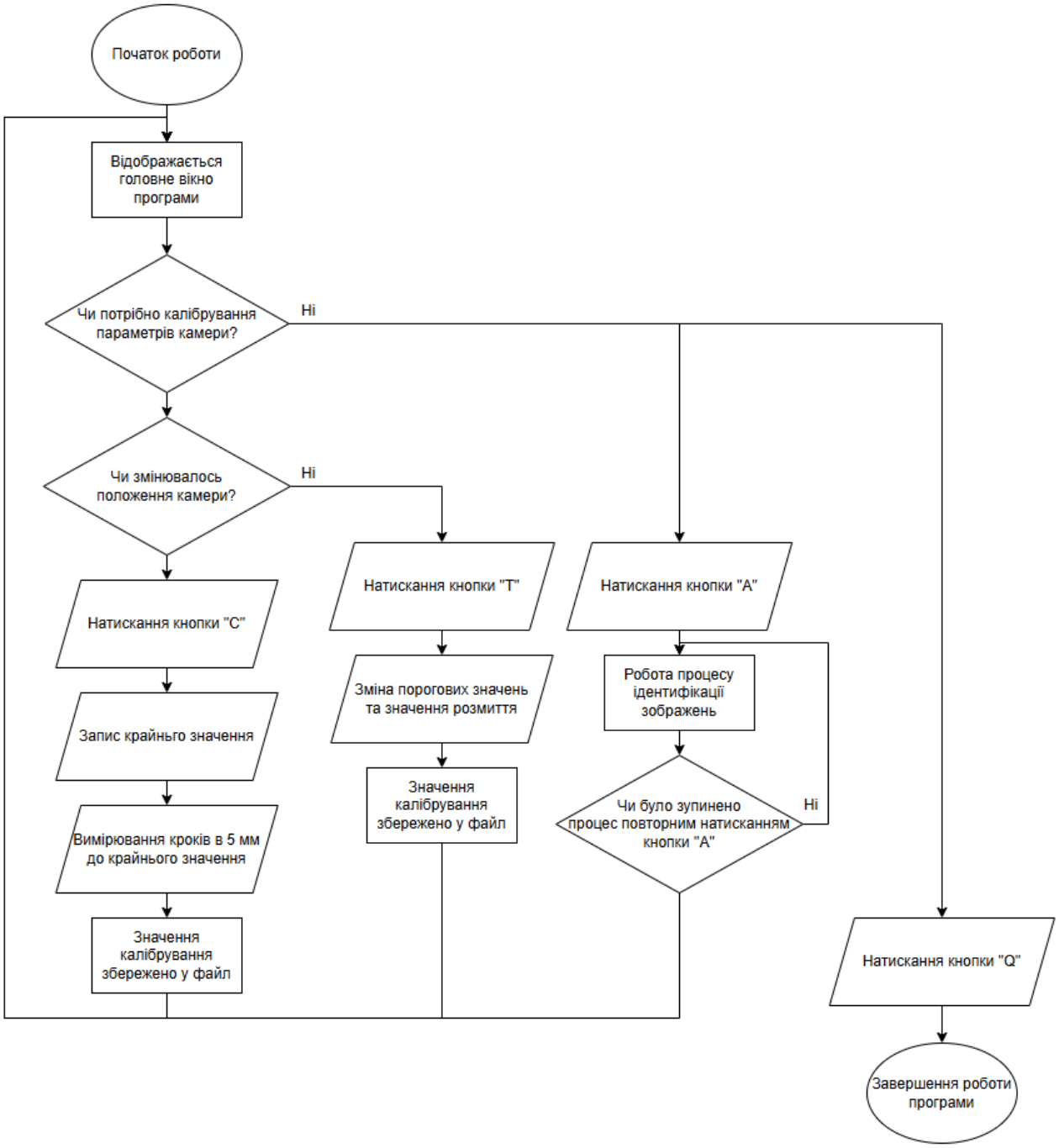


Рисунок 3.3 – Блок-схема алгоритму роботи програми розпізнавання розмірів деталі

3.4 Опис коду програми розпізнавання зображень.

Код програми розпізнавання зображень було написано мовою Python використовуючи для цього середовище програмування Visual code. Цей код є програмою для створення вимірювального інструмента з використанням камери та бібліотек OpenCV. Основна ідея полягає в тому, щоб забезпечити інтерфейс для вимірювання розмірів і відстаней у режимі реального часу, а також для виконання інших операцій, пов'язаних із камерою. Нижче наведено детальний опис різних розділів (рис. 3.4) і функцій програми:

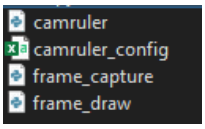


Рисунок 3.4 – Набір файлів програми

На початку головного модуля програми (camruller.py) імпортуються необхідні бібліотеки. Використовуються стандартні бібліотеки Python (os, sys, time, traceback), математична функція hypot, бібліотеки OpenCV (cv2), requests та NumPy. Зазвичай hypot використовують для обчислення відстані між двома точками на площині або для обчислення "модулю" векторів у двовимірному або тривимірному просторі. У контексті комп'ютерного зору або обробки зображень ця функція може бути корисною для обчислення відстаней між точками на зображенні. Також використовуються власні модулі frame_capture та frame_draw, які забезпечують роботу з кадрами та візуалізацію на них.

Програма має попередньо задані налаштування для камери, включаючи її роздільну здатність, частоту кадрів, ідентифікатор та чотирибуквений код стиснення (MJPG). Для автоматичної обробки подій миші визначаються початкові параметри, такі як рівень розмиття, поріг та нормалізація [25].

Програма підтримує завантаження конфігурацій із CSV-файлу (рис. 3.5) (camruler_config.csv). Цей файл дозволяє користувачу легко налаштовувати параметри, які потім читаються й використовуються у програмі.

```
# a config file "camruler_config.csv"
# this is a comma-separated file with one "item,value" pair per line
# use a "=" separated pair like "item=value"
# use # to comment a line
# the items must be named like the default variables above

# read config values
configfile = 'camruler_config.csv'
if os.path.isfile(configfile):
    with open(configfile) as f:
        for line in f:
            line = line.strip()
            if line and line[0] != '#' and (',' in line or '=' in line):
                if ',' in line:
                    item,value = [x.strip() for x in line.split(',',1)]
                elif '=' in line:
                    item,value = [x.strip() for x in line.split('=',1)]
                else:
                    continue
                if item in 'camera_id camera_width camera_height camera_frame_rate
                    try:
                        exec(f'{item}={value}')
                        print('CONFIG:',(item,value))
                    except:
                        print('CONFIG ERROR:',(item,value))
```

Рисунок 3.5 – Читання файлу конфігурації

Наступний блок коду налаштовує камеру. Якщо при запуску програми було вказано аргумент, його використовують як ідентифікатор камери. Встановлюється роздільна здатність, частота кадрів і формат. Камера запускається у вигляді окремого потоку через модуль frame_capture.

Цей код реалізує клас Camera_Thread, який використовує багатопоточність для захоплення та обробки відео з камери. Головна мета цього коду — ефективно отримувати кадри з камери та зберігати їх у буфері, щоб мати можливість працювати з останніми кадрами, не втрачаючи продуктивності. Для цього використовується клас cv2.VideoCapture із бібліотеки OpenCV для налаштування камери, де визначаються такі параметри, як джерело відео, роздільна здатність, частота кадрів і формат відео. Камера ініціалізується, а потім за допомогою цих параметрів налаштовується на потрібні значення [26].

Окремо створюється буфер для кадрів за допомогою черги queue.Queue, що дозволяє зберігати кадри та обробляти їх без втрат (рис. 3.6). У випадку, якщо потрібно зберігати всі кадри, буфер не переповнюється, і в нього додаються нові кадри по мірі їх отримання. Якщо ж буфер налаштовано на збереження лише

останнього кадру, старі кадри замінюються новими, як тільки буфер заповнюється. Це дає змогу підтримувати актуальність кадрів у буфері без надмірного споживання пам'яті.

```
def start(self):

    # buffer
    if self.buffer_all:
        self.buffer = queue.Queue(self.buffer_length)
    else:
        # last frame only
        self.buffer = queue.Queue(1)
```

Рисунок 3.6 – Старт буферу

Клас використовує багатопоточність для того, щоб захоплення кадрів не блокувало основний процес програми. Для цього створюється окремий потік, який займається захопленням кадрів і додає їх у буфер, забезпечуючи тим самим наявність останніх кадрів для подальшої обробки. Потік працює в циклі, періодично перевіряючи, чи є місце в буфері для нових кадрів, і при необхідності чекає, поки буфер не звільниться. Це дозволяє підтримувати стабільну роботу програми навіть при високих вимогах до продуктивності.

Крім того, в коді передбачені функції для старту та зупинки потоку захоплення кадрів. Метод start() ініціалізує камеру і запускає окремий потік для захоплення кадрів, а метод stop() зупиняє цей потік і звільняє ресурси. Функція next() дозволяє отримати кадр з буфера (рис. 3.7), і якщо кадр не доступний, повертається чорний кадр або None (рис. 3.8). Цей підхід забезпечує ефективне оброблення відео в реальному часі та дозволяє працювати з відео без значних затримок чи втрат продуктивності.

```
# load start frame
frame = self.black_frame
if not self.buffer.full():
    self.buffer.put(frame,False)

# status
self.frame_grab_on = True
self.loop_start_time = time.time()
```

Рисунок 3.7 – Завантаження стартового кадру

```
def next(self,black=True,wait=0):

    # black frame default
    if black:
        frame = self.black_frame

    # no frame default
    else:
        frame = None

    # get from buffer (fail if empty)
    try:
        frame = self.buffer.get(timeout=wait)
        self.frames_returned += 1
    except queue.Empty:
        #print('Queue Empty!')
        #print(traceback.format_exc())
        pass

    # done
    return frame
```

Рисунок 3.8 – Повернення чорного кадру

Далі в основному коді створюється об'єкт draw із модуля frame_draw, який відповідає за нанесення графічних елементів на відеокадри.

Модуль frame_draw містить клас DRAW, який реалізує різноманітні функції для малювання на кадрі відео. Його головне завдання — додавання графічних елементів, таких як текст, лінії, прямокутники, кола та хрестики на зображення або відео. Це може бути корисно для візуалізації різних даних або для відображення інтерфейсу користувача, наприклад, при створенні додатків для комп'ютерного зору.

У класі визначено кілька функцій для малювання. Наприклад, метод add_text_top_left дозволяє додавати текст у верхній лівий кут кадру, де текст може бути багаторядковим.

Функція add_text дає можливість розмістити текст у будь-якому місці кадру з урахуванням різних параметрів розташування (наприклад, по центру, праворуч або зверху) (рис. 3.9). Для малювання ліній є методи line, vline і hline, які дозволяють малювати звичайні, вертикальні або горизонтальні лінії відповідно.

```
# add text anywhere
def add_text(self,frame,text,x,y,size=0.8,color='yellow',center=False,middle=False,top=False,right=False):

    color = self.colors.get(color,(0,255,255))

    font = cv2.FONT_HERSHEY_SIMPLEX

    textsize = cv2.getTextSize(text,font,size,1)[0]

    if center:
        x -= textsize[0]/2
    elif right:
        x -= textsize[0]
    if top:
        y += textsize[1]
    elif middle:
        y += textsize[1]/2

    cv2.putText(frame,
                text,
                (int(x),int(y)),
                font,
                size,
                color,
                1, # line width
                cv2.LINE_AA,
                False)
```

Рисунок 3.9 – Метод додавання тексту

Також є функції для малювання прямокутників і кіл, методи rect і circle, які дозволяють задати їхні координати, товщину лінії та колір (рис. 3.10).

```
# line
def line(self,frame,x1,y1,x2,y2,weight=1,color='green'):
    cv2.line(frame,(int(x1),int(y1)),(int(x2),int(y2)),self.colors.get(color,(0,255,0)),weight)


# rectangle
def rect(self,frame,x1,y1,x2,y2,weight=1,color='green',filled=False):
    if filled:
        weight = -1
    cv2.rectangle(frame,(int(x1),int(y1)),(int(x2),int(y2)),self.colors.get(color,(0,255,0)),weight)

# circle
def circle(self,frame,x1,y1,x2,y2,r,weight=1,color='green',filled=False):
    if filled:
        weight = -1
    cv2.circle(frame,(int(x1),int(y1)),(int(x2),int(y2)),int(r),self.colors.get(color,(0,255,0)),weight)
```

Рисунок 3.10 – Методи відображення геометричних фігур

Якщо потрібно, можна зробити ці фігури заповненими, встановивши параметр filled в значення True. Для додавання хрестиків на кадрі існують методи

crosshairs та crosshairs_full, які малюють хрестики в центрі кадру, з можливістю зміщення або інвертування їхнього вигляду (рис. 3.11).

```
# centered crosshairs full frame
def crosshairs_full(self,frame,weight=1,color='green'):
    self.vline(frame,0,weight,color)
    self.hline(frame,0,weight,color)

# centered crosshairs
def crosshairs(self,frame,offset=10,weight=1,color='green',invert=False):
    offset = self.width*offset/200
    xcenter = self.width/2
    ycenter = self.height/2
    if invert:
        self.line(frame,0,ycenter,xcenter-offset,ycenter,weight,color)
        self.line(frame,xcenter+offset,ycenter,self.width,ycenter,weight,color)
        self.line(frame,xcenter,0,xcenter,ycenter-offset,weight,color)
        self.line(frame,xcenter,ycenter+offset,xcenter,self.height,weight,color)
    else:
        self.line(frame,xcenter-offset,ycenter,xcenter+offset,ycenter,weight,color)
        self.line(frame,xcenter,ycenter-offset,xcenter,ycenter+offset,weight,color)
```

Рисунок 3.11 – Методи відображення геометричних фігур

Цей код використовується для роботи з відео або зображеннями, де потрібно візуально відображати інформацію, як-от налаштування для алгоритму обробки зображень, інтерфейсу або індикаторів стану.

