



Politechnika
Śląska

PROJEKT INŻYNIERSKI

Robotyczny system sortowania klocków

Jakub PAJĄK

Nr albumu: 300566

Kierunek: Automatyka i Robotyka

Specjalność: Technologie Informacyjne w Automatyce i Robotyce

PROWADZĄCY PRACĘ

dr inż. Krzysztof Jaskot

KATEDRA Automatyki i Robotyki

Wydział Automatyki, Elektroniki i Informatyki

Gliwice 2024

Tytuł pracy

Robotyczny system sortowania klocków

Streszczenie

Celem projektu jest wykonanie autonomicznej platformy jezdnej, mającej na celu bezobsługowe sortowanie klocków spadających w określone miejsce z taśmociągu lub innego podajnika. Robot korzystając z możliwości wizji komputerowej będzie rozpoznawać kolor aktualnego klocka, a następnie na podstawie określonego koloru będzie wybierać trasę do odpowiedniego pojemnika. Domyślnie rozpoznawane będą trzy kolory (czerwony, zielony, niebieski) oraz trzy pojemniki, odpowiednie dla każdego koloru. Wymagania: znajomość programowania układów SBC (Raspberry Pi), Python, OpenCV

Słowa kluczowe

Robot mobilny, analiza wizyjna, Programowanie mikrokontrolerów

Thesis title

Robotic brick sorting system

Abstract

The goal of this project is to create an autonomous mobile platform designed for unattended sorting of blocks falling into a designated area from a conveyor belt or another feeder. The robot will use computer vision to recognize the color of the current block and, based on the identified color, will choose the path to the appropriate container. By default, three colors (red, green, blue) will be recognized, and there will be three containers, each corresponding to one of the colors. Requirements: knowledge of SBC programming (Raspberry Pi), Python, OpenCV

Key words

Mobile robot, visual analysis, microcontroller programming

Spis treści

1	Wstęp	1
1.1	Wprowadzenie w problem	1
1.2	Osadzenie problemu w dziedzinie	2
1.3	Cel pracy	2
1.4	Zakres pracy	2
1.5	Zwięzła charakterystyka rozdziałów	3
1.6	Wkład autora	3
2	Analiza tematu	5
2.1	Sformułowanie problemu	5
2.2	Osadzenie tematu w kontekście aktualnego stanu wiedzy (<i>state of the art</i>) .	5
2.3	Studia literaturowe	5
2.3.1	Opracowanie systemu wizyjnego	6
2.3.2	Opracowanie systemu kontroli silników	6
3	Założenia wstępne	9
3.1	Wymagania funkcjonalne i нефункционалне	9
3.2	Przypadki użycia i diagramy UML	10
3.3	Opis wykorzystanego sprzętu elektronicznego	10
3.4	Metodyka pracy i etapy projektu	11
4	Podstawy teoretyczne	13
4.1	Enkodery	13
4.1.1	Rodzaje enkoderów	13
4.1.2	Enkodery kwadraturowe	14
4.2	Regulator PID	15
4.2.1	Ogólny schemat układu regulacji	16
4.2.2	Działanie regulatora PID	16
4.2.3	Składnik proporcjonalny P	16
4.2.4	Składnik całkujący I	17
4.2.5	Składnik różniczkujący D	17

4.2.6	Pełne równanie regulatora PID	17
4.2.7	Zastosowanie regulatora PID w sterowaniu robotem mobilnym . . .	18
4.3	Podstawy wykrywania krawędzi w wizji komputerowej	18
4.3.1	Algorytm Canny’ego	18
4.4	Podstawy rozpoznawania kolorów w wizji komputerowej	19
4.4.1	Teoria koloru	19
4.4.2	Proces rozpoznawania kolorów	20
4.4.3	Algorytmy rozpoznawania kolorów	20
5	Wymagania i specyfikacja użytkowa	23
5.1	Wymagania sprzętowe i programowe	23
5.2	Sposób instalacji	24
5.2.1	Instalacja systemu operacyjnego	24
5.2.2	Aktualizacja systemu	25
5.2.3	Instalacja Python	25
5.2.4	Instalacja OpenCV	26
5.2.5	Instalacja Libcamera2	26
5.2.6	Instalacja RPi.GPIO	26
5.2.7	Weryfikacja instalacji pakietów	26
5.2.8	Sposób aktywacji	27
5.2.9	Kategorie użytkowników	28
5.2.10	Sposób obsługi	28
5.2.11	Kwestie bezpieczeństwa	28
5.2.12	Przykład działania	29
5.2.13	Scenariusze korzystania z systemu	29
6	Specyfikacja techniczna	31
6.1	Konstrukcja	31
6.2	Implementacja kontrolera silników	36
6.2.1	Konfiguracja pinów i ustawienia PWM	36
6.2.2	Algorytm sterowania PID	36
6.2.3	Funkcja zliczająca impulsy enkoderów	36
6.2.4	Realizacja komunikacji z Raspberry Pi	37
6.2.5	Pętla główna programu	37
6.3	Implementacja głównej aplikacji sterującej	41
6.3.1	Opis funkcji głównej aplikacji <code>mainTask()</code>	41
6.3.2	Opis sposobu komunikacji szeregowej	43
6.3.3	Opis sposobu detekcji klocka oraz rozpoznawania koloru	45
7	Weryfikacja i walidacja	51

8 Podsumowanie i wnioski	53
Bibliografia	56
Spis skrótów i symboli	59
Źródła	61
Lista dodatkowych plików, uzupełniających tekst pracy	63
Spis rysunków	66
Spis tabel	67

Rozdział 1

Wstęp

Niniejsza praca inżynierska koncentruje się na zaprojektowaniu i budowie autonomicznej platformy jezdnej, której zadaniem jest sortowanie klocków na podstawie ich koloru. W dobie rozwijającej się automatyzacji i robotyzacji coraz większą rolę odgrywają systemy zdolne do autonomicznego wykonywania zadań, które wcześniej wymagały udziału człowieka. Projekt ten wkomponowuje się w ten trend, mając na celu zautomatyzowanie procesu sortowania elementów w środowisku przemysłowym lub magazynowym. W tym rozdziale zostaną omówione kluczowe aspekty projektu, w tym wprowadzenie w problem, osadzenie go w odpowiednim kontekście naukowym oraz zakres i cele pracy.

1.1 Wprowadzenie w problem

Problem, który stara się rozwiązać niniejszy projekt, dotyczy automatyzacji procesów sortowania w różnych środowiskach przemysłowych, takich jak linie produkcyjne czy magazyny. W tradycyjnym podejściu sortowanie odbywa się ręcznie lub z wykorzystaniem prostych mechanicznych sorterów, które często są mało elastyczne i kosztowne w utrzymaniu. W odpowiedzi na to wyzwanie, projekt ten przewiduje stworzenie autonomicznego robota mobilnego, wykorzystującego kamerę i algorytmy wizji komputerowej do rozpoznawania kolorów klocków oraz następnie do przewiezienia ich do odpowiednich pojemników.

Takie rozwiązanie jest szczególnie przydatne w sytuacjach, gdzie istnieje potrzeba dynamicznej zmiany parametrów sortowania lub tam, gdzie wymagana jest szybka reakcja na zmiany w procesie produkcyjnym. Automatyzacja pozwala również na zmniejszenie kosztów operacyjnych, zwiększenie efektywności oraz redukcję błędów ludzkich w procesie sortowania.

1.2 Osadzenie problemu w dziedzinie

Tematyka projektu wpisuje się w kilka dynamicznie rozwijających się dziedzin, takich jak robotyka mobilna, automatyzacja przemysłowa oraz wizja komputerowa. Roboty mobilne, zdolne do samodzielnej nawigacji oraz wykonywania skomplikowanych zadań, znajdują coraz szersze zastosowanie w przemyśle, logistyce, a także w gospodarstwach domowych. W kontekście niniejszego projektu, kluczową rolę odgrywa zastosowanie odometrii oraz systemów napędowych, które pozwalają na precyzyjną kontrolę ruchu robota.

Wizja komputerowa, z kolei, umożliwia rozpoznawanie obiektów na podstawie ich cech wizualnych, takich jak kolor czy kształt. Technologia ta, oparta na bibliotekach takich jak OpenCV, stała się dostępna nawet dla małych systemów wbudowanych, takich jak Raspberry Pi. W niniejszym projekcie robot wykorzystuje kamerę do rejestrowania obrazów klocków, które następnie są analizowane w czasie rzeczywistym w celu określenia koloru, a na tej podstawie podejmowana jest decyzja gdzie analizowany klocek powinien zostać przetransportowany.

1.3 Cel pracy

Głównym celem projektu jest zaprojektowanie i zbudowanie autonomicznej platformy jezdnej, która potrafi sortować klocki na podstawie ich koloru. Robot będzie wykorzystywał kamerę oraz algorytmy wizji komputerowej do identyfikacji kolorów takich jak czerwony, zielony oraz niebieski, a następnie będzie odpowiednio przekierowywał klocki do odpowiednich pojemników. Proces ten ma odbywać się w pełni automatycznie, bez potrzeby ingerencji człowieka.

Dodatkowym celem jest stworzenie systemu kontroli, który pozwoli na bieżąco monitorować pracę robota oraz ewentualnie wprowadzać modyfikacje w jego działaniu. System powinien być elastyczny, umożliwiając łatwe rozszerzenie o dodatkowe funkcje, takie jak rozpoznawanie większej liczby kolorów lub innych cech klocków.

1.4 Zakres pracy

Projekt obejmuje szeroki zakres działań, zarówno w zakresie sprzętowym, jak i programistycznym. Poniżej przedstawiono główne etapy realizacji:

- Zbudowanie fizycznej platformy jezdnej, w tym konstrukcji mechanicznej oraz systemu napędowego z enkoderami,
- Implementacja systemu rozpoznawania kolorów za pomocą kamery oraz algorytmów wizji komputerowej z użyciem Raspberry Pi i biblioteki OpenCV,

- Opracowanie algorytmu podejmowania decyzji na podstawie rozpoznanego koloru i przekierowania klocka do odpowiedniego pojemnika,
- Testowanie i optymalizacja działania systemu, aby zapewnić jego poprawne funkcjonowanie w rzeczywistych warunkach, takich jak linie produkcyjne czy magazyny.

Zakres projektu przewiduje także możliwość dalszej rozbudowy o nowe funkcjonalności, co zwiększa elastyczność i potencjalne zastosowanie platformy w innych dziedzinach, takich jak automatyczne sortowanie elementów na podstawie kształtu czy materiału.

1.5 Zwięzła charakterystyka rozdziałów

Struktura pracy została podzielona na następujące rozdziały:

- **Rozdział 1: Wstęp** – Przedstawienie problemu, celów oraz zakresu projektu.
- **Rozdział 2: Analiza tematu** – Przegląd dostępnych rozwiązań, technik oraz technologii stosowanych w podobnych systemach, a także omówienie aktualnego stanu wiedzy w tej dziedzinie.
- **Rozdział 3: Wymagania i narzędzia** – Opis wymagań projektowych oraz narzędzi, takich jak Raspberry Pi, OpenCV oraz Python, które zostaną użyte do realizacji projektu.
- **Rozdział 4: Specyfikacja zewnętrzna** – Opis funkcjonalności systemu z perspektywy użytkownika, w tym interfejsu oraz interakcji z otoczeniem.
- **Rozdział 5: Specyfikacja wewnętrzna** – Szczegółowa analiza architektury wewnętrznej systemu, w tym struktury oprogramowania i integracji z komponentami sprzętowymi.
- **Rozdział 6: Weryfikacja i walidacja** – Opis przeprowadzonych testów, ocena efektywności systemu oraz zgodności z założeniami projektowymi.
- **Rozdział 7: Podsumowanie i wnioski** – Zakończenie projektu, omówienie osiągniętych celów, a także wskazanie możliwych kierunków rozwoju systemu.

1.6 Wkład autora

Projekt został w pełni zrealizowany przez autora. Autor odpowiada za wszystkie etapy pracy, począwszy od projektowania i budowy platformy, poprzez implementację algorytmów rozpoznawania kolorów i sterowania ruchem robota, aż po testowanie i optymalizację systemu. Ponadto, autor zaprojektował system kontrolny, który pozwala na monitorowanie pracy robota w czasie rzeczywistym oraz umożliwia wprowadzenie ewentualnych korekt.

Rozdział 2

Analiza tematu

W rozdziale tym zostanie przeanalizowany temat projektu, z uwzględnieniem aktualnego stanu wiedzy oraz znanych rozwiązań dotyczących automatycznych systemów sortowania za pomocą wizji komputerowej.

2.1 Sformułowanie problemu

Głównym problemem rozwiązywanym w tym projekcie jest automatyzacja procesu sortowania obiektów według ich cech wizualnych, w tym przypadku koloru. Tradycyjne metody sortowania wymagają interwencji człowieka lub stosowania mechanicznych sorterów, które często są mniej elastyczne i mniej precyzyjne w identyfikacji złożonych cech. Celem projektu jest wykorzystanie technologii wizji komputerowej, aby umożliwić robotowi autonomiczne rozpoznawanie i sortowanie klocków według koloru.