Однією з ключових функцій є перетворення пікселів у фізичні одиниці вимірювання (в даному випадку, міліметри). Для цього задається початкова таблиця калібрування, що залежить від розміру кадру. Функція cal_update відповідає за оновлення калібрувальних значень на основі даних, отриманих від користувача (рис. 3.12).

```
# calibration update
def cal_update(x,y,unit_distance):

    # basics
    pixel_distance = hypot(x,y)
    scale = abs(unit_distance/pixel_distance)
    target = baseround(abs(pixel_distance),pixel_base)

    # low-high values in distance
    low  = target*scale - (cal_base/2)
    high = target*scale + (cal_base/2)

    # get low start point in pixels
    start = target
    if unit_distance <= cal_base:
        start = 0
    else:
        while start*scale > low:
            start -= pixel_base
```

Рисунок 3.12 – Частина коду для оновлення даних калібрування

Важливою частиною є обробка подій клавіатури та миші. Залежно від натиснутих клавіш, активуються різні режими: режим конфігурації, автоматичної обробки, нормалізації тощо. Для кожного режиму визначено власну логіку, яка змінює спосіб обробки та відображення відео.

Основний цикл програми безперервно отримує кадри з камери, обробляє їх, додає текстову інформацію та графічні елементи залежно від активного режиму. Наприклад, у режимі автоматичної обробки знаходяться контури на зображенні, оцінюються їхні розміри та площі, і ця інформація виводиться на екран (рис. 3.13).

```
# percent area
percent = 100*w*h/area

# if the contour is too small, ignore it
if percent < auto_percent:
        continue

# if the contour is too large, ignore it
elif percent > 60:
        continue

# convert to center, then distance
x1c,y1c = conv(x1-(cx),y1-(cy))
x2c,y2c = conv(x2-(cx),y2-(cy))
xlen = abs(x1c-x2c)
ylen = abs(y1c-y2c)
alen = 0
if max(xlen,ylen) > 0 and min(xlen,ylen)/max(xlen,ylen) >= 0.95:
    alen = (xlen+ylen)/2
carea = xlen*ylen

# plot
draw.rect(frame0,x1,y1,x2,y2,weight=2,color='red')

# add dimensions
draw.add_text(frame0,f'{xlen:.2f}',x1-((x1-x2)/2),min(y1,y2)-8,center=True,color='red')
draw.add_text(frame0,f'Area: {carea:.2f}',x3,y2+8,center=True,top=True,color='red')
print (f'{xlen:.2f}',f'{ylen:.2f}')
if alen:
    draw.add_text(frame0,f'Avg: {alen:.2f}',x3,y2+34,center=True,top=True,color='green')
if x1 < width-x2:
    draw.add_text(frame0,f'{ylen:.2f}',x2+4,(y1+y2)/2,middle=True,color='red')
else:
    draw.add_text(frame0,f'{ylen:.2f}',x1-4,(y1+y2)/2,middle=True,right=True,color='red')
```

Рисунок 3.13 – Код автоматичного режиму роботи

Режим калібрування дозволяє користувачу вручну визначити масштаби. Користувач позначає точки, а програма обчислює відношення пікселів до реальних одиниць вимірювання, зберігаючи ці дані для подальшого використання.

Код також підтримує обробку клацань миші для вибору областей на екрані, розрахунку їхніх розмірів і нанесення вимірів на зображення.

Візуалізація здійснюється через вікно OpenCV, яке називається CamRuler. У ньому відображаються оброблені кадри з усіма накладеними графічними та текстовими елементами. Користувач може перемикати режими за допомогою клавіш (рис. 3.14), а саме: c для конфігурації, а для автоматичного режиму, або r щоб перевернути зображення на 180 градусів.

```
def key_event(key):

    global key_last
    global key_flags
    global mouse_mark
    global cal_last

    # config mode
    if key == 99:
        if key_flags['config']:
            key_flags['config'] = False
        else:
            key_flags_clear()
            key_flags['config'] = True
            cal_last,mouse_mark = 0,None
```

Рисунок 3.14 – Приклад коду для перемикання режимів роботи програми

Режим конфігурації працює наступним чином:

У програмі, при натисканні клавіші "C" (код клавіші 99), активується режим конфігурації (рис. 3.15). Виконується перевірка поточного стану режиму: Якщо режим конфігурації вже активний (ключ key_flags['config'] встановлений в True), то він буде вимкнений. Якщо режим конфігурації ще не активний, тоді буде ініціалізовано цей режим. Всі інші режими (як автоматичні, нормалізація та інші) вимикаються через функцію key_flags_clear(). Установлюється key_flags['config'] = True для активації конфігураційного режиму. Оновлюється змінні cal_last (що визначає поточну відстань для калібрування) та mouse_mark (запам'ятовує останнє місце кліку миші, якщо воно є) [27]. На екрані з’являється додаткова візуалізація: на екран наносяться червоні лінії, що показують перехрестя та максимальну відстань. Це важливо для зручності виконання калібрування. Якщо переміщення миші в конфігураційному режимі відбулося на область, яка ще не була калібрована, то починається калібрування. Значення виводиться на екран, і програма очікує

нового кліку, щоб продовжити калібрування на наступну точку. Калібрування необхідно продовжувати до моменту доки не буде отримано прийнятну якість розпізнавання об’єкту. Після завершення калібрування, поточні дані налаштування зберігаються у файл camruler_cal.csv для подальшого використання під час наступних запусків програми.

```
# config text data
text.append('')
text.append(f'CONFIG MODE')

# start cal
if not cal_last:
    cal_last = cal_base
    caltext = f'CONFIG: Click on D = {cal_last}'

# continue cal
elif cal_last <= cal_range:
    if mouse_mark:
        cal_update(*mouse_mark,cal_last)
        cal_last += cal_base
    caltext = f'CONFIG: Click on D = {cal_last}'

# done
else:
    key_flags_clear()
    cal_last == None
    with open(calfile,'w') as f:
        data = list(cal.items())
        data.sort()
        for key,value in data:
            f.write(f'd,{key},{value}\n')
        f.close()
    caltext = f'CONFIG: Complete.'

# add caltext
draw.add_text(frame0,caltext,(cx)+100,(cy)+30,color='red')

# clear mouse
mouse_mark = None
```

Рисунок 3.15 – Код режиму калібрування

Загалом, це програма для обробки відеоданих у реальному часі та вимірювання об’єктів на зображенні.

Відправка даних, про розміри деталі, до контролеру з використанням Node-Red виконується за допомогою коду на рис. 3.16. За допомогою бібліотеки requests у Python відправляється POST-запит до локального серверу Node-RED. Спочатку задається змінна url, яка містить адресу вузла, а саме http://localhost:1880/data, де сервер працює на локальній машині через порт 1880 і обробляє запити за шляхом /data. Далі створюється словник data, що містить дані для відправки у форматі

JSON. У цьому випадку словник включає ключі "key" та "key2" зі значеннями, які зберігаються у змінних xlen і ylen. Запит відправляється за допомогою функції requests.post, яка отримує адресу вузла і словник даних, що автоматично конвертується у JSON. Результат виконання запиту зберігається у змінній response, що містить відповідь сервера. Текст цієї відповіді виводиться у консоль за допомогою функції print(response.text). Такий підхід дозволяє інтегрувати Python із Node-RED для передачі даних і отримання результатів обробки.

```
url = "http://localhost:1880/data"  # Node adress Node-RED
data = {"key": xlen, "key2": ylen}  # Data sent
response = requests.post(url, json=data)
print(response.text)
```

Рисунок 3.16 – Код відправки отриманих розмірів до Node-red

В свою чергу в Node-red було створено потік для прийому даних з Python та подальшої їх відправки до контролеру (рис. 3.17). Цей потік Node-RED налаштований для взаємодії з протоколом OPC UA і HTTP, а також обробляє вхідні дані для запису в сервер OPC UA. На початку потоку знаходиться вузол http in, який приймає HTTP-запити методом POST за адресою /data. Дані, отримані з цього запиту, передаються на обробку далі. Наступний вузол — це функція function 6, яка аналізує вхідний об'єкт msg.payload, витягує з нього значення ключів key і key2, а потім формує повідомлення для запису цих значень у змінні OPC UA. Змінна Variable1 отримує значення ключа key, а Variable2 — значення ключа key2.

Далі сформовані дані передаються до вузла OpcUa-Item, який визначає змінну Variable1, що має тип даних Int16. Після цього вузол OpcUa-Client встановлює зв’язок із сервером OPC UA за адресою opc.tcp://192.168.0.5:4840 і виконує запис значень у змінні на сервері. Результати цієї операції передаються до вузла debug 8, який виводить їх у вкладці відладки для перевірки [28].

Паралельно з цим, на початковому етапі потоку, дані також передаються до вузла debug 9, який показує вихідні дані HTTP-запиту, і до вузла http response, що відповідає клієнту після обробки запиту. Таким чином, потік забезпечує інтеграцію між системою, що надсилає HTTP-запити, і сервером OPC UA, дозволяючи

отримувати дані через HTTP і записувати їх у змінні OPC UA з можливістю відладки на кожному етапі.

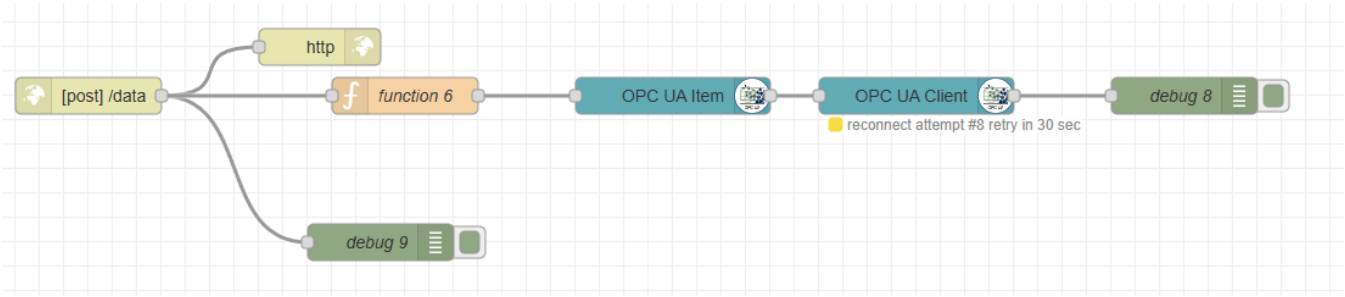


Рисунок 3.17 – Код потоку в Node-red

Після цього значення передається через протокол OPC UA до контролеру в якому наступна програма з рис. 3.18 запам’ятовує значення, на час виконання циклу переміщення вантажу, і виводить його на панель оператора.

```
//перевірка вмикнення нульового кроку
IF NOT "STEP_0_END" THEN
    "STEP_0" := 1;
    "ObjectHeight" := "ObjectHeightSaved";
    "ObjectWidth" := "ObjectWidthSaved";
ELSE
    "STEP_0" := 0;
    RESET_TIMER("IEC_Tim_OFF_0_DB");
END_IF;
```

Рисунок 3.18 – Код програми в TIA Portal

В результаті ці дві змінні виводяться на панель оператора і відображаються там (рис. 3.19) на час роботи стенду.

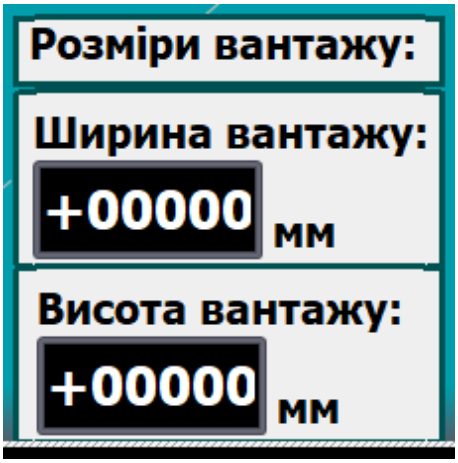


Рисунок 3.19 – Поля для відображення даних на панелі оператора

3.5 Робота користувача з інтерфейсом візуалізації

Після запуску програми в Visual code перед собою можна побачити головне вікно програми – зображення з камери, її параметри, інформація останнього натискання курсору та підказки для кнопок навігації по меню (рис. 3.20).

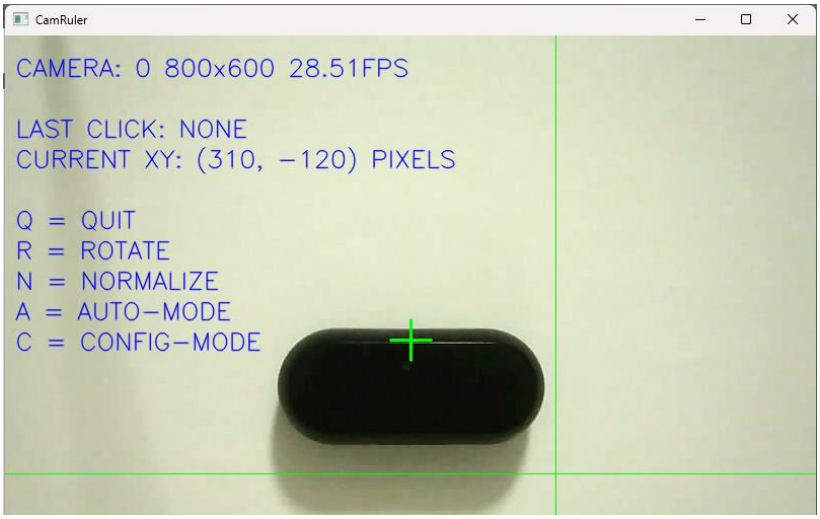


Рисунок 3.20 – Код режиму калібрування

Після натискання кнопки «A» на клавіатурі програма переходить в автоматичний режим роботи, в якому вона намагається ідентифікувати об’єкт у полі зору камери і визначити його розміри (рис. 3.21).

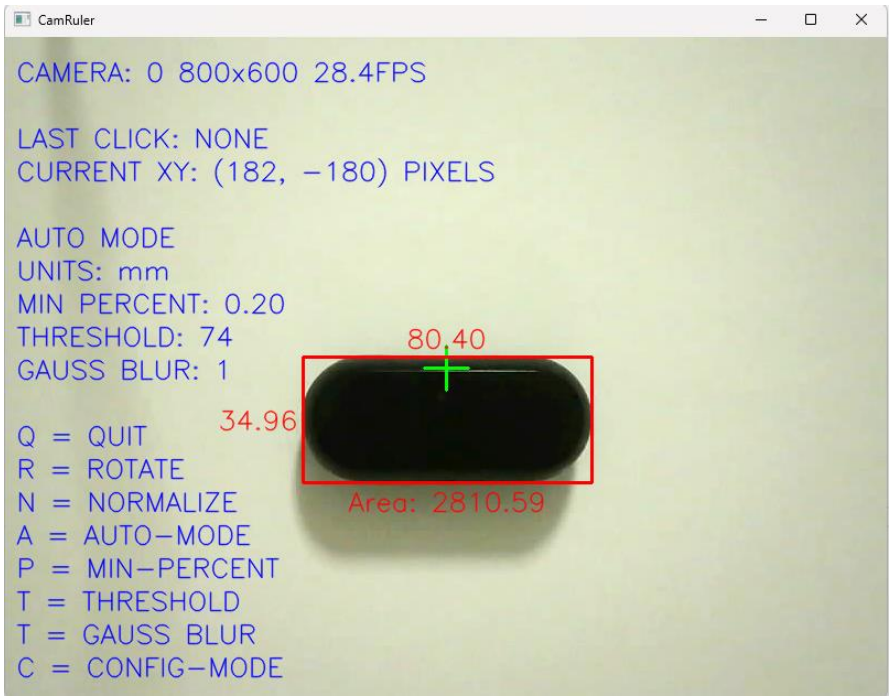


Рисунок 3.21 – Вікно автоматичного режиму роботи

Для коректної роботи можуть знадобитись додаткові налаштування параметрів. Наприклад якщо налаштоване занадто низьке порогове значення того що саме визначається як об’єкт то результат може виглядати як на рис. 3.22, коли окремими об’єктами можуть визначатись навіть дуже малі частини, тіні або частини текстури/рельєфу на об’єкті.. Для того щоб це виправити необхідно натиснути кнопку налаштування порогових значень та гаусового розмиття – кнопка «T». Після чого положення курсору миші в вікні програми починає змінювати значення цих параметрів. Рухи по горизонталі змінюють порогове значення (від меншого зліва, до більшого справа), рухи по вертикалі, в свою чергу, змінюють значення розмиття контурів об’єкта для алгоритму (на екрані розмиття не відображається), при цьому значення змінюються від меншого зверху до більшого знизу.

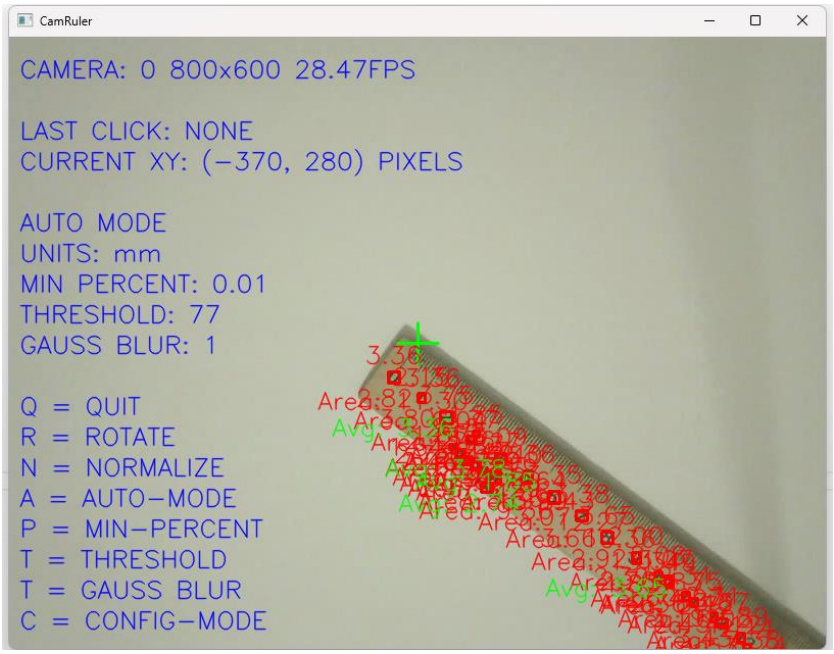


Рисунок 3.22 – Приклад некоректно налаштованих параметрів порогового значення та розмиття

Також, якщо змінювалось положення камери, а саме висота до об’єкту вимірювання, то необхідно провести перекалібровку параметрів для отримання коректних значень розмірів об’єкта. Для цього необхідно натиснути кнопку «C» чим відкриється вікно режиму конфігурації (рис. 3.23). При цьому на екрані з’являється перехрестя. По ньому необхідно вирівняти лінійку і записати з неї

найбільшу довжину в мм до краю зображення. Після цього необхідно перехрестя курсору пересувати вздовж лінійки, від центру до кута, і лівою клавішею миші відмічати крок в 5 мм. Це необхідно для корекції спотворень зображення (особливо якщо камера має більш виражений ефект розтягування зображення на його краях).

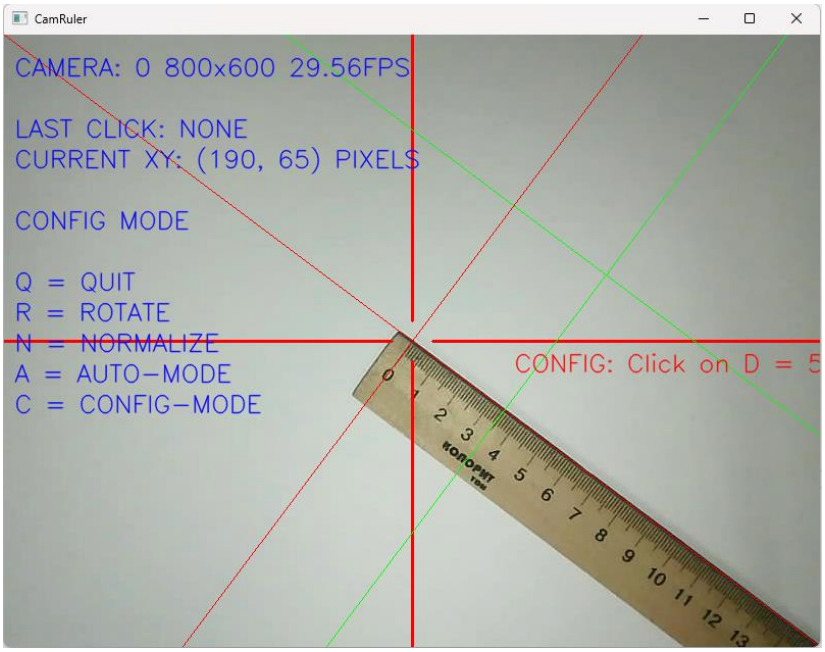


Рисунок 3.23 – Процес калібрування в режимі конфігурації

Для налаштування нормалізації зображення необхідно натиснути кнопку «N» та за аналогією пересуваючи курсор виконати налаштування (лівий верхній кут – найяскравіше зображення, правий нижній – найтемніше) (рис. 3.24).

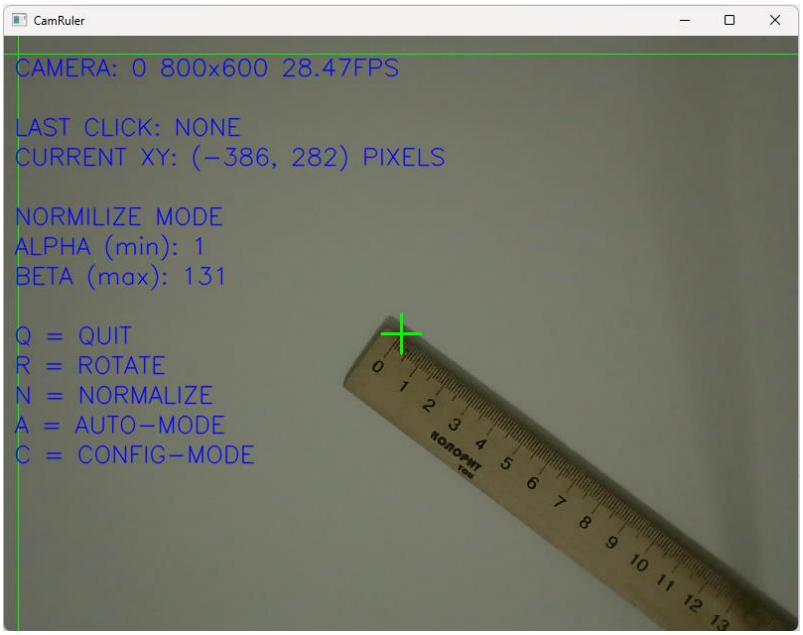


Рисунок 3.24 – Процес нормалізації зображення

Також в TIA Portal було змінено зовнішній вигляд панелі оператора шляхом додавання на неї полів які відображають ширину та висоту вантажу (рис. 3.25), що в даний момент рухається конвеєром, в міліметрах.