2.2 Osadzenie tematu w kontekście aktualnego stanu wiedzy (*state of the art*)

Temat projektu osadza się w dziedzinach robotyki mobilnej oraz wizji komputerowej, które rozwijają się bardzo dynamicznie w ostatnich latach. Rozwiązania oparte na wizji komputerowej, zwłaszcza w połączeniu z platformami SBC (Single-Board Computer), takimi jak Raspberry Pi, stają się coraz bardziej popularne w automatyzacji procesów przemysłowych i logistyczne. Wykorzystanie takich narzędzi jak OpenCV w projektach robotycznych pozwala na zwiększenie elastyczności i precyzji systemów sortujących.

2.3 Studia literaturowe

Podczas prac nad poszczególnymi elementami projektu istotną rolę odgrywała analiza dostępnej literatury naukowej. Poniżej znajduje się spis artykułów, książek, dokumentacji

oraz stron internetowych, których treść okazała się przydatna podczas pracy.

2.3.1 Opracowanie systemu wizyjnego

Projektowy system wizyjny oparty został na dedykowanej kamerze dla platformy Raspberry Pi oraz na popularnej bibliotece OpenCV, co pozwala na efektywną analizę obrazu i wykrywanie obiektów w czasie rzeczywistym. W tym celu kluczowa okazała się dokumentacja biblioteki Libcamera2 [7], oferująca szczegółowe informacje na temat konfiguracji kamery, procedur inicjalizacji, oraz wykorzystania łącza CSI (Camera Serial Interface) do przesyłu obrazu. Dokumentacja dostarczona przez Raspberry Pi Ltd zawiera obszerne wytyczne dotyczące ustawień obrazu, obsługi błędów oraz optymalizacji parametrów obrazu pod kątem specyficznych wymagań, co stanowiło cenną pomoc podczas implementacji i konfiguracji systemu wizyjnego.

W celu bardziej zaawansowanego przetwarzania obrazu wykorzystano bibliotekę OpenCV. Dla zrozumienia jej funkcjonalności, zwłaszcza w kontekście realizacji projektu, cennym źródłem była literatura specjalistyczna, m.in. książka [4], która stanowi kompleksowy przegląd możliwości tej biblioteki. Książka ta obejmuje szeroką gamę algorytmów i metod przetwarzania obrazu, z których jedynie wybrane fragmenty — jak detekcja krawędzi za pomocą algorytmu Canny’ego oraz analiza barw — zostały zaimplementowane w niniejszym projekcie. Wybór tych technik wynikał z ich efektywności oraz dostosowania do wymagań projektu, jakim jest klasyfikacja obiektów na podstawie koloru.

Przy opracowywaniu metody rozpoznawania kolorów wykorzystano również wyniki badań z publikacji [1], opisującej różnorodne metody i algorytmy analizy wizyjnej, stosowane w procesie klasyfikacji obiektów na podstawie ich cech kolorystycznych. Dzięki znajomości podstaw teoretycznych zaprezentowanych w tym dokumencie, wybór odpowiednich technik przetwarzania obrazu oraz implementacja algorytmu klasyfikacji kolorów przebiegały bardziej efektywnie.

Kolejnym istotnym krokiem była analiza metod detekcji krawędzi, przy czym szczególna uwaga została poświęcona algorytmom i funkcjom dostępnym w bibliotece OpenCV. Jednym z analizowanych dokumentów był artykuł [14], który szczegółowo omawia różne techniki detekcji krawędzi. Po dokonaniu analizy, jako metoda najbardziej odpowiednia dla projektu została wybrana detekcja krawędzi metodą Canny’ego, ze względu na jej skuteczność w kontekście warunków oświetleniowych oraz wysoką precyzję w identyfikacji krawędzi obiektów.

2.3.2 Opracowanie systemu kontroli silników

System kontroli silników dla platformy mobilnej oparto o napęd różnicowy, który wybrano ze względu na jego precyzję oraz stosunkowo prostą implementację algorytmów sterujących. Napęd różnicowy umożliwia manewrowanie robotem poprzez odpowiednie

sterowanie prędkością oraz kierunkiem obrotu poszczególnych kół, co jest szczególnie istotne w zastosowaniach wymagających dokładności na ograniczonej przestrzeni. Przy opracowywaniu systemu sterowania teoretyczną podstawę stanowiły wyniki badań opisane w artykule [8], który szczegółowo omawia technikę regulacji PID (proporcjonalno-całkująco-różniczkującą) w zastosowaniu do robotów mobilnych z napędem różnicowym. Artykuł ten dostarcza kompleksowej wiedzy na temat regulacji PID w robotyce, co pozwoliło na łatwiejsze zaimplementowanie programu do kontroli prędkości i kierunku działania poszczególnych silników.

Istotnym elementem systemu kontroli była również implementacja systemu enkoderów, które monitorują ruch robota i pozwalają na dokładną analizę przebytej drogi oraz prędkości. W projekcie dostępne były dwie metody odczytywania impulsów generowanych przez enkodery: poprzez przerwania generowane przy każdym impulsie oraz przy użyciu wbudowanego licznika mikrokontrolera. Wybór metody wiązał się z koniecznością dokładnego zrozumienia zarówno korzyści, jak i ograniczeń każdej z nich.

Dzięki analizie dokumentacji mikrokontrolera ATMega328 [5] oraz literatury opisującej przerwania [9], możliwe było dogłębne zrozumienie mechanizmów przerwań, co stanowiło podstawę do implementacji wybranego algorytmu. Dokumentacja mikrokontrolera dostarcza szczegółowych informacji na temat możliwości wykorzystywania przerwań i liczników, co pozwala na optymalizację algorytmu kontroli silników oraz dostosowanie jego parametrów do wymogów projektu. Analiza tych źródeł umożliwiła autorowi wybór optymalnego rozwiązania, które zwiększa precyzję pomiaru pozycji i prędkości przy jednoczesnym zachowaniu wydajności systemu.

Rozdział 3

Założenia wstępne

W rozdziale tym przedstawiono opis wymagań funkcjonalnych i нефункциональных, które określają, jakie zadania system musi spełniać oraz jakie kryteria jakościowe są wymagane do zapewnienia prawidłowego działania platformy mobilnej. Omówiono również schemat funkcjonalny aplikacji, opis wykorzystanego sprzętu elektronicznego oraz metodykę pracy i etapy projektu.

3.1 Wymagania funkcjonalne i нефункциональные

Wymagania funkcjonalne odnoszą się do kluczowych funkcji systemu, które muszą zostać spełnione, aby system mógł realizować swoje podstawowe zadania. W przypadku autonomicznej platformy mobilnej do sortowania klocków funkcje te obejmują między innymi:

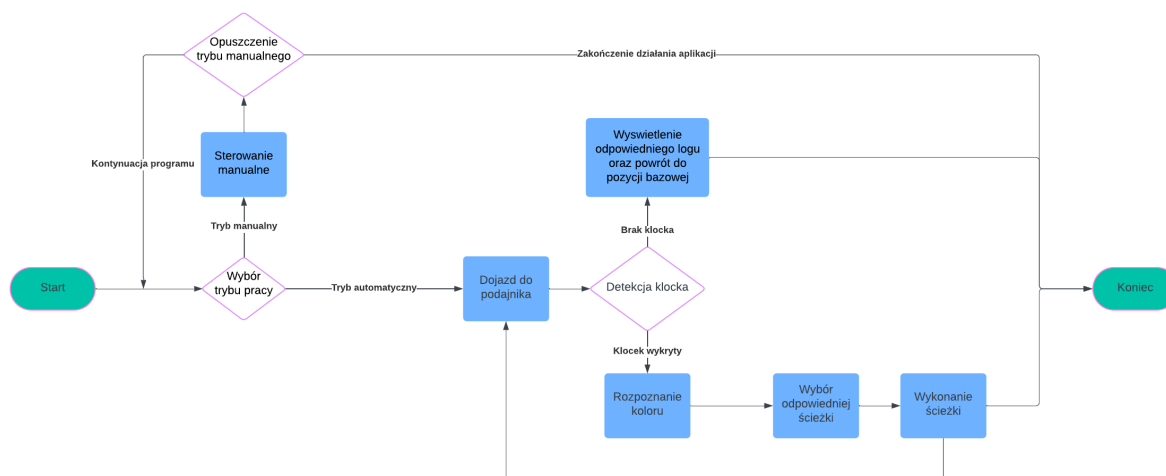
- Rozpoznawanie kolorów klocków – system musi identyfikować kolory klocków przy pomocy kamery i odpowiednich algorytmów przetwarzania obrazu (czerwony, zielony, niebieski).
- Przekierowywanie klocków – po rozpoznaniu koloru, system kieruje klocek do odpowiedniego pojemnika.
- Samodzielne poruszanie się – robot powinien nawigować w wyznaczonej przestrzeni zgodnie z zaprogramowanymi trasami.

Wymagania нефункциональные obejmują kryteria jakościowe, które determinują, jak system ma działać. Są to:

- Niezawodność – system powinien pracować nieprzerwanie przez określony czas bez błędów.
- Szybkość przetwarzania obrazu – analiza obrazu i detekcja koloru muszą odbywać się na bieżąco, co jest szczególnie istotne w przypadku szybkich linii produkcyjnych.

3.2 Przypadki użycia i diagramy UML

Poniższy diagram zawiera schemat funkcjonalny aplikacji. Wyszczególniono na nim najważniejsze funkcjonalności oraz prawdopodobne możliwe zdarzenia. Szczegółowa obsługa błędów nie została wprowadzona ze względu na chęć utrzymania dobrego poziomu czytelności schematu.



Rysunek 3.1: Schemat UML zawierający podstawowe funkcjonalności

3.3 Opis wykorzystanego sprzętu elektronicznego

W projekcie wykorzystano wiele urządzeń elektronicznych w celu zapewnienia wymaganej funkcjonalności. Szczegółowa lista wraz z opisem każdego z urządzeń znajduje się poniżej:

- Raspberry Pi – służy jako główny komputer zarządzający systemem, wyposażony w odpowiednie moduły do obsługi kamery oraz komunikacji.
- Arduino UNO R3 - mikrokontroler pełniący rolę kontrolera napędów oraz enkoderów.
- Cytron MDD10A - sterownik silników odpowiadający za generowanie odpowiednich sygnałów PWM oraz dostarczenie wymaganego napięcia do silników DC.
- Kamera Sony IMX519 16 Mpx - element odpowiadający za akwizycję obrazu oraz umożliwienie wykorzystania wizji komputerowej.
- Silniki DC Pololu 20,4:1 12V HP 25Dx65L - elementy wykonawcze robota mobilnego wyposażone w enkodery magnetyczne kwadraturowe działające na podstawie efektu Hall'a.

- Serwomechanizm DS3218 MG - element wykonawczy, którego zadaniem jest zamykanie oraz otwieranie chwytaka.
- Przetwornica Step-Up/Step-Down 5V S13V30F5 - element konieczny do poprawnego zasilenia układu SBC z pakietu ogniw litowo-jonowych.
- Ogniw litowo-jonowe typu 16850 - Pakiet składający się z czterech ogniw połączonych szeregowo odpowiada za zasilanie robota podczas pracy.
- Płytki stykowe - element umożliwiający wygodne wyprowadzenie pasma zasilającego enkodery oraz serwomechanizm.
- Przewody połączeniowe.

3.4 Metodyka pracy i etapy projektu

Projekt został zrealizowany zgodnie z metodyką iteracyjno-przyrostową, co umożliwiło sukcesywne rozwijanie i weryfikację funkcjonalności na poszczególnych etapach w rzeczywistych warunkach roboczych systemu. Na każdym etapie projektowania, implementacji oraz testowania gromadzono informacje zwrotne na temat działania systemu. Zebrane dane pozwalały na identyfikację obszarów wymagających poprawy, co skutkowało dostosowaniem parametrów systemu lub optymalizacją kodu w celu zapewnienia optymalnej wydajności.

Prace rozpoczęto od przygotowania środowiska programistycznego oraz poprawnego połączenia kamery z mikrokontrolerem Raspberry Pi. Na tym etapie skupiono się na konfiguracji narzędzi do programowania i przetestowaniu prawidłowego działania kamery. Był to kluczowy element, gdyż precyzyjny odczyt obrazu miał wpływ na późniejsze funkcje rozpoznawania kształtów i kolorów.

Kolejnym krokiem było podłączenie elementów wykonawczych oraz implementacja aplikacji kontrolującej silniki i enkodery, które stanowiły podstawę sterowania ruchem robota. Po wdrożeniu sterowników przeprowadzono wstępne testy napędu różnicowego, analizując dokładność przemieszczenia oraz reagowanie systemu na sygnały z enkoderów.

Następnie wykonano prototyp podwozia robota, aby sprawdzić działanie algorytmów sterowania w kontekście rzeczywistych warunków pracy. Użycie prototypu pozwoliło ocenić efektywność sterowania, jak i stabilność ruchu, co przyczyniło się do dokonania wcześniejszych poprawek w konstrukcji oraz kodzie. Po przeprowadzeniu testów prototypowych zaprojektowano właściwe podwozie i wydrukowano je w technologii druku 3D, umożliwiając integrację z wcześniej skonfigurowanymi komponentami elektronicznymi i mechanicznymi.