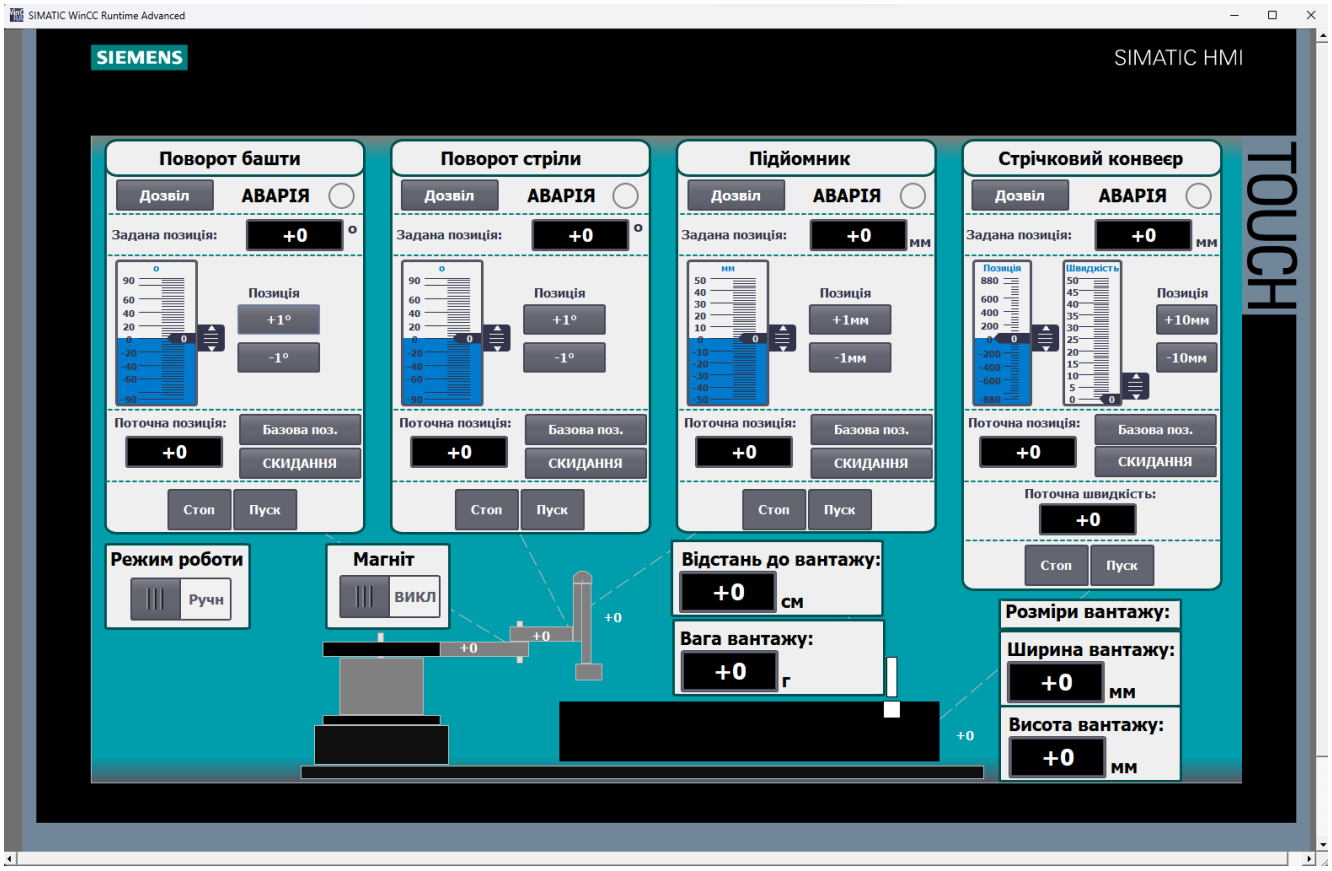


Рисунок 3.25 – Зовнішній вигляд панелі оператора

Для роботи з програмою необхідно перемкнути повзунок «Режим роботи» в режим «авто» і натиснути кнопку, що з’явиться – «Пуск АВТ». Після цього програма автоматично виконає цикл переміщення вантажу, при цьому відображаючи на екрані його вагу та розміри.

Висновки до розділу:

Було розроблено схему мережних інформаційних потоків та структурну схему роботи системи.

Накреслено блок-схему алгоритму роботи системи.

Було виконано написання та опис програми розпізнавання розмірів деталей. Налагоджено передачу результатів її роботи в TIA Portal з використанням програми в Node-red та протоколу OPC UA.

Було також детально описано інтерфейс програми та роботу користувача з ним. Включно з процесом калібрування камери та запуску роботи автоматичного режиму на панелі оператора.

ВИСНОВКИ

В ході виконання даної дипломної роботи було проведено всебічний аналіз технологічних процесів, що використовують системи комп’ютерного зору та зважування в автоматизованих виробничих процесах. Окрему увагу було приділено дослідженню існуючих рішень для інтеграції таких систем, їх переваг та недоліків, що дало змогу визначити доцільність модернізації лабораторного стенду Siemens аудиторії №367 в КНУ. Рішення щодо вдосконалення стенду базувалося на вимогах до точності розпізнавання розмірів і маси деталей на конвеєрі, а також на забезпеченні ефективності автоматизованого керування процесами.

Було сформульовано конкретне технічне завдання для модернізації системи, розроблено структуру комплексу технічних засобів та обрано оптимальне програмне забезпечення для реалізації системи керування. Математичне моделювання руху деталей на конвеєрі, корекція спотворень зображень та визначення глибини й розмірів дозволили точніше визначати параметри деталей, що підлягають обробці, а також забезпечити стабільну роботу системи в цілому.

В процесі математичного моделювання системи було розраховано кінематичні моделі маніпулятора, що забезпечили точність руху захвату. Також було визначено динамічні характеристики системи, що дозволило оптимізувати її роботу з урахуванням обмежень, що виникають у реальних умовах експлуатації. Розрахунки за допомогою Matlab показали, що система здатна забезпечити ефективну та точну роботу при інтеграції комп’ютерного зору і системи зважування.

Розробка схеми мережних інформаційних потоків та структурної схеми роботи системи дало змогу побудувати алгоритм, що забезпечує автоматичне розпізнавання розмірів та маси деталей. Написання та інтеграція програми розпізнавання зображень з TIA Portal, Node-red та протоколом OPC UA дозволили досягти необхідної синхронізації між різними частинами системи та забезпечити зручний інтерфейс для оператора.

Результати виконання цієї роботи є важливим кроком у модернізації лабораторного стенду, що дозволяє реалізувати сучасні методи автоматизації на базі універсальних стандартів промислових систем.

Подальше вдосконалення цієї системи може полягати в:

- Впровадження більш складних методів комп’ютерного зору, таких як машинне навчання та нейронні мережі, може дозволити системі не лише розпізнавати розміри та масу деталей, але й ідентифікувати дефекти на поверхні виробів або визначати інші характеристики, важливі для виробничого процесу.
- Підвищення швидкості обробки даних, що забезпечить можливість безперервної роботи з розпізнаванням деталей без зупинки конвеєра, а також в удосконаленні системи зважування для підвищення точності визначення маси вантажів.
- Оновлення механічної частини маніпулятора для підвищення його швидкості та точності роботи дозволить безперервно працювати з конвеєром, не знижуючи швидкість його руху. Це також може включати використання більш потужних і швидких крокових двигунів або іншого типу приводів.

Також результатом виконання даної роботи є створення нових методичок для виконання лабораторних робіт, з використанням доданого обладнання, на стенді аудиторії Siemens №367.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Концепція стрічки конвеєра, з виробництва деталей з подальшим розпізнаванням і сортуванням деталей URL: ПМАП_2020_final_compressed.pdf (дата звернення 27.08.2024)

2. IOT SIZE BASED PRODUCT SORTING MACHINE URL: https://www.researchgate.net/publication/374559795_IOT_SIZE_BASED_PRODUCT_SORTING_MACHINE (дата звернення 29.08.2024)

3. DESIGN AND FABRICATION OF COLOR SORTER MACHINE USING ARDUINO NANO URL: https://www.researchgate.net/publication/360139505_DESIGN_AND_FABRICATION_OF_COLOR_SORTER_MACHINE_USING_ARDUINO_NANO (дата звернення 30.08.2024)

4. Design and Fabrication of Automatic Weight, Color and Height Based Sorting System URL: https://www.researchgate.net/publication/374065055_Design_and_Fabrication_of_Automatic_Weight_Color_and_Height_Based_Sorting_System (дата звернення 03.09.2024)

5. Sensor-based sorting URL: https://www.researchgate.net/publication/374909542_ Sensor-based_sorting (дата звернення 08.09.2024)

6. Weight based Object Shorting Automation System Using IoT based Application URL: https://www.researchgate.net/publication/367614458_Weight_based_Object_Shorting_Automation_System_Using_IoT_based_Application (дата звернення 16.09.2024)

7. Siemens 6ES7215-1AG40-0XB0 Datasheet: URL: https://media.automation24.com/datasheet/en/6ES72151AG400XB0.pdf (Дата звернення 21.09. 2024)

8. Siemens 6EP1332-1SH71 Datasheet: URL: https://docs.rs-online.com/b91a/0900766b8117fb77.pdf (Дата звернення 28.09. 2024)

9. Siemens 6GK7277-1AA10-0AA0 Datasheet: URL: https://octopart.com/datasheet/6gk7277-1aa10-0aa0-siemens-22651728 (Дата звернення 02.10. 2024)

10. Siemens 6SL3210-1PB13-0UL0 Datasheet: URL: https://pnap.ir/wp-content/uploads/product_pdf/siemens/g120/6SL3210-1PB13-0UL0.pdf (Дата звернення 08.10. 2024)

11. TB6560 Datasheet: URL: https://www.alldatasheet.com/view.jsp?Searchword=Tb6560%20datasheet&gad=1&gclid=EAIaIQobChMIuMiAvJiq_wIVQzcYCh2KogD7EAAYASAAEgJ2pPD_BwE (Дата звернення 14.10. 2024)

12. 28BYJ48-12-300-01 Datasheet: URL: https://datasheetspdf.com/datasheet/28BYJ48-12-300-01.html (Дата звернення 19.10. 2024)

13. E3F-DS30C4 Datasheet: URL: https://www.epitran.it/ebayDrive/datasheet/SERIE_E3F.pdf (Дата звернення 25.10. 2024)

14. Encoder systems: URL: https://www.sew-eurodrive-partner.com/products/motors/servomotors/accessories_and_options/encoders/encoders.html (Дата звернення 26.10.2024)

15. The Use of Image Processing and Sensor in Tomato Sorting Machine by Color, Size, and Weight: https://www.researchgate.net/publication/361952613_The_Use_of_Image_Processing_and_Sensor_in_Tomato_Sorting_Machine_by_Color_Size_and_Weight (дата звернення 03.11.2024)

16. Flow dynamics and size-based sorting of bidispersed colloidal particles in evaporating sessile water droplets: substrate heating and wettability effects URL: https://www.researchgate.net/publication/377592026_Flow_dynamics_and_size-based_sorting_of_bidispersed_colloidal_particles_in_evaporating_sessile_water_droplets_substrate_heating_and_wettability_effects (дата звернення 07.11.2024)

17. Костюк В. И., Гавриш А. П., Ямпольский Л. С., Карлов А. Г. промислові роботи: Конструювання, управління, експлуатація. К.: Вища школа, 1985.

18. Цвіркун Л. І., Грулер Г. Робототехніка та мехатроніка: навч. посіб. /Нац. гірн. ун-т. 3-тє вид., перероб.. і доповн./ Д.: НГУ, 2017. 224 с.

19. Modeling and Control of 2-DOF Robot Arm: URL: https://www.ijeert.org/papers/v6-i11/3.pdf (Дата звернення 08.11. 2024)

20. Derive and Apply Inverse Kinematics to Two-Link Robot Arm: URL: https://www.mathworks.com/help/symbolic/derive-and-apply-inverse-kinematics-to-robot-arm.html (Дата звернення 12.11. 2024)

21. Зайцев Г. Ф., Стеклов В. К., Бріцький О. І. Теорія автоматичного управління. К.: Техніка, 2002. 688 с.

22. Application Research of Vision Sensor in Material Sorting Automation Control System URL: https://www.researchgate.net/publication/340657413_Application_Research_of_Vision_Sensor_in_Material_Sorting_Automation_Control_System (Дата звернення 16.11.2024)

23. Construction and Operation Analysis of Logistics Sorting Simulation Experimental Platform URL: https://www.researchgate.net/publication/373227715_Construction_and_Operation_Analysis_of_Logistics_Sorting_Simulation_Experimental_Platform (Дата звернення 18.11.2024)

24. A Size Filter Regulates Apical Protein Sorting URL: https://www.researchgate.net/publication/372929818_A_Size_Filter_Regulates_Apical_Protein_Sorting (Дата звернення 19.11.2024)

25. Object dimensioning system and method URL: https://patents.google.com/patent/US10937183B2/en (дата звернення 21.11.2024)

26. Method and system to determine distance to an object in an image URL: https://patents.justia.com/patent/10217240 (Дата звернення 24.11.2024)

27. Systems and methods for object detection including pose and size estimation URL: https://patent.nweon.com/31418 (Дата звернення 25.11.2024)

28. Computer Vision and Image Characteristic Search URL: https://patents.google.com/patent/US20190205962A1/en (Дата звернення 28.11.2024)

29. Weighing system using a conveyor belt with load cells URL: https://patents.google.com/patent/US20140216894A1/en (Дата звернення 28.11.2024)

30. Маринич І. А., Тронь В. В. Методичні рекомендації до виконання кваліфікаційної роботи магістра для студентів спеціальності 151 “Автоматизація та

комп’ютерно-інтегровані технології”. Кривий Ріг : Видавничий центр КНУ, 2022. 50с.

31. ДСТУ 3008:2015. Звіти у сфері науки і техніки. Структура і правила оформлення. Київ, ДП «УкрННЦ», 2015. 26с. (Інформація та документація).

32. ДСТУ 8302:2015. Бібліографічне посилання. Загальні вимоги та правила складання Київ, ДП «УкрННЦ», 2016. 16 с. (Інформація та документація).

33. ДСТУ 3582:2013. Бібліографічний опис. Скорочення слів і словосполучень в українській мові.

34. Загальні вимоги та правила. Київ, ДП «УкрННЦ», 2013. 23 с. (Інформація та документація)

35. ДСТУ 3651.0-97 Метрологія. Одиниці фізичних величин. Основні одиниці фізичних величин Міжнародної системи одиниць. Основні положення, назви та позначення Київ, Держстандарт України, 1998. 27 с. (Інформація та документація).

ДОДАТОК А

Кодування програми в TIA Portal v17

```
IF "Switch_AUT/MAN" AND "Start_AUT" AND NOT "Stop_AUT" AND NOT
"Conveyor".StatusBits.Error AND NOT "Axis_1".StatusBits.Error
   AND NOT "Axis_2".StatusBits.Error AND NOT "Axis_3".StatusBits.Error
THEN
   "AUTO_mode_work" := 1;
ELSE
   "AUTO_mode_work" := 0;
   "Start_AUT" := 0;
END_IF;

IF NOT "AUTO_mode_work" AND "Switch_AUT/MAN" THEN
   "Data_block_1"."stop D1" := 1;
   "Data_block_1"."stop D2" := 1;
   "Data_block_1"."stop D3" := 1;
   "Data_block_1"."conv stop" := 1;
END_IF;

IF "AUTO_mode_work" THEN

   IF NOT "STEP_0_END" THEN
      "SavedHeight" := "Height";
      "SavedWidth" := "Width";
      "SavedWeight" := "weight" / 3;
         IF "SavedHeight" >= 40 AND "SavedHeight" <= 60 AND "SavedWidth" >= 40
AND "SavedWidth" <= 60 THEN
            #WeightType := 'Detail A';
            ELSE
```

```
            IF "SavedHeight" >= 40 AND "SavedHeight" <= 60 AND "SavedWidth" >=
40 AND "SavedWidth" <= 60 THEN
                #WeightType := 'Detail B';
            ELSE
                #WeightType := 'Unknown';
        END_IF;
    "STEP_0" := 1;

    ELSE
        "STEP_0" := 0;
    END_IF;



    IF "STEP_0" THEN
        "Data_block_1"."On/off D1" := 1;
        "Data_block_1"."On/off D2" := 1;
        "Data_block_1"."On/off D3" := 1;
        "Data_block_1"."stop D1" := 0;
        "Data_block_1"."stop D2" := 0;
        "Data_block_1"."stop D3" := 0;
        "Data_block_1"."home D1" := 1;
        "Data_block_1"."home D2" := 1;
        "Data_block_1"."home D3" := 1;
        #Start_Timer_0 := 1;

        "IEC_Tim_OFF_0_DB".TON(IN := #Start_Timer_0,
                               PT := T#1s,
                               ET => #ET_TIME_ON_0,
                               Q => #Timer_0_END);
```

```
    IF #Timer_0_END THEN
        "Data_block_1"."home D1" := 0;
        "Data_block_1"."home D2" := 0;
        "Data_block_1"."home D3" := 0;
        "STEP_0_END" := 1;
        "STEP_1" := 1;
    END_IF;
END_IF;

IF "STEP_1" THEN
    "Data_block_1"."angle D3" := -45;
    "Data_block_1"."move D3" := 1;
    #Start_Timer_1 := 1;

    "IEC_Tim_OFF_1_DB".TON(IN := #Start_Timer_1,
                           PT := T#12s,
                           ET => #ET_TIME_ON_1,
                           Q => #Timer_1_END);

    IF #Timer_1_END THEN
        "Data_block_1"."move D3" := 0;
        "magnit" := 1;
        #Start_Timer_2 := 1;
    END_IF;

    "IEC_Tim_OFF_2_DB".TON(IN := #Start_Timer_2,
                           PT := T#1s,
                           ET => #ET_TIME_ON_2,
                           Q => #Timer_2_END);
```

```
    IF #Timer_2_END THEN
        "Data_block_1"."angle D3" := 40;
        "Data_block_1"."move D3" := 1;
        #Start_Timer_3 := 1;
    END_IF;

    "IEC_Tim_OFF_3_DB".TON(IN := #Start_Timer_3,
                           PT := T#20s,
                           ET => #ET_TIME_ON_3,
                           Q => #Timer_3_END);

    IF #Timer_3_END THEN
        "Data_block_1"."move D3" := 0;
        "STEP_1" := 0;
        "STEP_2" := 1;
    END_IF;
END_IF;


IF "STEP_2" THEN
    "Data_block_1"."angle D2" := -90;
    "Data_block_1"."move D2" := 1;
    #Start_Timer_4 := 1;

    "IEC_Tim_OFF_4_DB".TON(IN := #Start_Timer_4,
                           PT := T#10s,
                           ET => #ET_TIME_ON_4,
                           Q => #Timer_4_END);

    IF #Timer_4_END THEN
```

```
        "Data_block_1"."move D2" := 0;
        "STEP_2" := 0;
        "STEP_3" := 1;
    END_IF;
END_IF;


IF "STEP_3" THEN
    "Data_block_1"."angle D1" := 90;
    "Data_block_1"."move D1" := 1;
    #Start_Timer_5 := 1;

    "IEC_Tim_OFF_5_DB".TON(IN := #Start_Timer_5,
                           PT := T#10s,
                           ET => #ET_TIME_ON_5,
                           Q => #Timer_5_END);

    IF #Timer_5_END THEN
        "Data_block_1"."move D1" := 0;
        "STEP_3" := 0;
        "STEP_4" := 1;
    END_IF;
END_IF;


IF "STEP_4" THEN
    "Data_block_1"."angle D2" := 0;
    "Data_block_1"."move D2" := 1;
    #Start_Timer_6 := 1;
```

```
    "IEC_Tim_OFF_6_DB".TON(IN := #Start_Timer_6,
                           PT := T#10s,
                           ET => #ET_TIME_ON_6,
                           Q => #Timer_6_END);

    IF #Timer_6_END THEN
        "Data_block_1"."move D2" := 0;
        "STEP_4" := 0;
        "STEP_5" := 1;
    END_IF;
END_IF;

IF "STEP_5" THEN
    "Data_block_1"."angle D3" := -15;
    "Data_block_1"."move D3" := 1;
    #Start_Timer_7 := 1;

    "IEC_Tim_OFF_7_DB".TON(IN := #Start_Timer_7,
                           PT := T#15s,
                           ET => #ET_TIME_ON_7,
                           Q => #Timer_7_END);

    IF #Timer_7_END THEN
        "Data_block_1"."move D3" := 0;
        "magnit" := 0;
        #Start_Timer_8 := 1;
    END_IF;

    "IEC_Tim_OFF_8_DB".TON(IN := #Start_Timer_8,
                           PT := T#1s,
```

```
                    ET => #ET_TIME_ON_8,
                    Q => #Timer_8_END);

    IF #Timer_8_END THEN
        "Data_block_1"."angle D3" := 40;
        "Data_block_1"."move D3" := 1;
        "Data_block_1"."conv ON/OFF" := 1;
        "Data_block_1"."conv stop" := 0;
        #Start_Timer_9 := 1;
    END_IF;

    "IEC_Tim_OFF_9_DB".TON(IN := #Start_Timer_9,
                    PT := T#15s,
                    ET => #ET_TIME_ON_9,
                    Q => #Timer_9_END);

    IF #Timer_9_END THEN
        "Data_block_1"."move D3" := 0;
        "STEP_5" := 0;
        "STEP_6" := 1;
    END_IF;
END_IF;


IF "STEP_6" THEN
    "Data_block_1"."conv home" := 1;

    IF "RightSensor" THEN
        "Data_block_1"."conv position" := 820;
        "Data_block_1"."conv speed" := 50;
```

```
        "Data_block_1"."conv move" := 1;
        "Data_block_1"."angle D2" := 90;
        "Data_block_1"."move D2" := 1;
        "Start_Timer_10" := 1;
    END_IF;

    "IEC_Tim_OFF_10_DB".TON(IN := "Start_Timer_10",
                            PT := T#20s,
                            ET => #ET_TIME_ON_10,
                            Q => #Timer_10_END);

    IF #Timer_10_END AND "LeftSensor" THEN
        "Data_block_1"."move D2" := 0;
        "Data_block_1"."conv move" := 0;
        "Data_block_1"."conv ON/OFF" := 0;
        "Data_block_1"."conv home" := 0;
        "Data_block_1"."conv speed" := 0;
        "Start_Timer_10" := 0;
        "Data_block_1"."conv position" := 0;
        "STEP_6" := 0;
        "STEP_7" := 1;
    END_IF;
END_IF;


IF "STEP_7" THEN
    "Data_block_1"."angle D1" := 0;
    "Data_block_1"."move D1" := 1;
    #Start_Timer_11 := 1;
```

```
    "IEC_Tim_OFF_11_DB".TON(IN := #Start_Timer_11,
                            PT := T#10s,
                            ET => #ET_TIME_ON_11,
                            Q => #Timer_11_END);

    IF #Timer_11_END THEN
        "Data_block_1"."move D1" := 0;
        "Data_block_1"."conv home" := 1;
        "STEP_7" := 0;
        "STEP_8" := 1;
    END_IF;
END_IF;


IF "STEP_8" THEN
    "Data_block_1"."angle D2" := 0;
    "Data_block_1"."move D2" := 1;
    #Start_Timer_12 := 1;

    "IEC_Tim_OFF_12_DB".TON(IN := #Start_Timer_12,
                            PT := T#10s,
                            ET => #ET_TIME_ON_12,
                            Q => #Timer_12_END);

    IF #Timer_12_END THEN
        "Data_block_1"."move D2" := 0;
        "Data_block_1"."angle D3" := 0;
        "Data_block_1"."move D3" := 1;
        #Start_Timer_13 := 1;
    END_IF;
```

```
"IEC_Tim_OFF_13_DB".TON(IN := #Start_Timer_13,
                        PT := T#15s,
                        ET => #ET_TIME_ON_13,
                        Q => #Timer_13_END);

IF #Timer_13_END THEN
    "Data_block_1"."move D3" := 0;
    "Data_block_1"."angle D1" := 0;
    "STEP_8" := 0;
    "STEP_0_END" := 0;
    "Data_block_1"."On/off D1" := 0;
    "Data_block_1"."On/off D2" := 0;
    "Data_block_1"."On/off D3" := 0;
    "Data_block_1"."conv home" := 0;
    "Switch_AUT/MAN" := 0;
    RESET_TIMER("IEC_Tim_OFF_0_DB");
    RESET_TIMER("IEC_Tim_OFF_1_DB");
    RESET_TIMER("IEC_Tim_OFF_2_DB");
    RESET_TIMER("IEC_Tim_OFF_3_DB");
    RESET_TIMER("IEC_Tim_OFF_4_DB");
    RESET_TIMER("IEC_Tim_OFF_5_DB");
    RESET_TIMER("IEC_Tim_OFF_6_DB");
    RESET_TIMER("IEC_Tim_OFF_7_DB");
    RESET_TIMER("IEC_Tim_OFF_8_DB");
    RESET_TIMER("IEC_Tim_OFF_9_DB");
    RESET_TIMER("IEC_Tim_OFF_10_DB");
    RESET_TIMER("IEC_Tim_OFF_11_DB");
    RESET_TIMER("IEC_Tim_OFF_12_DB");
    RESET_TIMER("IEC_Tim_OFF_13_DB");
```

```
            END_IF;
        END_IF;

END_IF;
```