W kolejnych iteracjach skupiono się na dalszym testowaniu i optymalizacji programu kontrolera silników, co pozwoliło uzyskać wymaganą precyzję i stabilność pracy napędu.

Po uzyskaniu satysfakcjonujących rezultatów w zakresie sterowania przystąpiono do projektowania górnej części obudowy robota, która miała za zadanie stabilne zamocowanie kamery oraz częściowe osłonięcie komponentów wewnętrznych.

Ostatecznym etapem była pełna integracja wszystkich komponentów systemu oraz finalne testy funkcjonalne. Testy obejmowały analizę wszystkich kluczowych funkcji robota w warunkach rzeczywistych, umożliwiając ocenę pracy systemu jako całości oraz identyfikację ostatnich potencjalnych usprawnień. Cały proces zakończył się uzyskaniem gotowego rozwiązania, które spełniało założenia projektowe i osiągało zamierzoną funkcjonalność.

Rozdział 4

Podstawy teoretyczne

W niniejszym rozdziale przedstawiono kluczowe zagadnienia teoretyczne, które stanowią fundament techniczny realizowanego projektu. Przedstawienie podstaw działania zastosowanych elementów pozwoli lepiej zrozumieć wyzwania projektowe oraz przyjęte metody rozwiązywania problemów związanych ze sterowaniem i nawigacją mobilnej platformy robotycznej.

Omówione zostaną najważniejsze zasady funkcjonowania enkoderów kwadraturowych, które umożliwiają monitorowanie ruchu - podstawowej odometrii. Zaprezentowane zostaną także zasady działania regulatora PID, powszechnie stosowanego w układach sterowania.

W dalszej części omówiono wybrane algorytmy z zakresu wizji komputerowej, takie jak detekcja krawędzi i rozpoznawanie kolorów, które stanowią istotne elementy systemu wizyjnego robota.

4.1 Enkodery

Enkodery są urządzeniami pomiarowymi, które umożliwiają śledzenie ruchu obrotowego lub liniowego. W zastosowaniach robotycznych, takich jak opisywana platforma mobilna, enkodery umożliwiają śledzenie prędkości i pozycji, co jest kluczowe dla poprawnej implementacji odometrii – procesu śledzenia pozycji robota w przestrzeni. Enkodery dzielą się na enkodery optyczne, magnetyczne i mechaniczne, w zależności od zastosowanej technologii detekcji. W omawianym projekcie szczególne zastosowanie znajdują enkodery kwadraturowe, montowane przy silnikach odpowiedzialnych za napęd różnicowy robota.

4.1.1 Rodzaje enkoderów

Enkodery klasyfikowane są na podstawie zasady ich działania, co wpływa na precyzję pomiarów oraz możliwości aplikacyjne. Główne typy enkoderów to:

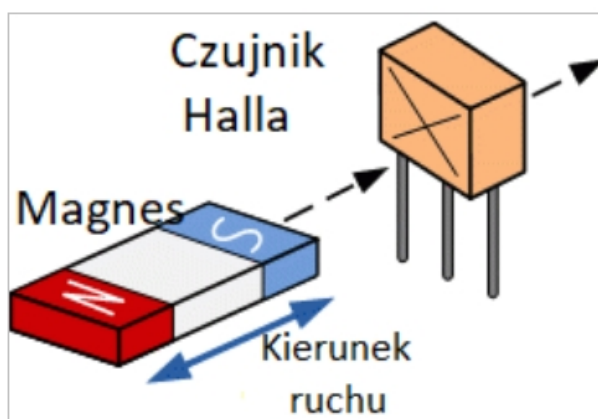
- **Enkodery optyczne** - działają na zasadzie przerywania wiązki światła przez tarczę

z otworami, generując impulsy odpowiadające ruchowi. Ten typ enkoderów charakteryzuje się dużą precyzją, co czyni je popularnym wyborem w robotyce.

- **Enkodery magnetyczne** - działają w oparciu o zmiany pola magnetycznego, zwykle generowane przez magnes przymocowany do osi. Zastosowanie efektu Hall'a w tych enkoderach pozwala na wykrywanie zmian w polu magnetycznym, co jest kluczowe w enkoderach kwadraturowych. Są wytrzymałe i odporne na kurz czy wibracje, przez co dobrze sprawdzają się w trudnych warunkach.
- **Enkodery mechaniczne** - najprostszy rodzaj enkoderów, wykorzystujący mechaniczne styki do generowania impulsów. Ze względu na ograniczoną trwałość są rzadziej stosowane w precyzyjnych aplikacjach.

Efekt Hall'a

Efekt Hall'a stanowi podstawę działania magnetycznych enkoderów kwadraturowych, a jego działanie opiera się na generowaniu napięcia (tzw. napięcia Hall'a) prostopadłego do kierunku przepływu prądu i linii pola magnetycznego. Gdy magnes jest umieszczony w pobliżu sensora Hall'a, zmiany pola magnetycznego wywołują impulsy wykrywane przez enkoder. W przypadku enkoderów kwadraturowych dwa czujniki Hall'a rozmieszczone pod kątem 90° umożliwiają rozróżnienie kierunku obrotu wału oraz precyzyjny pomiar przebytej odległości. Zastosowanie efektu Hall'a w enkoderach kwadraturowych daje możliwość tworzenia solidnych, odpornych na zużycie i precyzyjnych pomiarów.

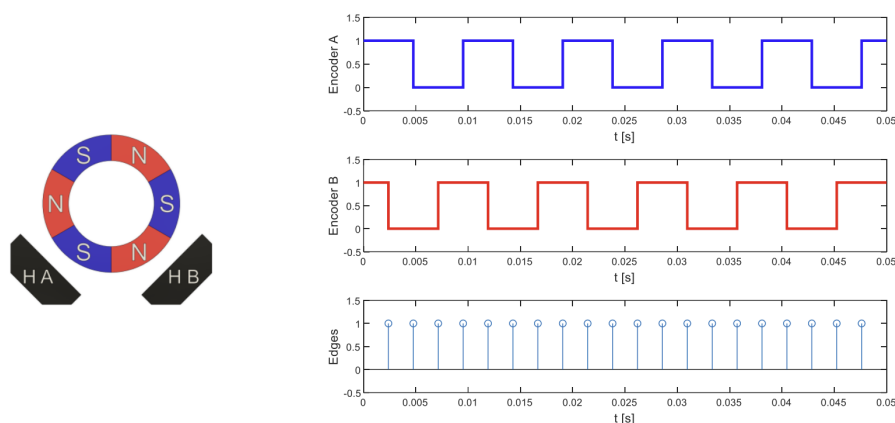


Rysunek 4.1: Rysunek przedstawiający zasadę działania czujnika Hall'a [6]

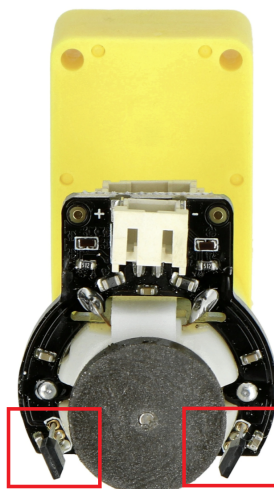
4.1.2 Enkodery kwadraturowe

Enkodery kwadraturowe to specjalny rodzaj enkoderów, które umożliwiają nie tylko pomiar prędkości i położenia, ale także kierunku obrotu. Składają się z dwóch sygnałów wyjściowych – A i B – przesuniętych względem siebie o fazę 90° . Dzięki temu fazowemu

przesunięciu system sterujący jest w stanie określić kierunek obrotu, analizując, który z sygnałów A lub B wyprzedza drugi. Każdy impuls generowany przez sygnały A i B odpowiada określonemu przemieszczeniu kątowemu, co pozwala na wyznaczenie pozycji kątowej osi obrotu z wysoką precyzją.



Rysunek 4.2: Rysunek przedstawiający ideowo budowę enkodera kwadraturowego oraz sygnałów przez niego generowanych [2]

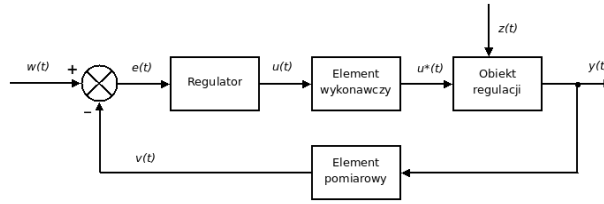


Rysunek 4.3: Rysunek przedstawiający przykładowy wygląd enkodera opartego o efekt Hall'a. Czerwonymi kwadratami są zaznaczone tranzystory wykrywające zmianę pola magnetycznego podczas obrotu wału [3]

4.2 Regulator PID

Regulator PID (ang. *Proportional-Integral-Derivative*) jest jednym z najczęściej stosowanych regulatorów w systemach automatycznego sterowania, ze względu na swoją prostotę implementacji oraz skuteczność w wielu aplikacjach przemysłowych i robotycznych. Regulator ten wykorzystuje trzy składniki: proporcjonalny, całkujący oraz różniczkujący.

4.2.1 Ogólny schemat układu regulacji



Rysunek 4.4: Rysunek przedstawiający ogólny schemat układu regulacji automatycznej [13]

4.2.2 Działanie regulatora PID

Działanie regulatora PID opiera się na przetwarzaniu sygnału uchybu $e(t)$, czyli różnicy między wartością zadaną y_{set} a wartością rzeczywistą $y(t)$ wielkości regulowanej:

$$e(t) = y_{set} - y(t) \quad (4.1)$$

Regulator PID wyznacza wartość sygnału sterującego $u(t)$ jako sumę trzech składników:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} \quad (4.2)$$

gdzie:

- K_p – współczynnik proporcjonalny, który wzmacnia aktualny uchyb,
- K_i – współczynnik całkujący, który reaguje na sumę uchybów w czasie,
- K_d – współczynnik różniczkujący, który reaguje na zmianę uchybu.

Każda ze składowych regulatora PID pełni inną funkcję, a ich kombinacja pozwala na precyzyjne kontrolowanie zachowania systemu.

4.2.3 Składnik proporcjonalny P

Składnik proporcjonalny jest liniowo zależny od uchybu $e(t)$ i odpowiada za natychmiastową reakcję na różnicę między wartością zadaną a rzeczywistą. Działa zgodnie ze wzorem:

$$P(t) = K_p e(t) \quad (4.3)$$

Im większa wartość K_p , tym regulator bardziej dynamicznie reaguje na uchyb, co może przyspieszyć osiągnięcie wartości zadanej, ale przy zbyt dużych wartościach może prowadzić do przeregulowania lub oscylacji.

4.2.4 Składnik całkujący I

Składnik całkujący sumuje uchyby w czasie, co pozwala na eliminację uchybu statycznego – różnicy między wartością rzeczywistą a zadaną, która może utrzymywać się w przypadku stosowania tylko składnika proporcjonalnego. Składnik całkujący jest dany równaniem:

$$I(t) = K_i \int_0^t e(\tau) d\tau \quad (4.4)$$

Dzięki temu składnikowi, regulator PID potrafi zniwelować długotrwały uchyb, chociaż przy zbyt dużych wartościach K_i może wystąpić zjawisko przeregulowania i niestabilność.

4.2.5 Składnik różniczkujący D

Składnik różniczkujący uwzględnia szybkość zmian uchybu $e(t)$, co pozwala na szybsze reagowanie na gwałtowne zmiany i przeciwdziałanie efektowi przeregulowania. Opisuje go wyrażenie:

$$D(t) = K_d \frac{de(t)}{dt} \quad (4.5)$$

Ten składnik regulatora poprawia stabilność i tłumi oscylacje, jednak zbyt duża wartość K_d może prowadzić do niestabilności układu, zwłaszcza w obecności szumów.

4.2.6 Pełne równanie regulatora PID

Łącząc wszystkie trzy składniki, równanie regulatora PID przyjmuje postać:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} \quad (4.6)$$

gdzie:

- $K_p e(t)$ – składnik proporcjonalny, który odpowiada za szybką reakcję na zmiany,
- $K_i \int_0^t e(\tau) d\tau$ – składnik całkujący, który eliminuje uchyb statyczny,
- $K_d \frac{de(t)}{dt}$ – składnik różniczkujący, który przeciwdziała nagłym zmianom uchybu.

Regulacja parametrów K_p , K_i i K_d pozwala na dostosowanie charakterystyki regulatora PID do specyfiki sterowanego obiektu. Optymalne wartości tych parametrów zazwyczaj są dobierane metodą prób i błędów lub za pomocą metod takich jak strojenie Zieglera-Nicholsa.

4.2.7 Zastosowanie regulatora PID w sterowaniu robotem mobilnym

W omawianym projekcie regulator PID znajduje zastosowanie w kontrolowaniu odległości jaką pokonuje platforma mobilna poprzez odpowiednie sterowanie napięciem zasilania elementów wykonawczych. Nastawy regulatora zostały dostosowane metodą prób i błędów z racji trudnego do zamodelowania układu dynamicznego. Uzyskana jakość regulacji jest wystarczająca do spełnienia wymagań projektowych.

4.3 Podstawy wykrywania krawędzi w wizji komputerowej

Wykrywanie krawędzi stanowi kluczowy element analizy obrazów w kontekście wizji komputerowej. Jest to technika mająca na celu identyfikację istotnych zmian w intensywności pikseli obrazu, które zazwyczaj wskazują na obecność krawędzi obiektów. Krawędzie odgrywają fundamentalną rolę w percepcji kształtów i struktur, dlatego ich wykrywanie jest podstawą wielu aplikacji, takich jak segmentacja obrazu, rozpoznawanie obiektów oraz analiza ruchu.

Jedną z najbardziej popularnych metod wykrywania krawędzi jest algorytm Canny’ego, który został zaproponowany przez Johna F. Canny’ego w 1986 roku. Metoda ta jest cenniejsza za swoją zdolność do wydobywania krawędzi z obrazów przy jednoczesnym zachowaniu ich jakości oraz redukcji szumów.

4.3.1 Algorytm Canny’ego

Algorytm Canny’ego składa się z kilku kluczowych kroków:

1. **Wygładzenie** (ang. *Smoothing*): Na początku obraz poddawany jest filtracji w celu zredukowania szumów. Zazwyczaj stosuje się filtr Gaussa, który jest reprezentowany przez macierz:

$$G = \begin{bmatrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{bmatrix}$$

2. **Obliczenie gradientu**: Następnie obliczane są gradienty obrazu, aby zidentyfikować kierunki zmian intensywności. Używa się do tego macierzy *Sobela*, które pozwalają na obliczenie gradientu w kierunkach poziomym (G_x) i pionowym (G_y):

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Gradient można obliczyć jako:

$$G = \sqrt{G_x^2 + G_y^2}$$

oraz jego kierunek:

$$\Theta = \text{atan2}(G_y, G_x)$$

3. Tłumienie nie-maksymalne (ang. *Non-maximum suppression*): W kolejnym kroku następuje eliminacja pikseli, które nie są lokalnymi maksimumami w kierunku gradientu, co pozwala na wyostrenie krawędzi.

4. Podwójne progowanie (ang. *Double thresholding*): Algorytm wprowadza dwa progi: wysoki i niski. Piksele, które przekraczają wysoki próg, są uznawane za krawędzie, natomiast te, które są poniżej niskiego progu, są odrzucane. Piksele, które mieszczą się między tymi dwoma progami, są klasyfikowane jako krawędzie słabe.

5. Śledzenie krawędzi z histerezą (ang. *Edge tracking by hysteresis*): Ostatnim krokiem jest śledzenie krawędzi, gdzie słabe krawędzie są zachowywane, jeśli są połączone z mocnymi krawędziami. To pozwala na eliminację przypadkowych detekcji krawędzi i zapewnia spójność wyników.

4.4 Podstawy rozpoznawania kolorów w wizji komputerowej

Rozpoznawanie kolorów w wizji komputerowej to kluczowy aspekt w wielu aplikacjach automatyki, robotyki oraz sztucznej inteligencji. W kontekście robotyki, techniki te pozwalają na identyfikację i klasyfikację obiektów w otoczeniu robota, co jest istotne dla jego nawigacji oraz interakcji z otoczeniem. W niniejszej sekcji omówione zostaną podstawowe zasady oraz techniki wykorzystywane w procesie rozpoznawania kolorów.

4.4.1 Teoria koloru

Kolor jest percepcją widzialnego spektrum światła, które jest odbierane przez ludzkie oczy i interpretowane przez mózg. W wizji komputerowej kolory są często reprezentowane za pomocą różnych modeli kolorów, w tym w przestrzeni kolorów RGB (ang. *Red, Green, Blue*) rys. [4.5], HSV (ang. *Hue, Saturation, Value*) rys. [4.6] oraz YCrCb (ang. *Luminance, Chrominance*) rys. [4.7]. W modelu RGB kolory są definiowane przez intensywności

trzech składowych kolorów, podczas gdy w modelu HSV kolory są określane na podstawie ich odcienia, nasycenia i jasności, co ułatwia przetwarzanie i rozpoznawanie kolorów w kontekście wizji komputerowej. Model YCrCb, z kolei, oddziela informację o luminancji (jasności) od informacji o chrominancji (kolorze), co jest szczególnie przydatne w kompresji obrazów oraz w systemach wizyjnych, umożliwiając efektywną analizę kolorów przy jednoczesnym zredukowaniu wpływu zmian oświetleniowych.

4.4.2 Proces rozpoznawania kolorów

Proces rozpoznawania kolorów w obrazach polega na identyfikacji oraz klasyfikacji pikseli na podstawie ich wartości kolorystycznych. Kluczowe etapy tego procesu obejmują:

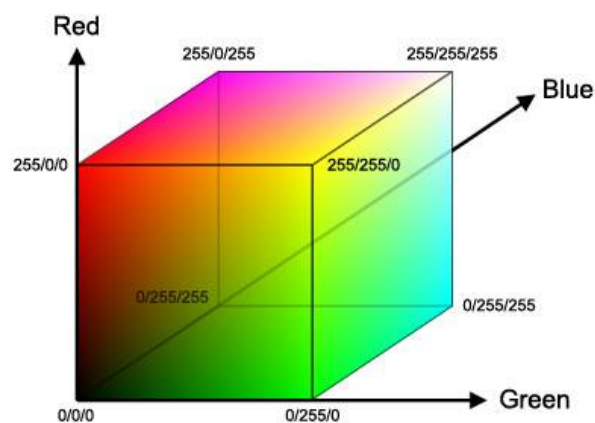
1. **Przechwytywanie obrazu:** Obraz jest przechwytywany za pomocą kamery cyfrowej, która generuje dane w formacie cyfrowym.
2. **Przetwarzanie obrazu:** Na tym etapie obraz jest poddawany różnym operacjom, takim jak filtracja, segmentacja i transformacja kolorów, aby wyodrębnić istotne informacje.
3. **Ekstrakcja cech:** Następnie przeprowadzana jest analiza danych kolorystycznych w celu identyfikacji i klasyfikacji obiektów na podstawie ich kolorów.
4. **Decyzja:** Na końcu system podejmuje decyzje o klasyfikacji obiektów bazując na wcześniej zebranych informacjach o kolorze.

4.4.3 Algorytmy rozpoznawania kolorów

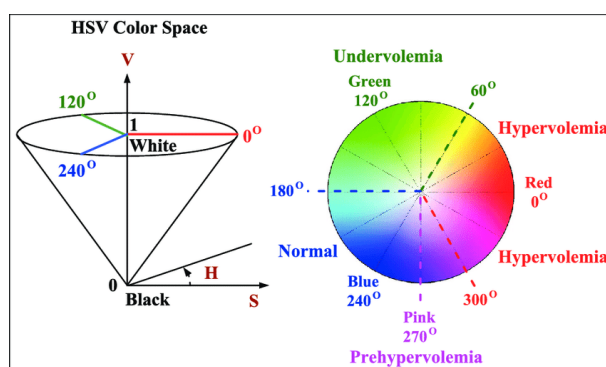
W literaturze przedmiotu opisano wiele algorytmów wykorzystywanych w rozpoznawaniu kolorów, w tym metody oparte na histogramie, segmentacji obrazu oraz analizie cech. Histogramy kolorów są popularnym narzędziem do analizy rozkładu kolorów w obrazie, co pozwala na szybką identyfikację dominujących kolorów.

Segmentacja obrazu to proces dzielenia obrazu na różne regiony na podstawie ich kolorów, co ułatwia identyfikację obiektów w danym kolorze. Algorytmy takie jak k-means clustering czy metoda watershed są często stosowane w celu wydzielenia obiektów o określonym kolorze z tła.

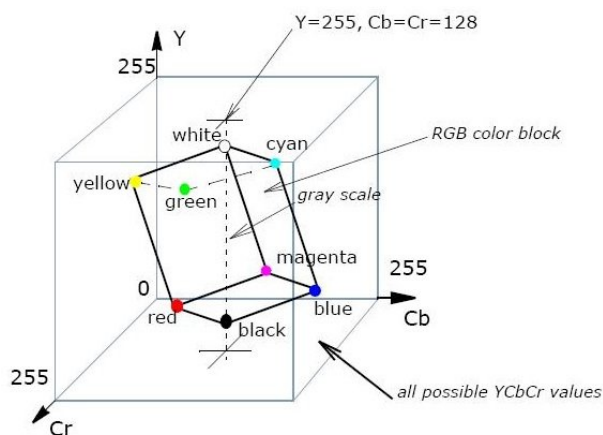
DO POPRAWY - TRZEBA DO JESZCZE SENSOWNIE OPISAĆ (!!!!)



Rysunek 4.5: Rysunek przedstawiający wizualizację przestrzeni kolorów RGB [10]



Rysunek 4.6: Rysunek przedstawiający wizualizację przestrzeni HSV [11]



Rysunek 4.7: Rysunek przedstawiający wizualizację przestrzeni YCrCb [12]

Rozdział 5

Wymagania i specyfikacja użytkowa

W poniższym rozdziale szczegółowo opisano wymagania sprzętowe i programowe, sposób instalacji, aktywacji oraz wszystkie inne informacje konieczne dla użytkownika, które są niezbędne do skutecznego działania platformy mobilnej.

5.1 Wymagania sprzętowe i programowe

Podstawowym założeniem było wykorzystanie mikrokomputera Raspberry Pi, który stanowi centralny element systemu. Ze względu na jego kompaktowe rozmiary, niską wagę oraz odpowiednią moc obliczeniową, Raspberry Pi idealnie wpisuje się w potrzeby projektu.

Wymagania sprzętowe

- Mikrokomputer Raspberry Pi – wybrany model Raspberry Pi 4 Model B, który charakteryzuje się czterordzeniowym procesorem, co zapewnia wystarczającą moc obliczeniową do przetwarzania obrazu oraz zarządzania algorytmami sterowania.
- Kamera – zastosowano dedykowaną kamerę kompatybilną z Raspberry Pi, która umożliwia przechwytywanie obrazów w wysokiej rozdzielczości. Kamera jest podłączona przez interfejs CSI (ang. *Camera Serial Interface*), co zapewnia niskie opóźnienia i wysoką jakość obrazu.
- Silniki i napęd różnicowy – do napędu platformy wykorzystano silniki prądu stałego z enkoderami, które umożliwiają precyzyjne sterowanie ruchem robota oraz monitorowanie jego pozycji.
- Zasilanie – system zasilany jest z pakietu czterech ogniw litowo-jonowych, dostarczających wysatraczącą ilość napięcia oraz umożliwiającą pracę przez odpowiedni czas.

- Chwytnik - wykorzystano chwytak zasilany jednym serwomechanizmem, wystarczający do realizacji zadań.

Wymagania programowe

- Python - język programowania bardzo dobrze sprawdzający się przy wykorzystaniu komputera Raspberry Pi.
- OpenCV - biblioteka umożliwiająca bardzo dużą ilość operacji na obrazie cyfrowym.
- Libcamera2 - biblioteka konieczna do poprawnego wykorzystywania kamery wraz z OpenCV na komputerze Raspberry Pi.
- Raspbian OS - system operacyjny, będący specjalną implementacją systemu Linux, posiadającą wiele wbudowanych bibliotek oraz innych udogodnień wspomagających rozwój oprogramowania na komputerze Raspberry Pi.
- git - system kontroli wersji, wykorzystany w celu spełnienia standardów rozwoju oprogramowania.

5.2 Sposób instalacji

5.2.1 Instalacja systemu operacyjnego

Pierwszym krokiem w procesie instalacji systemu kontroli robota jest zainstalowanie systemu operacyjnego Raspbian na jednostce centralnej mikrokomputera Raspberry Pi. W celu skutecznej realizacji tego zadania zaleca się skorzystanie z oprogramowania dostarczanego przez firmę Raspberry Pi Ltd, które umożliwia utworzenie przenośnego dysku uruchomieniowego. Dysk ten może przyjąć postać karty microSD lub przenośnego dysku USB. Istnieje również szereg innych programów, które mogą spełniać tę funkcję; jednakże najbardziej rekomendowanym rozwiązaniem jest użycie aplikacji *Raspberry Pi Imager*. Program ten, oprócz możliwości utworzenia dysku startowego, oferuje opcję pobrania odpowiedniej wersji systemu Raspbian lub innego dystrybucji systemu Linux, co czyni go bardzo elastycznym narzędziem, dopasowanym do potrzeb użytkownika.

Zaleca się korzystanie z 64-bitowej wersji systemu Raspbian, zapewnia to lepszą wydajność i większą kompatybilność z aplikacjami.