Кодування програми в Visual Code

```
#------------------------------
# imports
#------------------------------

# builtins
import os,sys,time,traceback
from math import hypot

import numpy as np
import cv2
import paho.mqtt.client as mqtt
import requests
import threading
import time

# local clayton libs
import frame_capture
import frame_draw

#------------------------------
# default settings
#------------------------------
```

```
# camera values
camera_id = 0
camera_width = 960
camera_height = 1080
camera_frame_rate = 30
#camera_fourcc = cv2.VideoWriter_fourcc(*"YUYV")
camera_fourcc = cv2.VideoWriter_fourcc(*"MJPG")

# auto measure mouse events
auto_percent = 0.2
auto_threshold = 127
auto_blur = 5

# normalization mouse events
norm_alpha = 0
norm_beta = 255

#------------------------------
# read config file
#------------------------------

# read config values
configfile = 'camruler_config.csv'
if os.path.isfile(configfile):
    with open(configfile) as f:
        for line in f:
            line = line.strip()
            if line and line[0] != '#' and (',' in line or '=' in line):
                if ',' in line:
                    item,value = [x.strip() for x in line.split(',',1)]
```

```
            elif '=' in line:
                item,value = [x.strip() for x in line.split('=',1)]
            else:
                continue
            if item in 'camera_id camera_width camera_height camera_frame_rate camera_fourcc auto_percent auto_threshold auto_blur norm_alpha norm_beta'.split():
                try:
                    exec(f'{item}={value}')
                    print('CONFIG:',(item,value))
                except:
                    print('CONFIG ERROR:',(item,value))

#------------------------------
# camera setup
#------------------------------

# get camera id from argv[1]
# "python3 camruler.py 2"
if len(sys.argv) > 1:
    camera_id = sys.argv[1]
    if camera_id.isdigit():
        camera_id = int(camera_id)

# camera thread setup
camera = frame_capture.Camera_Thread()
camera.camera_source = camera_id # SET THE CORRECT CAMERA NUMBER
camera.camera_width  = camera_width
camera.camera_height = camera_height
camera.camera_frame_rate = camera_frame_rate
camera.camera_fourcc = camera_fourcc
```

```
#1 start camera thread
camera.start()

# initial camera values (shortcuts for below)
width  = camera.camera_width
height = camera.camera_height
area = width*height
cx = int(width/2)
cy = int(height/2)
dm = hypot(cx,cy) # max pixel distance
frate  = camera.camera_frame_rate
print('CAMERA:',[camera.camera_source,width,height,area,frate])

#------------------------------
# frame drawing/text module
#------------------------------

draw = frame_draw.DRAW()
draw.width = width
draw.height = height

#------------------------------
# conversion (pixels to measure)
#------------------------------

# distance units designator
unit_suffix = 'mm'

# calibrate every N pixels
```

```
pixel_base = 10

# maximum field of view from center to farthest edge
# should be measured in unit_suffix
cal_range = 150

# initial calibration values table {pixels:scale}
# this is based on the frame size and the cal_range
cal = dict([(x,cal_range/dm) for x in range(0,int(dm)+1,pixel_base)])

# calibration loop values
# inside of main loop below
cal_base = 5
cal_last = None

# send data to Node-RED
def send_data(xlen, ylen):
    try:
        url = "http://localhost:1880/data"  # Node address Node-RED
        data = {"key": xlen, "key2": ylen}  # Data sent
        response = requests.post(url, json=data)
        print(response.text)
    except Exception as e:
        print(f"Error occurred: {e}")

# calibration update
def cal_update(x,y,unit_distance):

    # basics
    pixel_distance = hypot(x,y)
```

```
    scale = abs(unit_distance/pixel_distance)
    target = baseround(abs(pixel_distance),pixel_base)

    # low-high values in distance
    low  = target*scale - (cal_base/2)
    high = target*scale + (cal_base/2)

    # get low start point in pixels
    start = target
    if unit_distance <= cal_base:
        start = 0
    else:
        while start*scale > low:
            start -= pixel_base

    # get high stop point in pixels
    stop = target
    if unit_distance >= baseround(cal_range,pixel_base):
        high = max(cal.keys())
    else:
        while stop*scale < high:
            stop += pixel_base

    # set scale
    for x in range(start,stop+1,pixel_base):
        cal[x] = scale
        print(f'CAL: {x} {scale}')

# read calibration data
calfile = 'camruler_cal.csv'
```

```
if os.path.isfile(calfile):
    with open(calfile) as f:
        for line in f:
            line = line.strip()
            if line and line[0] in ('d',):
                axis,pixels,scale = [_.strip() for _ in line.split(',',2)]
                if axis == 'd':
                    print(f'LOAD: {pixels} {scale}')
                    cal[int(pixels)] = float(scale)

# convert pixels to units
def conv(x,y):

    d = distance(0,0,x,y)

    scale = cal[baseround(d,pixel_base)]

    return x*scale,y*scale

# round to a given base
def baseround(x,base=1):
    return int(base * round(float(x)/base))

# distance formula 2D
def distance(x1,y1,x2,y2):
    return hypot(x1-x2,y1-y2)

#------------------------------
# define frames
#------------------------------
```

```
# define display frame
framename = "CamRuler"
cv2.namedWindow(framename,flags=cv2.WINDOW_NORMAL|cv2.WINDOW_GUI_
NORMAL)

#------------------------------
# key events
#------------------------------

key_last = 0
key_flags = {'config':False, # c key
            'auto':False,   # a key
            'thresh':False, # t key
            'percent':False,# p key
            'norms':False,  # n key
            'rotate':False, # r key
            'lock':False,   #
            }

def key_flags_clear():

    global key_flags

    for key in list(key_flags.keys()):
        if key not in ('rotate',):
            key_flags[key] = False

def key_event(key):
```

```
global key_last
global key_flags
global mouse_mark
global cal_last

# config mode
if key == 99:
    if key_flags['config']:
        key_flags['config'] = False
    else:
        key_flags_clear()
        key_flags['config'] = True
        cal_last,mouse_mark = 0,None

# normilization mode
elif key == 110:
    if key_flags['norms']:
        key_flags['norms'] = False
    else:
        key_flags['thresh'] = False
        key_flags['percent'] = False
        key_flags['lock'] = False
        key_flags['norms'] = True
        mouse_mark = None

# rotate
elif key == 114:
    if key_flags['rotate']:
        key_flags['rotate'] = False
    else:
```

```
            key_flags['rotate'] = True

    # auto mode
    elif key == 97:
        if key_flags['auto']:
            key_flags['auto'] = False
        else:
            key_flags_clear()
            key_flags['auto'] = True
            mouse_mark = None

    # auto percent
    elif key == 112 and key_flags['auto']:
        key_flags['percent'] = not key_flags['percent']
        key_flags['thresh'] = False
        key_flags['lock'] = False

    # auto threshold
    elif key == 116 and key_flags['auto']:
        key_flags['thresh'] = not key_flags['thresh']
        key_flags['percent'] = False
        key_flags['lock'] = False

    # log
    print('key:',[key,chr(key)])
    key_last = key

#------------------------------
# mouse events
#------------------------------
```

```
# mouse events
mouse_raw  = (0,0) # pixels from top left
mouse_now  = (0,0) # pixels from center
mouse_mark = None  # last click (from center)

# mouse callback
def mouse_event(event,x,y,flags,parameters):

    # globals
    global mouse_raw
    global mouse_now
    global mouse_mark
    global key_last
    global auto_percent
    global auto_threshold
    global auto_blur
    global norm_alpha
    global norm_beta

    # update percent
    if key_flags['percent']:
        auto_percent = 5*(x/width)*(y/height)

    # update threshold
    elif key_flags['thresh']:
        auto_threshold = int(255*x/width)
        auto_blur = int(20*y/height) | 1 # insure it is odd and at least 1

    # update normalization
```

```
elif key_flags['norms']:
    norm_alpha = int(64*x/width)
    norm_beta  = min(255,int(128+(128*y/height)))

# update mouse location
mouse_raw = (x,y)

# offset from center
# invert y to standard quadrants
ox = x - cx
oy = (y-cy)*-1

# update mouse location
mouse_raw = (x,y)
if not key_flags['lock']:
    mouse_now = (ox,oy)

# left click event
if event == 1:

    if key_flags['config']:
        key_flags['lock'] = False
        mouse_mark = (ox,oy)

    elif key_flags['auto']:
        key_flags['lock'] = False
        mouse_mark = (ox,oy)

    if key_flags['percent']:
        key_flags['percent'] = False
```

```
            mouse_mark = (ox,oy)

        elif key_flags['thresh']:
            key_flags['thresh'] = False
            mouse_mark = (ox,oy)

        elif key_flags['norms']:
            key_flags['norms'] = False
            mouse_mark = (ox,oy)

        elif not key_flags['lock']:
            if mouse_mark:
                key_flags['lock'] = True
            else:
                mouse_mark = (ox,oy)
        else:
            key_flags['lock'] = False
            mouse_now = (ox,oy)
            mouse_mark = (ox,oy)

        key_last = 0

    # right click event
    elif event == 2:
        key_flags_clear()
        mouse_mark = None
        key_last = 0

# register mouse callback
cv2.setMouseCallback(framename,mouse_event)
```

```
#-----------------------------
# main loop
#-----------------------------

# loop
while 1:

    # get frame
    frame0 = camera.next(wait=1)
    if frame0 is None:
        time.sleep(0.1)
        continue

    # normalize
    cv2.normalize(frame0,frame0,norm_alpha,norm_beta,cv2.NORM_MINMAX)

    # rotate 180
    if key_flags['rotate']:
        frame0 = cv2.rotate(frame0,cv2.ROTATE_180)

    # start top-left text block
    text = []

    # camera text
    fps = camera.current_frame_rate
    text.append(f'CAMERA: {camera_id} {width}x{height} {fps}FPS')

    # mouse text
    text.append('')
```

```
if not mouse_mark:
    text.append(f'LAST CLICK: NONE')
else:
    text.append(f'LAST CLICK: {mouse_mark} PIXELS')
text.append(f'CURRENT XY: {mouse_now} PIXELS')

#------------------------------
# normalize mode
#------------------------------
if key_flags['norms']:

    # print
    text.append('')
    text.append(f'NORMILIZE MODE')
    text.append(f'ALPHA (min): {norm_alpha}')
    text.append(f'BETA (max): {norm_beta}')

#------------------------------
# config mode
#------------------------------
if key_flags['config']:

    # quadrant crosshairs
    draw.crosshairs(frame0,5,weight=2,color='red',invert=True)

    # crosshairs aligned (rotated) to maximum distance
    draw.line(frame0,cx,cy, cx+cx, cy+cy,weight=1,color='red')
    draw.line(frame0,cx,cy, cx+cy, cy-cx,weight=1,color='red')
    draw.line(frame0,cx,cy,-cx+cx,-cy+cy,weight=1,color='red')
    draw.line(frame0,cx,cy, cx-cy, cy+cx,weight=1,color='red')
```

```
# mouse cursor lines (parallel to aligned crosshairs)
mx,my = mouse_raw
draw.line(frame0,mx,my,mx+dm,my+(dm*( cy/cx)),weight=1,color='green')
draw.line(frame0,mx,my,mx-dm,my-(dm*( cy/cx)),weight=1,color='green')
draw.line(frame0,mx,my,mx+dm,my+(dm*(-cx/cy)),weight=1,color='green')
draw.line(frame0,mx,my,mx-dm,my-(dm*(-cx/cy)),weight=1,color='green')

# config text data
text.append('')
text.append(f'CONFIG MODE')

# start cal
if not cal_last:
    cal_last = cal_base
    caltext = f'CONFIG: Click on D = {cal_last}'

# continue cal
elif cal_last <= cal_range:
    if mouse_mark:
        cal_update(*mouse_mark,cal_last)
        cal_last += cal_base
    caltext = f'CONFIG: Click on D = {cal_last}'

# done
else:
    key_flags_clear()
    cal_last == None
    with open(calfile,'w') as f:
        data = list(cal.items())
```

```
            data.sort()
            for key,value in data:
                f.write(f'd,{key},{value}\n')
            f.close()
        caltext = f'CONFIG: Complete.'

    # add caltext
    draw.add_text(frame0,caltext,(cx)+100,(cy)+30,color='red')

    # clear mouse
    mouse_mark = None

#------------------------------
# auto mode
#------------------------------
elif key_flags['auto']:

    mouse_mark = None

    # auto text data
    text.append('')
    text.append(f'AUTO MODE')
    text.append(f'UNITS: {unit_suffix}')
    text.append(f'MIN PERCENT: {auto_percent:.2f}')
    text.append(f'THRESHOLD: {auto_threshold}')
    text.append(f'GAUSS BLUR: {auto_blur}')

    # gray frame
    frame1 = cv2.cvtColor(frame0,cv2.COLOR_BGR2GRAY)
```

```
        # blur frame
        frame1 = cv2.GaussianBlur(frame1,(auto_blur,auto_blur),0)

        # threshold frame n out of 255 (85 = 33%)
        frame1 = cv2.threshold(frame1,auto_threshold,255,cv2.THRESH_BINARY)[1]

        # invert
        frame1 = ~frame1

        # find contours on thresholded image
        contours,nada                                                    =
cv2.findContours(frame1,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)

        # small crosshairs (after getting frame1)
        draw.crosshairs(frame0,5,weight=2,color='green')

        # loop over the contours
        for c in contours:

            # contour data (from top left)
            x1,y1,w,h = cv2.boundingRect(c)
            x2,y2 = x1+w,y1+h
            x3,y3 = x1+(w/2),y1+(h/2)

            # percent area
            percent = 100*w*h/area

            # if the contour is too small, ignore it
            if percent < auto_percent:
                continue
```

```
        # if the contour is too large, ignore it
        elif percent > 60:
            continue

        # convert to center, then distance
        x1c,y1c = conv(x1-(cx),y1-(cy))
        x2c,y2c = conv(x2-(cx),y2-(cy))
        xlen = abs(x1c-x2c)
        ylen = abs(y1c-y2c)
        alen = 0
        if max(xlen,ylen) > 0 and min(xlen,ylen)/max(xlen,ylen) >= 0.95:
           alen = (xlen+ylen)/2
        carea = xlen*ylen

        # plot
        draw.rect(frame0,x1,y1,x2,y2,weight=2,color='red')

        # add dimensions
        draw.add_text(frame0,f'{xlen:.2f}',x1-((x1-x2)/2),min(y1,y2)-
8,center=True,color='red')
        draw.add_text(frame0,f'Area:
{carea:.2f}',x3,y2+8,center=True,top=True,color='red')
        print (f'{xlen:.2f}',f'{ylen:.2f}')
        if alen:
           draw.add_text(frame0,f'Avg:
{alen:.2f}',x3,y2+34,center=True,top=True,color='green')
        if x1 < width-x2:
           draw.add_text(frame0,f'{ylen:.2f}',x2+4,(y1+y2)/2,middle=True,color='red')
        else:
```

```
            draw.add_text(frame0,f'{ylen:.2f}',x1-
4,(y1+y2)/2,middle=True,right=True,color='red')

        thread2 = threading.Thread(target=send_data, args=(xlen, ylen))
        thread2.daemon = True
        thread2.start()

    #------------------------------
    # dimension mode
    #------------------------------
    else:

        # small crosshairs
        draw.crosshairs(frame0,5,weight=2,color='green')

        # mouse cursor lines
        draw.vline(frame0,mouse_raw[0],weight=1,color='green')
        draw.hline(frame0,mouse_raw[1],weight=1,color='green')

        # draw
        if mouse_mark:

            # locations
            x1,y1 = mouse_mark
            x2,y2 = mouse_now

            # convert to distance
            x1c,y1c = conv(x1,y1)
            x2c,y2c = conv(x2,y2)
            xlen = abs(x1c-x2c)
```

```
ylen = abs(y1c-y2c)
llen = hypot(xlen,ylen)
alen = 0
if max(xlen,ylen) > 0 and min(xlen,ylen)/max(xlen,ylen) >= 0.95:
    alen = (xlen+ylen)/2
carea = xlen*ylen

# print distances
text.append('')
text.append(f'X LEN: {xlen:.2f}{unit_suffix}')
text.append(f'Y LEN: {ylen:.2f}{unit_suffix}')
text.append(f'L LEN: {llen:.2f}{unit_suffix}')

# convert to plot locations
x1 += cx
x2 += cx
y1 *= -1
y2 *= -1
y1 += cy
y2 += cy
x3 = x1+((x2-x1)/2)
y3 = max(y1,y2)

# line weight
weight = 1
if key_flags['lock']:
    weight = 2

# plot
draw.rect(frame0,x1,y1,x2,y2,weight=weight,color='red')
```

```
            draw.line(frame0,x1,y1,x2,y2,weight=weight,color='green')

            # add dimensions
            draw.add_text(frame0,f'{xlen:.2f}',x1-((x1-x2)/2),min(y1,y2)-
8,center=True,color='red')
            draw.add_text(frame0,f'Area:
{carea:.2f}',x3,y3+8,center=True,top=True,color='red')
            if alen:
                draw.add_text(frame0,f'Avg:
{alen:.2f}',x3,y3+34,center=True,top=True,color='green')
            if x2 <= x1:
                draw.add_text(frame0,f'{ylen:.2f}',x1+4,(y1+y2)/2,middle=True,color='red')
                draw.add_text(frame0,f'{llen:.2f}',x2-4,y2-4,right=True,color='green')
            else:
                draw.add_text(frame0,f'{ylen:.2f}',x1-
4,(y1+y2)/2,middle=True,right=True,color='red')
                draw.add_text(frame0,f'{llen:.2f}',x2+8,y2-4,color='green')

    # add usage key
    text.append('')
    text.append(f'Q = QUIT')
    text.append(f'R = ROTATE')
    text.append(f'N = NORMALIZE')
    text.append(f'A = AUTO-MODE')
    if key_flags['auto']:
        text.append(f'P = MIN-PERCENT')
        text.append(f'T = THRESHOLD')
        text.append(f'T = GAUSS BLUR')
    text.append(f'C = CONFIG-MODE')
```

```
    # draw top-left text block
    draw.add_text_top_left(frame0,text)

    # display
    cv2.imshow(framename,frame0)

    # key delay and action
    key = cv2.waitKey(1) & 0xFF

    # esc ==  27 == quit
    # q   == 113 == quit
    if key in (27,113):
        break

    # key data
    #elif key != 255:
    elif key not in (-1,255):
        key_event(key)

#------------------------------
# kill sequence
#------------------------------

# close camera thread
camera.stop()

# close all windows
cv2.destroyAllWindows()

# done
```

```
exit()

#------------------------------
# end
#------------------------------
```