Po utworzeniu nośnika rozruchowego, należy umieścić kartę microSD lub przenośny dysk USB w odpowiednim gnieździe mikrokomputera Raspberry Pi. Następnie, po podłączeniu urządzenia do źródła zasilania, rozpocznie się proces instalacji systemu operacyjnego. W przypadku, gdy obraz nie jest wyświetlany po podłączeniu mikrokomputera do monitora za pomocą kabla microHDMI, użytkownik powinien zweryfikować ustawienia pliku konfiguracyjnego *config.txt* znajdującego się na nośniku startowym. W szczególności,

aby dostosować rozdzielczość wyświetlacza, należy dodać lub zmodyfikować odpowiednie linie w tym pliku. Przykładowy fragment kodu, który można umieścić w pliku *config.txt*, aby ustawić rozdzielczość na 1920x1080, wygląda następująco:

```
1     hdmi_group=1
2     hdmi_mode=16
```

Powyższe parametry konfiguracyjne odpowiadają za wybór standardu HDMI oraz określenie trybu rozdzielczości, gdzie wartość 16 oznacza rozdzielczość 1080p. Po wprowadzeniu zmian w pliku konfiguracyjnym, należy zapisać plik i ponownie uruchomić mikrokomputer.

W przypadku problemów z wyświetlaniem obrazu warto również przetestować inny monitor, jeśli jest on dostępny, ponieważ istnieje możliwość, że dany model monitora nie obsługuje określonego formatu sygnału HDMI.

Po instalacji systemu operacyjnego Raspbian na urządzeniu Raspberry Pi, kolejnym istotnym etapem jest instalacja wymaganych pakietów oraz bibliotek, które zapewnią funkcjonalność kluczowych elementów systemu sterowania robota. Wśród podstawowych bibliotek, niezbędnych do poprawnego działania systemu, znajdują się: Python, OpenCV, Libcamera2 oraz RPi.GPIO.

Wszystkie poniższe komendy należy wykonać w terminalu Raspberry Pi, aby pobrać i zainstalować odpowiednie pakiety. Warto zauważyć, że przed przystąpieniem do instalacji zaleca się aktualizację pakietów systemowych.

5.2.2 Aktualizacja systemu

Dobłą praktyką, umożliwiającą uniknięcie wielu błędów wynikających z braku kompatybilności systemu bazującego na jądrze Linux jest wykonywanie aktualizacji oraz uaktualnienia zainstalowanych pakietów poprzez zdalne repozytorium *apt* wykonując poniższe komendy.

```
1 sudo apt update
2 sudo apt upgrade -y
```

5.2.3 Instalacja Python

Python jest podstawowym językiem programowania, w którym napisana została aplikacja sterująca robotem. Raspbian posiada zazwyczaj zainstalowaną wersję Python, jednak można zainstalować lub zaktualizować go do nowszej lub wymaganej wersji przy użyciu następującej komendy. Niektóre biblioteki lub pakiety wymagają konkretnej wersji Python, na przykład w wersji 3.10, podczas gdy najnowsza wykracza poza wersję 3.12. W

związku z powyższym możliwe jest zaistnienie problemu kompatybilności, przez co zaleca się zweryfikowanie, która wersja będzie odpowiednia. W przypadku poniższego projektu wersja 3.10 będzie odpowiednia.

```
1 sudo apt install -y python3 python3-pip
```

5.2.4 Instalacja OpenCV

OpenCV (ang. *Open Source Computer Vision Library*) to biblioteka służąca do przetwarzania obrazu i analizy wizyjnej. W projekcie OpenCV zostanie wykorzystane m.in. do detekcji koloru klocków. Instalacja OpenCV jest możliwa za pomocą menedżera pakietów pip:

```
1 pip3 install opencv-python-headless
```

5.2.5 Instalacja Libcamera2

Biblioteka *Libcamera2* jest odpowiedzialna za obsługę kamery, która jest podłączona do złącza CSI Raspberry Pi. Aby zainstalować pakiet Libcamera2, należy wykonać następujące polecenie (więcej informacji można znaleźć w dokumentacji[[7]]):

```
1 sudo apt install -y python3-picamera2
```

5.2.6 Instalacja RPi.GPIO

Biblioteka *RPi.GPIO* zapewnia interfejs programistyczny do korzystania z pinów GPIO (ang. *General Purpose Input/Output*) mikrokomputera Raspberry Pi. Zalecana jest instalacja poprzez repozytorium *apt* zamiast *pip*, przez wzgląd na możliwe problemy z kompatybilnością.

```
1 sudo apt-get update
2 sudo apt-get -y install python-rpi.gpio
```

Po zakończeniu instalacji wszystkich wymienionych pakietów i bibliotek, system Raspberry Pi jest gotowy do uruchomienia aplikacji kontrolującej robota. Przed przystąpieniem do pracy zaleca się również zweryfikowanie poszczególnych bibliotek.

5.2.7 Weryfikacja instalacji pakietów

Po zakończeniu procesu instalacji pakietów i bibliotek warto przeprowadzić weryfikację, aby upewnić się, że wszystkie komponenty zostały zainstalowane poprawnie i są

gotowe do użycia. Testowanie każdego pakietu z osobna pozwala na szybkie wykrycie ewentualnych problemów i zapewnia, że środowisko pracy jest w pełni skonfigurowane.

Weryfikacja instalacji Python

Aby sprawdzić, czy Python jest poprawnie zainstalowany, w terminalu wpisz:

```
1 python3 --version
```

Komenda ta powinna wyświetlić zainstalowaną wersję Python, np. `Python 3.x.x`. Jeśli polecenie działa bez błędów, oznacza to, że Python jest zainstalowany poprawnie.

Weryfikacja instalacji OpenCV

Aby upewnić się, że biblioteka OpenCV jest dostępna, można przeprowadzić test bezpośrednio w Python. Uruchom interpreter Python komendą:

```
1 python3
```

Następnie zaimportuj bibliotekę OpenCV, wpisując:

```
1 import cv2
2 print(cv2.__version__)
```

Polecenie to wyświetli wersję OpenCV, np. `4.5.x`. Jeśli import przebiegnie bez błędów, oznacza to, że biblioteka OpenCV działa poprawnie.

Weryfikacja instalacji Libcamera2

Aby sprawdzić poprawność instalacji biblioteki Libcamera2 oraz połączenie kamery, wykonaj testowe przechwycenie obrazu za pomocą poniższej komendy:

```
1 libcamera-still -o test_image.jpg
```

Jeżeli kamera działa poprawnie, komenda ta zapisze zdjęcie o nazwie `test_image.jpg` w katalogu domyślnym. Otwarcie pliku i weryfikacja jego zawartości potwierdzi działanie kamery i poprawność instalacji.

5.2.8 Sposób aktywacji

Proces aktywacji systemu sterowania robotem jest stosunkowo prosty, zakładając poprawną instalację wszystkich wymaganych bibliotek i pakietów. Należy jednak przed przystąpieniem do aktywacji zweryfikować, czy wszystkie wymagane komponenty zostały poprawnie zainstalowane, aby uniknąć potencjalnych problemów.

Po upewnieniu się, że wszystkie pakiety są zainstalowane poprawnie, można przystąpić do pobrania repozytorium zawierającego kod aplikacji. Aby to wykonać, należy w terminalu wprowadzić poniższe polecenie:

```
1 git clone https://github.com/JakubPajak/robot-control-system
    .git
```

Po udanym sklonowaniu repozytorium należy przejść do katalogu z pobranymi plikami, aby uzyskać dostęp do aplikacji:

```
1 cd current_v2/control_system/
```

Następnie uruchomić główny plik aplikacji:

```
1 python3 main.py
```

Po wykonaniu powyższych kroków aplikacja uruchomi się w trybie terminalowym. Planowane jest, aby przyszłe wersje aplikacji zostały dostosowane do uruchamiania w przeglądarce, poprawi jakość użytkowania oraz umożliwi wprowadzenie dodatkowych funkcji.

Po uruchomieniu aplikacji wyświetli się menu wyboru trybu pracy, w którym użytkownik może wybrać tryb odpowiedni dla swoich potrzeb. Domyślnym trybem pracy robota jest tryb automatyczny, który nie wymaga ingerencji operatora. Inne opcje umożliwiają ręczną weryfikację działania systemów, takich jak system wizji komputerowej czy tryb sterowania manualnego.

5.2.9 Kategorie użytkowników

Obecna wersja systemu sterowania robotem nie różnicuje ról użytkowników, co oznacza, że każda osoba mająca dostęp do terminala może swobodnie obsługiwać robota. W planowanych przyszłych wersjach aplikacji, szczególnie tych działających w przeglądarce, zostanie wprowadzona autoryzacja użytkowników. Dzięki niej system będzie posiadał bazę danych użytkowników oraz przypisane im role, co zapewni lepsze zarządzanie dostępem zgodnie z kompetencjami użytkowników.

5.2.10 Sposób obsługi

Aby rozpocząć pracę z robotem, należy ustawić go w pozycji startowej, zamieszczonej w załączniku [DODAC ZAŁĄCZNIK] oraz uruchomić aplikację zgodnie z instrukcją zawartą w sekcji *Sposób aktywacji*. Robot automatycznie rozpocznie wykonywanie zadań w trybie autonomicznym, co oznacza, że obsługa użytkownika ograniczy się jedynie do monitorowania pracy robota i ewentualnej korekty ścieżki za pomocą dostępnych opcji sterowania manualnego.

5.2.11 Kwestie bezpieczeństwa

W związku z tym, że robot wykorzystuje komponenty mechaniczne oraz podzespoły elektroniczne, przestrzeganie zasad bezpieczeństwa jest kluczowe dla ochrony użytkownika

i urządzenia. Przed uruchomieniem robota należy upewnić się, że w jego otoczeniu nie znajdują się żadne przeszkody lub przedmioty, które mogłyby zakłócić jego pracę. W trakcie działania należy unikać bezpośredniego kontaktu z ruchomymi częściami robota. W przypadku ręcznej kontroli urządzenia sugeruje się korzystanie z trybu terminalowego lub przeglądarkowego, który zminimalizuje ryzyko kontaktu z urządzeniem.

5.2.12 Przykład działania

Aby zapoznać się z przykładowym działaniem robotycznego systemu sortowania, użytkownik może obejrzeć nagrania zamieszczone na dołączonym nośniku CD. Nagrania ilustrują funkcjonowanie systemu podczas sortowania elementów według ustalonych kryteriów i przedstawiają interakcje pomiędzy poszczególnymi modułami.

5.2.13 Scenariusze korzystania z systemu

System sortowania został zaprojektowany z myślą o szerokim zastosowaniu w różnych gałęziach przemysłu, takich jak produkcja i zarządzanie powierzchniami magazynowymi. Dzięki elastycznej budowie robot może zostać rozbudowany o dodatkowe funkcje, które umożliwią wykorzystanie go w grupach autonomicznych robotów. Tego rodzaju rozwiązania mogą znaleźć zastosowanie w zadaniach wymagających pełnej autonomii, jak na przykład transport materiałów w rozbudowanych konfiguracjach magazynowych i produkcyjnych, zwiększając efektywność i automatyzację procesów.

Rozdział 6

Specyfikacja techniczna

6.1 Konstrukcja

Proces konstrukcji robota przebiegał w dwóch głównych etapach. Pierwszy z nich miał na celu przygotowanie prototypowej wersji robota, która spełnia podstawowe wymagania wynikające z założeń konstrukcyjnych dotyczących napędu oraz sterowania. Drugi etap opiewał na wykonanie pełnego projektu robota w programie Autodesk Fusion 360, spełniającego wszystkie techniczne wymagania projektu, z bardzo dobrą dokładnością, w celu wyeliminowania błędów płynących z konstrukcji mechanicznej.

W fazie prototypowej zdecydowano się na wykorzystanie sklejki brzoźowej – materiału taniego, lekkiego oraz łatwego w obróbce. Platforma bazowa prototypu miała wymiary 240 mm długości oraz 220 mm szerokości, co pozwalało na osadzenie silników w osi centralnej zgodnie z przyjętym projektem napędu. Taka konfiguracja umożliwiła szybką weryfikację działania elementów wykonawczych oraz systemu sterowania, jeszcze przed przystąpieniem do bardziej zaawansowanych prac konstrukcyjnych.

W toku prac nad prototypem, po osiągnięciu zadowalającej precyzji sterowania, do konstrukcji dołączono przednią ścianę, do której przymocowano chwytak. Umożliwiło to przetestowanie poprawności działania wszystkich elementów wykonawczych obecnych na platformie mobilnej. Dodatkowo, równocześnie przeprowadzono testy pełnej komunikacji między kontrolerem silników a komputerem Raspberry Pi, który pełnił rolę głównej jednostki obliczeniowej robota. W tym kroku zweryfikowano, czy platforma poprawnie reaguje na przesyłane komendy oraz czy nie występują żadne błędy w komunikacji lub wykonaniu poleceń.