Кодування програми в Node-RED

```
let key = msg.payload.key;     // Отримуємо значення "key"
let key2 = msg.payload.key2;   // Отримуємо значення "key2"
// Зберігаємо їх у нових полях
msg.key = key;
msg.key2 = key2;
msg.payload = [
    { nodeId: "ns=4;i=2", value: { dataType: "Float", value: msg.key } },
    { nodeId: "ns=4;i=3", value: { dataType: "Float", value: msg.key2 } }
];
return msg;
```

ДОДАТОК Б

Проект методичних вказівок

Лабораторна робота №1

Тема: Дослідження процесу калібрування веб-камери.

Мета: Вивчення алгоритму калібрування для коректного розпізнавання об’єктів та їх розмірів.

Теоретичні відомості

Для коректної роботи можуть знадобитись додаткові налаштування параметрів. Наприклад якщо налаштоване занадто низьке порогове значення того що саме визначається як об’єкт то результат може виглядати як на рис. Б.2, коли окремими об’єктами можуть визначатись навіть дуже малі частини, тіні або частини текстури/рельєфу на об’єкті.. Для того щоб це виправити необхідно натиснути кнопку налаштування порогових значень та гаусового розмиття – кнопка «T». Після чого положення курсору миші в вікні програми починає змінювати значення цих параметрів. Рухи по горизонталі змінюють порогове значення (від меншого зліва, до більшого справа), рухи по вертикалі, в свою чергу, змінюють значення розмиття контурів об’єкта для алгоритму (на екрані розмиття не відображається), при цьому значення змінюються від меншого зверху до більшого знизу.

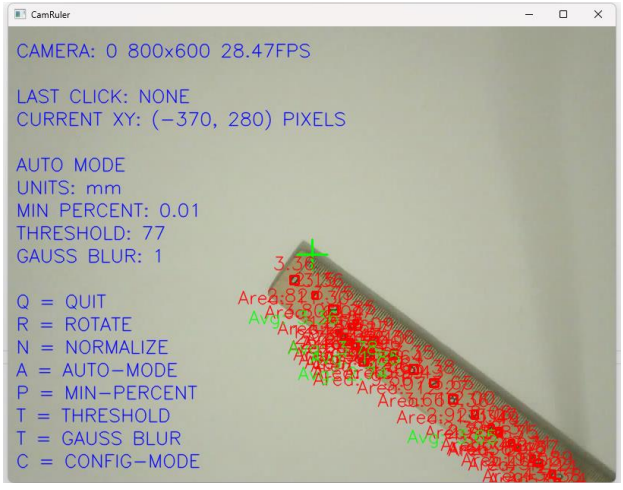


Рисунок Б.1 – Приклад некоректно налаштованих параметрів порогового значення та розмиття

Також, якщо змінювалось положення камери, а саме висота до об’єкту вимірювання, то необхідно провести перекалібровку параметрів для отримання коректних значень розмірів об’єкта. Для цього необхідно натиснути кнопку «C» чим відкриється вікно режиму конфігурації (рис. Б.2). При цьому на екрані з’являється перехрестя. По ньому необхідно вирівняти лінійку і записати з неї найбільшу довжину в мм до краю зображення до змінної cal_range в Python. Після цього необхідно перехрестя курсору пересувати вздовж лінійки, від центру до кута, і лівою клавішею миші відмічати крок в 5 мм. Це необхідно для корекції спотворень зображення (особливо якщо камера має більш виражений ефект розтягування зображення на його краях).

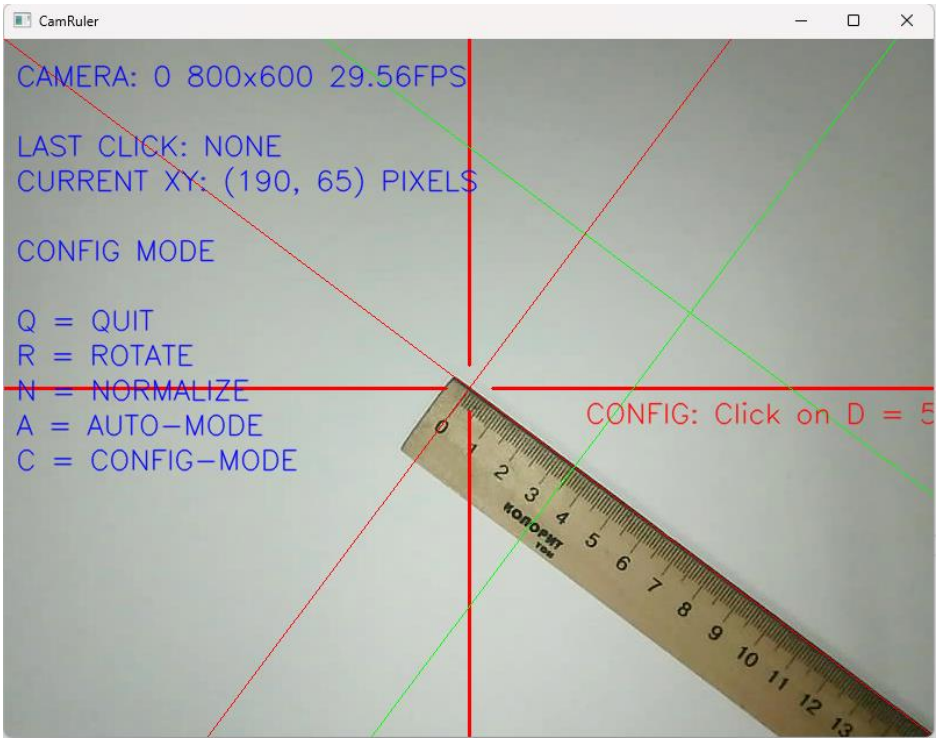


Рисунок Б.2 – Процес калібрування в режимі конфігурації

Для налаштування нормалізації зображення необхідно натиснути кнопку «N» та за аналогією пересуваючи курсор виконати налаштування (лівий верхній кут – найяскравіше зображення, правий нижній – найтемніше) (рис. Б.3).

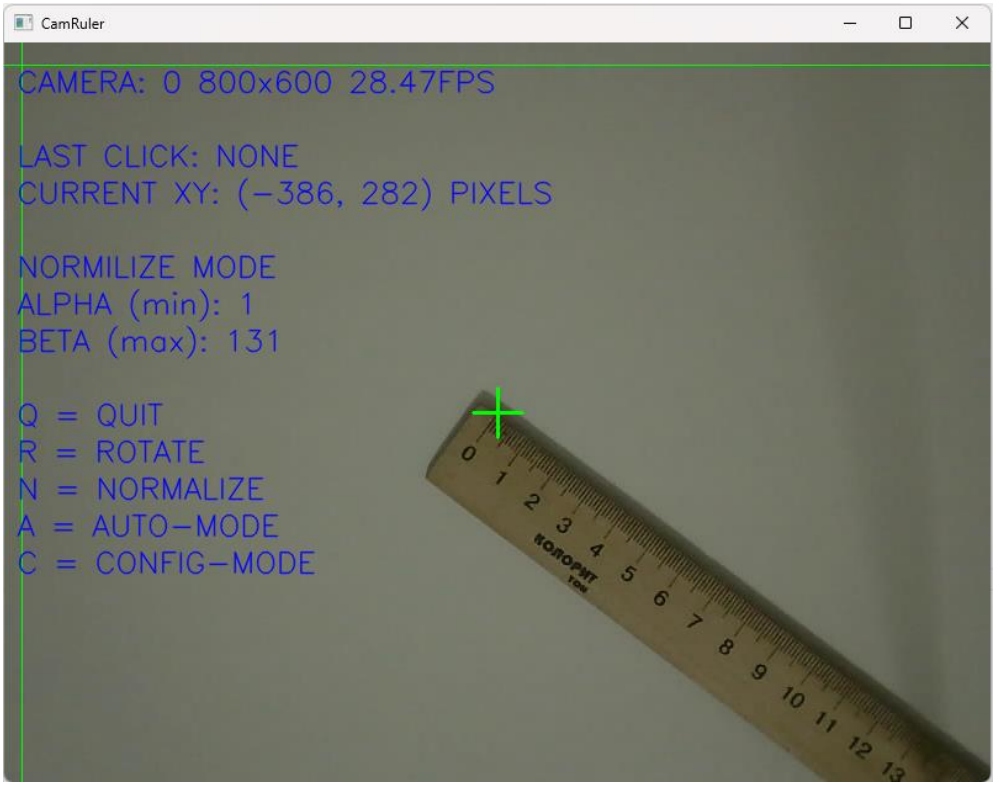


Рисунок Б.3 – Процес нормалізації зображення

Після натискання кнопки «A» на клавіатурі програма переходить в автоматичний режим роботи, в якому вона намагається ідентифікувати об’єкт у полі зору камери і визначити його розміри (рис. Б.4).