Po pozytywnym zakończeniu walidacji funkcjonowania algorytmów sterowania i komunikacji, przystąpiono do drugiego etapu konstrukcji – zaprojektowania i wytworzenia docelowej obudowy bazowej robota. Obudowa została wykonana przy użyciu technologii druku 3D, co pozwoliło na uzyskanie wysokiej precyzji oraz wprowadzenie niezbędnych modyfikacji konstrukcyjnych w stosunku do prototypu. Ze względu na ograniczenia prze-

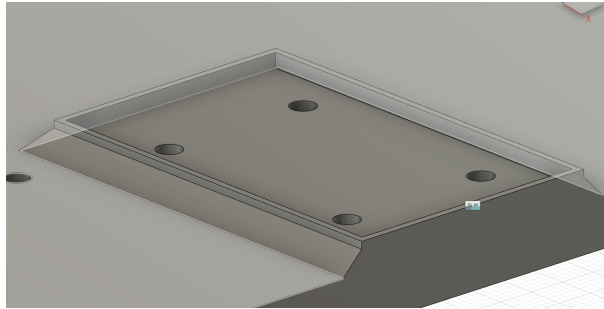
strzeni roboczej urządzenia drukującego, wymiary podstawy platformy zostały zmienione na plan kwadratu o boku 220 mm. Finalny projekt podstawy, przygotowany w programie Autodesk Fusion 360, został przedstawiony na rysunku 6.6.

Poniżej zostały zamieszczone zrzuty ekranu z programu Fusion 360 w celu zobrazowania, zdaniem autora, najważniejszych elementów. Obudowa była projektowana w dwóch częściach, najpierw baza, a po ukończeniu fazy testów została wydrukowana górna część konstrukcji.

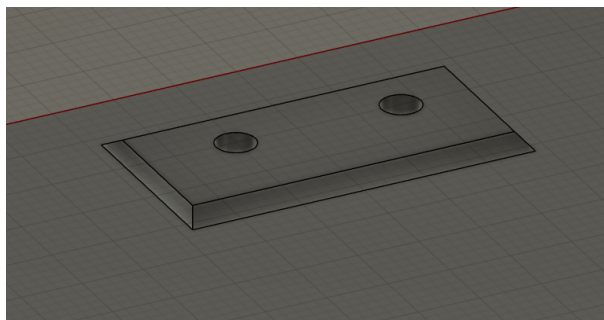
Technologia druku 3D umożliwiła precyzyjne rozmieszczenie mocowań na silniki oraz zastosowanie podpór kulowych na spodzie platformy, co zwiększyło stabilność konstrukcji. Przednia ściana została zaprojektowana jako integralna część podłogi, aby zapewnić jej odpowiednią sztywność i wytrzymałość. Wewnątrz konstrukcji zainstalowano centralnie umieszczony element wspornikowy, który służy jako oparcie dla drugiego poziomu platformy. Na wyższej warstwie znajdują się komputer Raspberry Pi oraz kamera odpowiedzialna za wizyjne śledzenie otoczenia.

Ostatni etap obejmował zaprojektowanie górnej części obudowy, która pełni funkcję ochronną oraz pozwala na łatwy dostęp do niżej położonych elementów robota. W tym celu zastosowano mocowania, umożliwiające łatwe zdejmowanie i zakładanie górnej części obudowy na podstawę, co znacząco ułatwia konserwację oraz serwisowanie wewnętrznych komponentów.

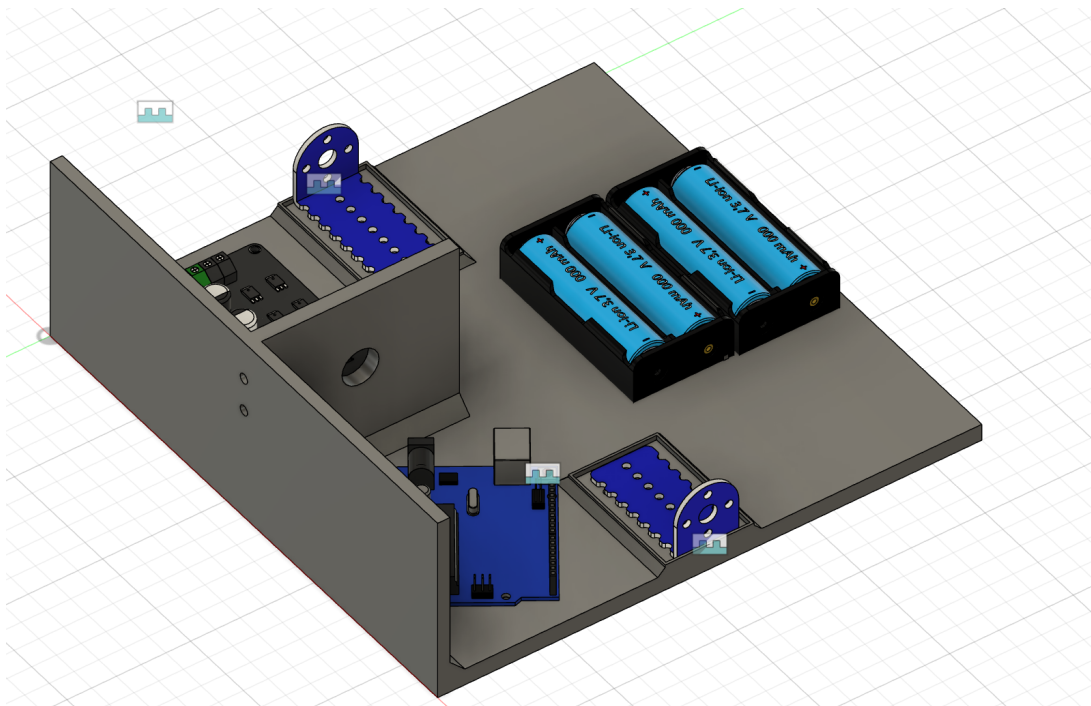
Ostateczna wersja platformy mobilnej, zaprezentowana na poniższej grafice oraz na nagraniach dołączonych do pracy, stanowi w pełni funkcjonalny model robota gotowy do realizacji zaplanowanych zadań w zakresie nawigacji i interakcji z otoczeniem.



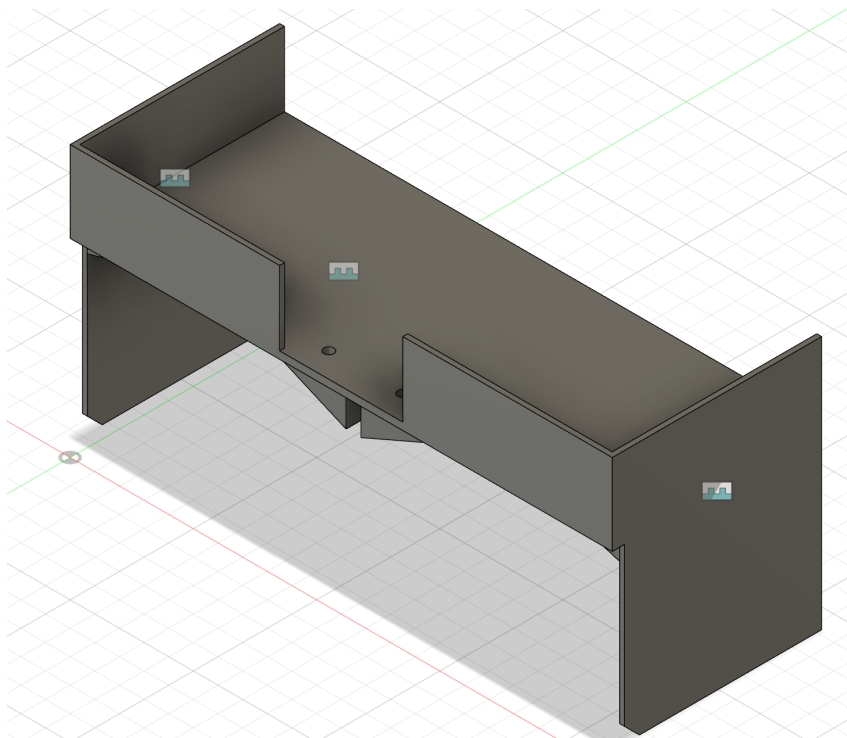
Rysunek 6.1: Przybliżenie miejsca na mocowania silników



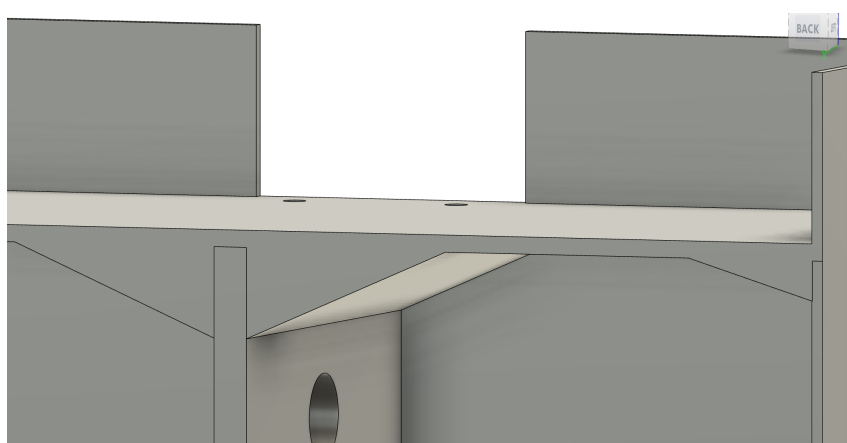
Rysunek 6.2: Przybliżenie miejsca mocowania kulek podporowych w osi robota



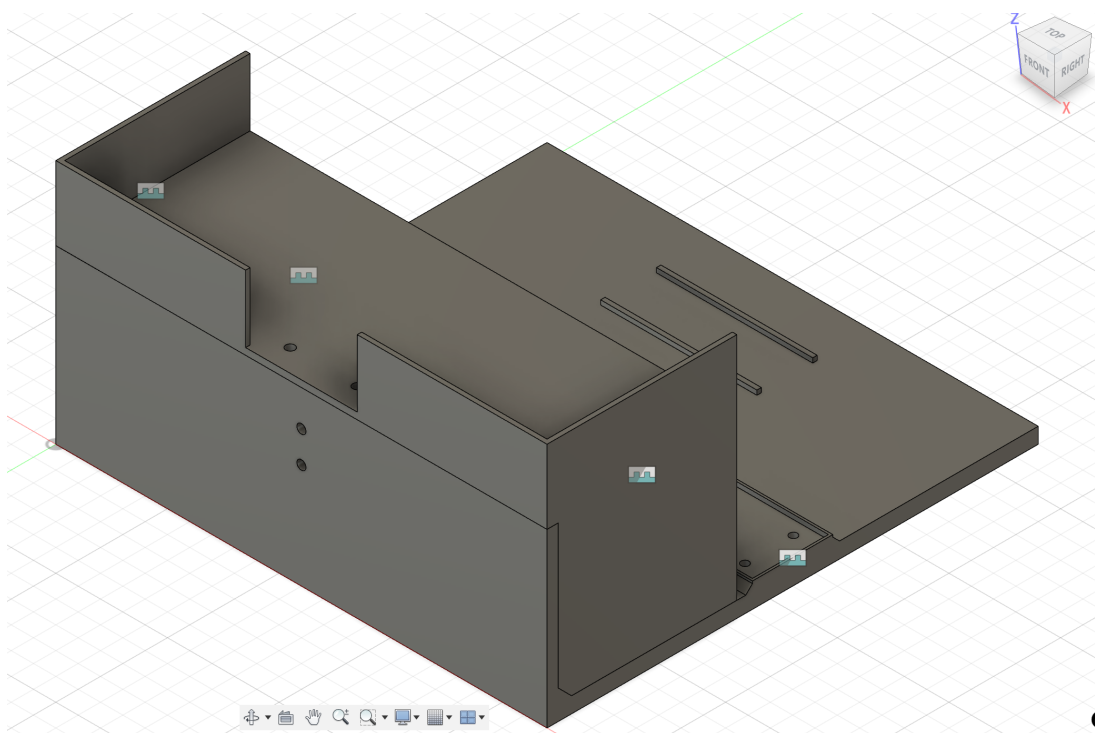
Rysunek 6.3: Projekt podwozia robota wraz z większością elementów w celu zobrazowania gotowej wersji



Rysunek 6.4: Przybliżenie miejsca mocowania kulek podporowych w osi robota



Rysunek 6.5: Przybliżenie sposobu połączenia górnej części obudowy z dolną. Widoczny jest również otwór przeznaczony na przewody połączeniowe



Rysunek 6.6: Prezentacja finalnej wersji obudowy robota

6.2 Implementacja kontrolera silników

Aplikacja kontrolera silników została zaimplementowana w środowisku Arduino IDE, co było podyktowane użyciem mikrokontrolera ATmega328P na płytce Arduino UNO R3. Zadaniem kontrolera było wysyłanie odpowiednich sygnałów PWM do sterownika silników oraz zliczanie impulsów enkoderów w celu uzyskania pożądanej dokładności.

W tej sekcji opisano kluczowe elementy aplikacji, w tym konfigurację pinów, algorytmy sterujące oraz komunikację z komputerem Raspberry Pi.