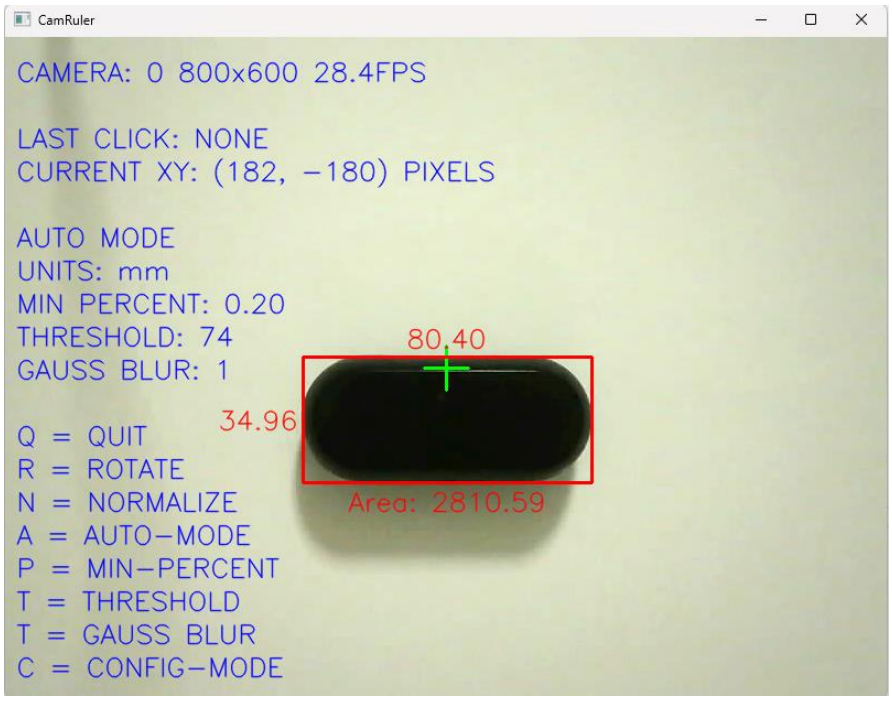


Рисунок Б.4 – Вікно автоматичного режиму роботи

Якщо налаштування було проведено правильно, то можна побачити результат як на рисунку вище.

Завдання

1. Ознайомитися з теоретичної частиною лабораторної роботи.
2. Розглянути алгоритм роботи програми.
3. Виконати процес калібрування роботи камери.
4. Дослідити вплив якості освітлення на роботи системи.

Контрольні запитання

1. Принцип роботи досліджуваної системи?
2. Принцип підключення камери?
3. Як виконується налаштування датчика?
4. Які зовнішні фактори впливають на роботу системи?
5. Які параметри піддаються калібруванню?

Лабораторна робота №2

Тема: Дослідження процесу роботи програми ідентифікації параметрів деталей з використанням веб-камери.

Мета: Вивчення принципу та алгоритму роботи програми ідентифікації параметрів деталей.

Теоретичні відомості

В якості датчику для розпізнавання розмірів деталей використовується веб-камера (рис. Б.5). Використання веб-камери як датчика для зчитування розмірів об'єктів із зображення має низку переваг, що обґрунтовують її вибір у багатьох проектах. Веб-камери є доступними, недорогими та легко інтегруються з комп’ютерними системами, що робить їх економічно вигідним рішенням для проектів, де необхідно забезпечити функціонал вимірювання без значних витрат. Сучасні веб-камери забезпечують достатню роздільну здатність для отримання якісних зображень, які можна обробляти за допомогою алгоритмів комп'ютерного зору, таких як OpenCV. Завдяки таким алгоритмам можливо вимірювати розміри об'єктів, використовуючи визначення контурів, аналіз геометрії чи порівняння масштабів.

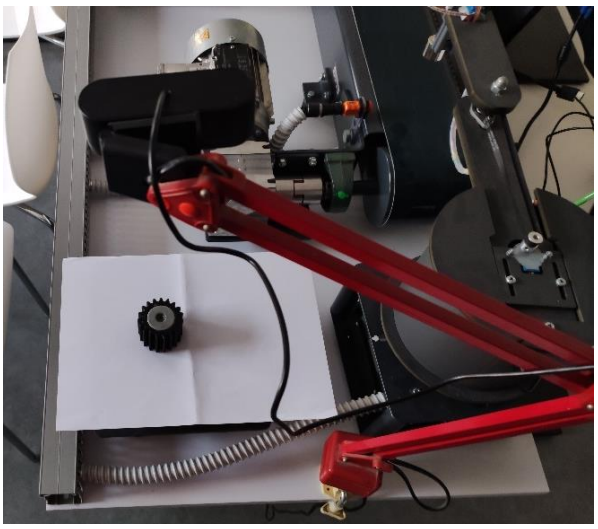


Рисунок Б.5 – Зовнішній вигляд встановлення камери на стенді

Універсальність веб-камер дозволяє застосовувати їх для аналізу об'єктів різної форми, розміру та розташування. Програмні інструменти забезпечують автоматизацію процесу розпізнавання та вимірювання, а калібрування камери за допомогою еталонних об'єктів дає змогу досягти високої точності. Веб-камери також підтримують роботу в реальному часі, що робить їх зручними для використання в системах моніторингу, контролю або аналізу динамічних процесів. Їх легко інтегрувати з різними платформами, такими як комп’ютери, Raspberry Pi чи спеціалізовані контролери, що дозволяє створювати автоматизовані вимірювальні системи.

Однак є й виклики, які необхідно враховувати. Для точного вимірювання потрібно калібрувати камеру з урахуванням перспективних спотворень та фокусної відстані. Крім того, якість вимірювання залежить від стабільності освітлення, тому важливо забезпечити належні умови для роботи системи. Програмна реалізація вимірювань вимагає розробки або налаштування алгоритмів аналізу. Незважаючи на ці виклики, веб-камера є ефективним та практичним рішенням для задач вимірювання, особливо для низькобюджетних проектів чи створення прототипів.

Хід роботи

Після запуску програми в Visual code перед собою можна побачити головне вікно програми – зображення з камери, її параметри, інформація останнього натискання курсору та підказки для кнопок навігації по меню (рис. Б.6).

Рисунок Б.6 – Код режиму калібрування

Після натискання кнопки «A» на клавіатурі програма переходить в автоматичний режим роботи, в якому вона намагається ідентифікувати об’єкт у полі зору камери і визначити його розміри (рис. Б.7).

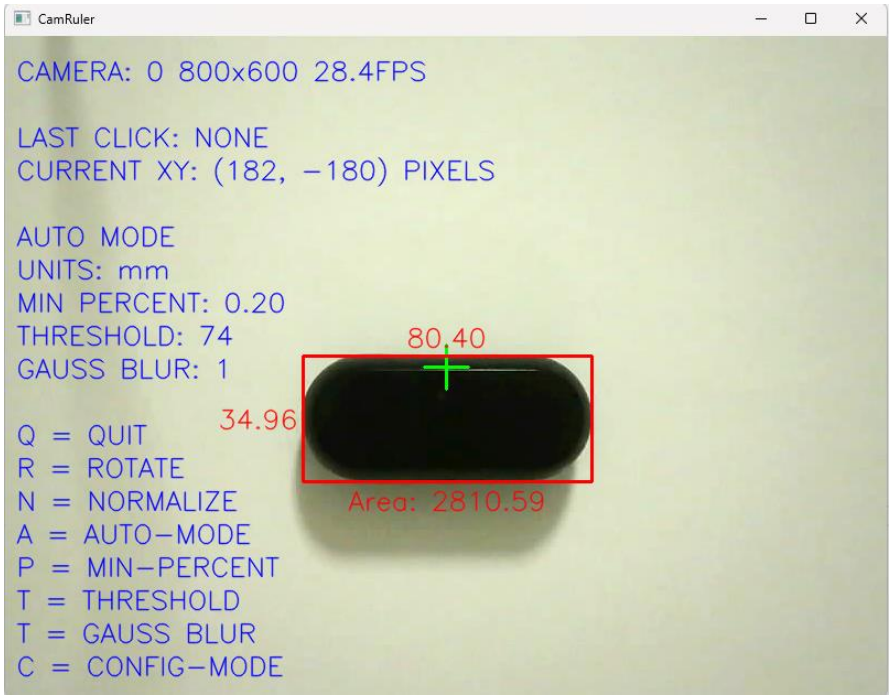


Рисунок Б.7 – Вікно автоматичного режиму роботи

Також потрібно запустити програму в Node-RED, а також UAExpert для роботи OPC UA серверу. В TIA Portal необхідно відкрити панель оператора (рис. Б.8).

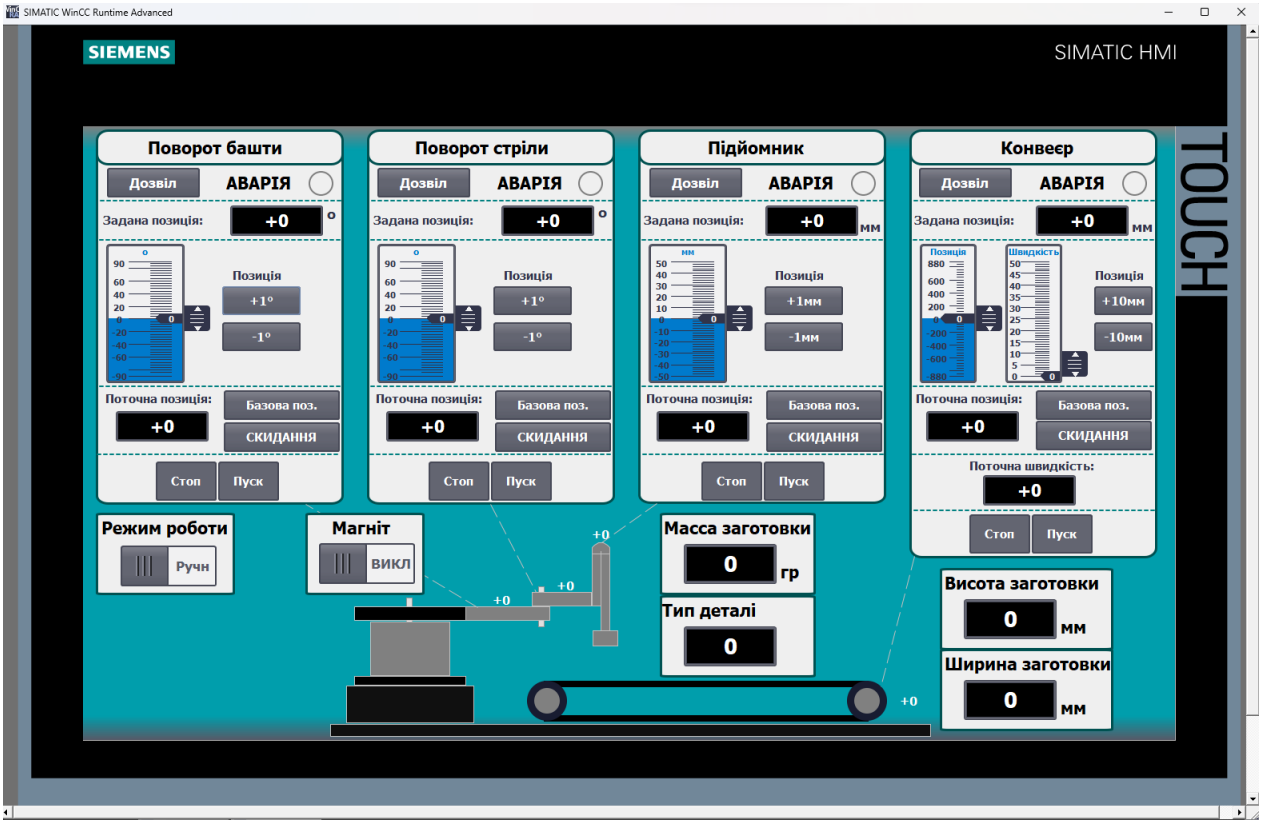


Рисунок Б.8 – Зовнішній вигляд панелі оператора

Для роботи з програмою необхідно перемкнути повзунок «Режим роботи» в режим «авто» і натиснути кнопку, що з’явиться – «Пуск АВТ». Після цього програма автоматично виконає цикл переміщення вантажу, при цьому відображаючи на екрані його вагу та розміри.

Завдання

1. Ознайомитися з теоретичної частиною лабораторної роботи.
2. Вивчити принцип роботи системи.
3. Дослідити інтерфейс програми
4. Розглянути алгоритм роботи програми.

Контрольні запитання

1. Принцип роботи досліджуваної системи?
2. Принцип підключення камери?
3. Які ще існують варіанти реалізації схожої системи?
4. Переваги та недоліки даного типу вимірювання?
5. Діапазон вимірювання?