6.2.1 Konfiguracja pinów i ustawienia PWM

W kontrolerze skonfigurowano piny odpowiadające za obsługę enkoderów oraz sterowanie kierunkiem i prędkością silników za pomocą sygnałów PWM. Poniższy kod definiuje przypisanie pinów do odpowiednich funkcji, takich jak odczyt kanałów enkoderów oraz wyjść sterujących silnikami:

6.2.2 Algorytm sterowania PID

Algorytm sterowania PID odpowiada za utrzymanie zadanej prędkości kątowej każdego z silników poprzez dostosowanie sygnału PWM. Wyjście regulatora PID jest obliczane na podstawie bieżącego błędu pozycji enkodera, wartości całki i pochodnej błędu. Parametry regulatora, takie jak wzmocnienie proporcjonalne (K_p), całkujące (K_i) oraz różniczkujące (K_d), zostały dobrane empirycznie i wynoszą:

Na podstawie obliczonego błędu funkcja `performDrive()` dokonuje aktualizacji sygnału sterującego dla każdego silnika, uwzględniając przy tym warunki zatrzymania po osiągnięciu zadanej pozycji. Wartość wyjścia jest ograniczana do przedziału ± 70 , aby zapobiec zbyt dużemu skokowi napięcia sterującego:

Dodatkowo został wykorzystany globalny marker `stopCond1` odpowiedzialny za podtrzymanie silnika pierwszego (lub adekwatnie drugiego) w sytuacji, gdy pierwszy z nich zakończy pracę. Ze względu na pewną bezwładność silnika i dużą czułość enkoderów, brak zastosowania tej zmiennej skutkowało nieskończonym działaniem pętli.

6.2.3 Funkcja zliczająca impulsy enkoderów

Dla uzyskania dokładnych odczytów pozycji, w kontrolerze zastosowano obsługę przerw na pinach podłączonych do enkoderów. Funkcje `readEncoder1()` i `readEncoder2()` zliczają impulsy z enkoderów, aktualizując wartość liczników pozycji. W ten sposób każda zmiana sygnału na kanale enkodera jest przetwarzana w czasie rzeczywistym, co umożliwia precyzyjne śledzenie pozycji kół:

6.2.4 Realizacja komunikacji z Raspberry Pi

Komunikacja została zaimplementowana z wykorzystaniem ramki bitowej składającej się z dwóch bajtów. Pierwszy bajt odpowiada za przekazanie informacji o typie ruchu takich jak:

- Typ ruchu - czy jest to ruch do przodu czy skręt.
- Kierunek ruchu - jeśli typ ruchu to skręt, ten bit określa kierunek, w który robot ma się przemieścić.
- Ruch serwa - flaga określająca czy serwo ma wykonać ruch.
- Typ ruchu serwa - informuje mikrokontroler, czy serwo powinno się zamknąć czy otworzyć.

Drugi bajt odpowiada za przekazanie informacji o odległości jaką ma przejechać robot lub o jaki kąt powinien się obrócić.

Odczyt z magistrali szeregowej odbywa się w momencie, gdy są możliwe do odczytania dokładnie dwa bajty. Zrealizowane jest to za pomocą wbudowanej funkcji *Serial.available() >= 2*. Jeżeli warunek jest spełniony następuje odczytanie zawartości ramki oraz przepisanie jej wartości do zmiennych globalnych mikrokontrolera.

Fragment kodu realizujący odczyt znajduje się poniżej.

6.2.5 Pętla główna programu

Pętla główna programu opiera się na sprawdzaniu warunku dostępności dwóch bajtów w magistrali szeregowej. Jeżeli nie ma żadnych danych lub jest ich za mało, program przechodzi do sprawdzania kolejnych warunków. Jeżeli nastąpił odczyt, któryś z ich na pewno zostanie spełniony.

W kolejnych instrukcjach warunkowych znajdują się wywołania odpowiednich funkcji, realizujących ruch. Każda funkcja wykonawcza zwraca wartość typu *Boolean*, a więc prawda lub fałsz. Jeżeli zadanie zostanie poprawnie zrealizowane, zwrócona zostanie wartość *true*, a tym samym ostatnia instrukcja warunkowa, wysyłająca informację do komputera Raspberry Pi informację, że zadanie zostało ukończone i można przejść do kolejnej instrukcji oraz następuje wyzerowanie zmiennych.

Fragment realizujący zarządzanie zadaniami znajduje się poniżej.

```
1 // Encoders pins
2 #define ENCA1 2
3 #define ENCB1 4
4 #define ENCA2 3
5 #define ENCB2 5
6
7 // Motor driver pins
8 #define PWM1 10
9 #define DIR1 13
10 #define PWM2 11
11 #define DIR2 12
12
13 // Servo PWM pin
14 #define SERVO_PIN 9
```

Rysunek 6.7: Fragment przedstawiający konfigurację pinów

```
1 float Kp = 1.0;
2 float Ki = 0.5;
3 float Kd = 0.1;
```

Rysunek 6.8: Wartości parametrów regulatora PID

```

1  float error1 = targetCountsWheelRight - abs(pos1);
2  float error2 = targetCountsWheelLeft - abs(pos2);
3  float output1 = 0;
4  float output2 = 0;
5  unsigned long currT = micros();
6  float deltaT = (currT - prevT) / 1e6;
7  prevT = currT;
8
9  if(!stopCond1){
10     integral1 = error1 * deltaT;
11     float derivative1 = (error1 - previousError1) / deltaT;
12     output1 = Kp * error1 + Ki * integral1 + Kd * derivative1;
13
14     output1 = constrain(output1, -55, 55);
15     if (output1 > 0 && output1 < 30) output1 = 30;
16     if (output1 < 0 && output1 > -30) output1 = -30;
17
18     if (abs(error1) > tolerance && targetCounts != 0) {
19         setMotor(1, abs(output1), PWM1, DIR1);
20     } else {
21         stopCond1 = true;
22         setMotor(1, 0, PWM1, DIR1);
23     }
24 }

```

Rysunek 6.9: Fragment przedstawiający realizację regulacji PID

```

1  void readEncoder1() {
2      static int lastEncoded1 = 0;
3      int MSB1 = digitalRead(ENCA1); // Najbardziej znaczący
        bit
4      int LSB1 = digitalRead(ENCB1); // Najmniej znaczący bit
5      int encoded1 = (MSB1 << 1) | LSB1; // czenie dwóch
        bit w
6      if (MSB1 == LSB1) pos1++; else pos1--;
7      lastEncoded1 = encoded1;
8  }

```

Rysunek 6.10: Funkcja zliczająca impulsy enkodera

```
1  if (Serial.available() >= 2) {
2
3      firstByte = Serial.read();
4      secondByte = Serial.read();
5
6      movementType = (firstByte & 0b11000000) >> 6;
7
8      dir = (firstByte & 0b00100000) >> 5;
9
10     servo = (firstByte & 0b00010000) >> 4;
11
12     servoAction = (firstByte & 0b00001000) >> 3;
13
14     stopCond1 = false;
15     stopCond2 = false;
```

Rysunek 6.11: Fragment przedstawiający odczyt ramki bitowej z magistrali szeregowej

```
1  if(movementType == 1 && targetCounts != 0){
2      if(dir == 1){
3          status = performTurn(1, -1);
4      } else {
5          status = performTurn(-1, 1);
6      }
7      // status = dir == 1 ? performTurn(1, -1) : performTurn
        (-1, 1);
8  }
9  else if(movementType == 0 && targetCounts != 0){
10     status = performDrive();
11 }
12
13 if(servo){
14     //Serial.print("Servo Func evoke: ");
15     status = performGrab(servoAction);
16 }
17
18 if (status) {
19     sprintf(res, "FINISH->P1: %d | P2: %d | E1: %d | E2: %d",
20             , P1, P2, E1, E2);
21     Serial.println(res);
22     targetCounts=0;
23     targetCountsWheelRight = 0;
24     targetCountsWheelLeft = 0;
25     status = false;
}
```

Rysunek 6.12: Fragment przedstawiający logikę zarządzania otrzymanym zadaniem

6.3 Implementacja głównej aplikacji sterującej

Główna aplikacja sterująca, odpowiedzialna za zarządzanie operacjami robota, została napisana w języku Python w wersji 3.10. W aplikacji tej wykorzystano biblioteki takie jak OpenCV (do przetwarzania obrazu), *pyserial* (do komunikacji szeregowej z mikrokontrolerem) oraz *threading* (do zarządzania współbieżnością zadań). Aplikacja działa w trybie konsolowym, co pozwala na jej zdalne uruchomienie i kontrolowanie za pomocą protokołu SSH.

6.3.1 Opis funkcji głównej aplikacji `mainTask()`

Funkcja `mainTask()` pełni rolę głównej pętli zarządzającej działaniem aplikacji. Po uruchomieniu prezentuje użytkownikowi menu opcji sterujących, umożliwiających wybór trybu pracy robota: trybu automatycznego, trybu manualnego, otwarcia panelu operatorskiego lub zakończenia działania aplikacji. Umożliwia także płynne zarządzanie procesami robota przy użyciu współbieżnych podprocesów uruchamianych za pomocą biblioteki *threading*.

W poniższym kodzie, fragment wstępny inicjalizuje odpowiednie elementy biblioteki *threading*, takie jak flaga statusu `status` (typ `Event`), która umożliwia asynchroniczne monitorowanie stanu zakończenia poszczególnych zadań, co pozwala na bezpieczne zakończenie każdego procesu:

W kolejnych sekcjach opisano szczegółowo działanie każdej z dostępnych opcji w funkcji `mainTask()`.

Tryb automatyczny

Tryb automatyczny, aktywowany wyborem opcji 1, uruchamia niezależny wątek kontrolujący działanie robota w sposób autonomiczny. Wątek ten realizuje operacje poprzez funkcję `start_auto_control_task()`, której zadaniem jest uruchomienie instancji klasy `AutoModeModule`. Klasa ta, poprzez wywołanie metody `selectPath()`, realizuje sekwencję ruchów oraz operacji robota w trybie w pełni autonomicznym, umożliwiając monitorowanie postępu poprzez zmienną `status`.

Wątek `auto_control_thread` jest uruchamiany wewnątrz instrukcji `try-except`, co zabezpiecza program przed niespodziewanymi wyjątkami w trakcie jego działania, jak pokazano poniżej:

Funkcja `start_auto_control_task()` wykorzystuje klasę `AutoModeModule` do komunikacji z mikrokontrolerem przez port szeregowy oraz wywołuje zadania w pętli głównej.

```

1 import threading
2 import time
3 import os
4 import sys
5
6 # Configuration of module path
7 module_path = os.path.abspath( '/home/jakub/engineering_proj/
    robot-control-system/current_v2 ' )
8 sys.path.insert(0, module_path)
9
10 # Importing modules
11 from serial_communication import SerialCommunication
12 from auto_control_mode import AutoModeModule
13 from camera_module import CameraModule
14 from web_app.app import WebAppVisu

```

Rysunek 6.13: Importowanie bibliotek i modułów sterujących aplikacją

```

1 if choice == '1':
2     try:
3         auto_control_thread = threading.Thread(target=
4             start_auto_control_task, args=(serial_com, status))
5         auto_control_thread.start()
6         print("Auto_mode_has_started")
7
8         # Monitoring the auto mode until finished
9         while not status.is_set():
10             time.sleep(0.01)
11         auto_control_thread.join()
12         print("Auto_mode_has_finished")
13 except:
14     print("Exception_occurred_during_attempt_to_perform_auto
        _control")

```

Rysunek 6.14: Wybór i uruchomienie trybu automatycznego

Tryb ręczny i panel operatorski

Opcja 2, reprezentująca tryb manualny, jest zarezerwowana na dalszą implementację, umożliwiając użytkownikowi kontrolę robota w czasie rzeczywistym za pomocą klawiatury.

Opcja 3 natomiast uruchamia wątek `camera_module_task`, który włącza panel operatorski z wizualizacją kamery przy użyciu biblioteki OpenCV. Działa on poprzez funkcję `start_web_app_task()`, która inicjalizuje obiekt klasy `WebAppVisu` i wywołuje metodę `run()` odpowiedzialną za uruchomienie serwera aplikacji wizualizacyjnej. Wybór trybu ręcznego i uruchomienie panelu operatorskiego przedstawiono poniżej:

Zakończenie działania aplikacji

Wybór opcji 5 inicjuje procedurę zakończenia pracy programu. Zaimplementowana instrukcja warunkowa sprawdza, czy wątek `auto_control_thread` jest aktywny. W przypadku jego działania ustawiany jest `status` jako zakończony, co umożliwia bezpieczne zamknięcie wątku poprzez wywołanie metody `join()`, a następnie bezpieczne zamknięcie aplikacji.

Inicjalizacja funkcji pomocniczych

Funkcja `start_auto_control_task()` inicjuje działanie klasy `AutoModeModule`, która umożliwia pełną kontrolę autonomicznego poruszania się robota oraz wybór i realizację zadań. Obiekt tej klasy, przekazany przez parametr, komunikuje się z mikrokontrolerem, przysyłając komendy dotyczące prędkości i kierunku poruszania się.

Dodatkowo, funkcja `start_web_app_task()` inicjuje aplikację wizualizacyjną `WebAppVisu`, umożliwiającą przysyłanie i odbiór danych z kamery robota, co daje operatorowi podgląd w czasie rzeczywistym.

6.3.2 Opis sposobu komunikacji szeregowej

Komunikacja szeregową między jednostką sterującą (Raspberry Pi) a mikrokontrolerem (Arduino) jest realizowana za pomocą dedykowanej klasy `SerialCommunication`. Klasa ta została zaprojektowana w celu zapewnienia niezawodnej i bezpiecznej wymiany danych poprzez interfejs szeregowy z wykorzystaniem formatu binarnego, co umożliwia precyzyjne przekazywanie instrukcji do mikrokontrolera.

Inicjalizacja połączenia szeregowego

Klasa `SerialCommunication` otwiera połączenie szeregowe na wybranym porcie (w tym przypadku `/dev/ttyUSB0`) oraz ustawia szybkość transmisji (baud rate) na poziomie 250000 bps, co pozwala na szybkie przysyłanie danych w czasie rzeczywistym. Podczas ini-

```
1 elif choice == '3':
2     try:
3         camera_module_task = threading.Thread(target=
4             start_web_app_task)
5         camera_module_task.start()
6         print("Camera task has started correctly")
7     except:
8         print("Exception occurred during attempt to start camera
9             task")
```

Rysunek 6.15: Uruchomienie panelu operatorskiego z wizualizacją

```
1 elif choice == '5':
2     if auto_control_thread and auto_control_thread.is_alive():
3         status.set()
4         auto_control_thread.join()
5         print("Exiting...")
6     break
```

Rysunek 6.16: Kod zamykający wątki i kończący działanie aplikacji

```
1 def start_auto_control_task(serial_com, status):
2     auto_module = AutoModeModule(serial_com, status)
3     auto_module.selectPath()
4
5 def start_web_app_task():
6     web_app = WebAppVisu()
7     web_app.run()
```

Rysunek 6.17: Kod funkcji uruchamiających zadania kontrolne robota

cializacji sprawdzane jest, czy port został otwarty poprawnie, a następnie następuje reset buforów wejściowego i wyjściowego w celu zapewnienia spójności przesyłanych danych.

Wysyłanie danych do Arduino

Metoda `send_data` w klasie `SerialCommunication` odpowiada za przesyłanie preformatowanych instrukcji (dwubajtowych ramek binarnych) do Arduino. Funkcja przyjmuje jako argument `bytearray`, który reprezentuje ramkę danych składającą się z informacji dotyczących typu oraz parametrów ruchu robota. Każda ramka binarna jest zapisywana bezpośrednio do portu szeregowego jako surowe dane binarne, co zapewnia, że dane przesyłane są dokładnie w formacie oczekiwanym przez Arduino, bez dodatkowych konwersji.

Odbiór i potwierdzenie zakończenia operacji

Po wysłaniu ramki `send_data` oczekuje na potwierdzenie zakończenia zadania przez Arduino, które przesyła odpowiedź tekstową zawierającą frazę "FINISH" po zakończeniu zadanej operacji ruchu. Oczekiwanie na odpowiedź realizowane jest poprzez sprawdzenie, czy w buforze wejściowym pojawiły się dane do odczytu.

W przypadku obecności danych w buforze, metoda odczytuje zawartość jako strumień bajtów, konwertuje na tekst i sprawdza, czy zawiera on potwierdzenie zakończenia. Jeśli Arduino przesłało frazę "FINISH", metoda zwraca wartość `True`, co wskazuje, że zadanie zostało wykonane pomyślnie. W przeciwnym razie, jeśli brak jest danych w buforze lub wystąpił błąd, metoda zwraca `False`.

Obsługa błędów w komunikacji szeregowej

Komunikacja szeregową między Raspberry Pi a Arduino, mimo że jest stosunkowo niezawodna, może być podatna na błędy związane z zakłóceniami lub nieoczekiwanym przerwaniem połączenia. W związku z tym metoda `send_data` zabezpieczona jest instrukcją `try-except`, która wyłapuje wyjątki występujące podczas przesyłania danych. W przypadku wykrycia błędu zostaje wyświetlony komunikat o błędzie, a metoda zwraca `False`, co sygnalizuje niepowodzenie wykonania zadania.

Dzięki temu rozwiązaniu aplikacja jest bardziej odporna na nieprzewidziane błędy i może odpowiednio reagować w sytuacjach awaryjnych, takich jak utrata łączności.

6.3.3 Opis sposobu detekcji klocka oraz rozpoznawania koloru

Moduł kamery odpowiedzialny jest za detekcję klocków oraz rozpoznawanie ich kolorów w obrazie wideo na podstawie analizy kolorów w przestrzeni barw HSV oraz konturów. Realizacja tego zadania opiera się na przetwarzaniu strumienia wideo, który jest przechwytywany za pomocą kamery, a następnie przekształcany w czasie rzeczywistym.

```
1 def __init__(self):
2     self.ser = serial.Serial('/dev/ttyUSB0', 250000, timeout=1)
3
4     if self.ser.is_open:
5         print("Serial_port_is_open")
6
7     self.ser.reset_input_buffer()
8     self.ser.reset_output_buffer()
```

Rysunek 6.18: Kod inicjalizujący połączenie szeregowe

```
1 def send_data(self, data):
2     try:
3         # Wylij ramkę danych
4         self.ser.write(data)
5         print(f"The_binary_message_{data}_has_been_sent")
6         time.sleep(4)
```

Rysunek 6.19: Kod odpowiedzialny za wysyłanie danych do Arduino

```
1 # Sprawdź, czy są dostępne dane do odczytu
2 if self.ser.in_waiting > 0:
3     dane = b""
4
5     while self.ser.in_waiting > 0:
6         dane += self.ser.read(self.ser.in_waiting)
7
8     dane_str = dane.decode('utf-8', errors='ignore')
9     print("Received_data:", dane_str)
10
11     if "FINISH" in dane_str:
12         print('Arduino_confirmed_movement_completion.')
13         return True
14 else:
15     print("No_incoming_data!")
16     return False
```

Rysunek 6.20: Kod odbierający dane potwierdzające zakończenie zadania

```
1 except Exception as e:
2     print(f'Exception_occurred_during_serial_communication:{e}')
3     return False
```

Rysunek 6.21: Obsługa błędów komunikacji szeregowej

Klasa `CameraModule` zarządza procesem przechwytywania i analizowania obrazu, natomiast funkcje pomocnicze realizują zadania związane z maskowaniem kolorów, identyfikacją konturów oraz rozpoznawaniem charakterystycznych kształtów, takich jak kwadrat.

Definicja zakresów kolorów

Aby zidentyfikować kolory klocków, każdy kolor definiowany jest przez odpowiedni zakres wartości HSV. Zakresy te określają dolną i górną granicę składowych H (Hue), S (Saturation) i V (Value), umożliwiając tworzenie masek kolorystycznych, które izolują wybrane kolory w obrazie.

Detekcja koloru

Funkcja `find_color` przetwarza obraz w przestrzeni HSV w celu detekcji zadanych kolorów. Tworzy maskę kolorystyczną przy użyciu wcześniej zdefiniowanych zakresów HSV, a następnie identyfikuje kontury zgodne z kolorem poszukiwanego klocka. W procesie tym uwzględniane są jedynie kontury o powierzchni większej niż ustalony próg (5000 pikseli), co pozwala eliminować drobne zakłócenia.

Dla każdego wykrytego konturu obliczane są jego momenty, co umożliwia wyznaczenie środka masy (współrzędnych `cx` oraz `cy`). Jeśli kolorowy kontur zostanie wykryty, funkcja zwraca kontur oraz jego położenie, co pozwala na dalsze przetwarzanie w klasie `CameraModule`.

Detekcja i klasyfikacja kształtów

Po przekształceniu obrazu na odcienie szarości generowany jest binarny obraz progowy, który pozwala na detekcję konturów. Dla każdego konturu wyliczana jest jego przybliżona forma przy pomocy metody `approxPolyDP`. Kontury, które posiadają dokładnie cztery wierzchołki oraz odpowiednią powierzchnię (>1000 pikseli), klasyfikowane są jako kwadraty.

Po rozpoznaniu kwadratu obliczane są jego momenty w celu określenia współrzędnych środka. Kwadraty oznaczane są zielonym konturem oraz etykietą „Square”, co pozwala na wizualne potwierdzenie ich detekcji w obrazie.

Wizualizacja wyników

Wyniki detekcji są wizualizowane w czasie rzeczywistym na przechwyconym obrazie. Dla każdego zidentyfikowanego koloru rysowany jest kontur, etykieta koloru oraz punkt w centrum kształtu. W przypadku rozpoznania kwadratu, wokół konturu rysowana jest zielona ramka, a w jego środku wyświetlana etykieta „Square”. Proces przetwarzania kontynuowany jest w pętli, co pozwala na analizę kolejnych klatek obrazu w czasie rzeczywistym, aż do momentu zatrzymania programu.

```
1 colors = {
2     'blue': [np.array([95, 255, 85]), np.array([120, 255, 255])
3             ],
4     'red': [np.array([161, 165, 127]), np.array([178, 255, 255])
5            ],
6     'yellow': [np.array([16, 0, 99]), np.array([39, 255, 255])],
7     'green': [np.array([33, 19, 105]), np.array([77, 255, 255])
8             ],
9     'white': [np.array([0, 0, 200]), np.array([180, 50, 255])],
10 }
```

Rysunek 6.22: Definicje zakresów kolorów dla identyfikacji klocków

```
1 def find_color(frame, points):
2     mask = cv2.inRange(frame, points[0], points[1]) # Create
3     mask with boundaries
4     cnts = cv2.findContours(mask, cv2.RETR_TREE, cv2.
5     CHAIN_APPROX_SIMPLE)
6     cnts = imutils.grab_contours(cnts)
7
8     for c in cnts:
9         area = cv2.contourArea(c) # Calculate the area of the
10        contour
11        if area > 5000:
12            M = cv2.moments(c)
13            if M['m00'] != 0:
14                cx = int(M['m10'] / M['m00']) # X position
15                cy = int(M['m01'] / M['m00']) # Y position
16                return c, cx, cy
17
18 return None
```

Rysunek 6.23: Kod odpowiedzialny za detekcję kolorów klocków

- przedstawienie idei
- architektura systemu
- opis struktur danych (i organizacji baz danych)
- komponenty, moduły, biblioteki, przegląd ważniejszych klas (jeśli występują)
- przegląd ważniejszych algorytmów (jeśli występują)
- szczegóły implementacji wybranych fragmentów, zastosowane wzorce projektowe
- diagramy UML

```

1 for contour in contours:
2     approx = cv2.approxPolyDP(contour, 0.02 * cv2.arcLength(
        contour, True), True)
3     if len(approx) == 4 and cv2.contourArea(approx) > 1000:
4         x, y, w, h = cv2.boundingRect(approx)
5         aspect_ratio = float(w) / h
6         if 0.8 <= aspect_ratio <= 1.2:
7             cv2.drawContours(frame, [approx], 0, (0, 255, 0), 3)
8             M = cv2.moments(contour)
9             if M['m00'] != 0:
10                 cx = int(M['m10'] / M['m00'])
11                 cy = int(M['m01'] / M['m00'])
12                 cv2.putText(frame, "Square", (cx, cy), cv2.
                        FONT_HERSHEY_SIMPLEX, 0.6, display_color, 2)

```

Rysunek 6.24: Kod detekcji i klasyfikacji kształtów kwadratowych

```

1 cv2.drawContours(frame, [c], -1, display_color, 3)
2 cv2.circle(frame, (cx, cy), 7, display_color, -1)
3 cv2.putText(frame, name, (cx, cy), cv2.FONT_HERSHEY_SIMPLEX, 1,
        display_color, 2)

```

Rysunek 6.25: Kod wizualizacji wyników detekcji kolorów i kształtów

Rozdział 7

Weryfikacja i walidacja

- sposób testowania w ramach pracy (np. odniesienie do modelu V)
- organizacja eksperymentów
- przypadki testowe zakres testowania (pełny/niepełny)
- wykryte i usunięte błędy
- opcjonalnie wyniki badań eksperymentalnych

Tabela 7.1: Nagłówek tabeli jest nad tabelą.

ζ	metoda						
	alg. 1	alg. 2	alg. 3			alg. 4, $\gamma = 2$	
			$\alpha = 1.5$	$\alpha = 2$	$\alpha = 3$	$\beta = 0.1$	$\beta = -0.1$
0	8.3250	1.45305	7.5791	14.8517	20.0028	1.16396	1.1365
5	0.6111	2.27126	6.9952	13.8560	18.6064	1.18659	1.1630
10	11.6126	2.69218	6.2520	12.5202	16.8278	1.23180	1.2045
15	0.5665	2.95046	5.7753	11.4588	15.4837	1.25131	1.2614
20	15.8728	3.07225	5.3071	10.3935	13.8738	1.25307	1.2217
25	0.9791	3.19034	5.4575	9.9533	13.0721	1.27104	1.2640
30	2.0228	3.27474	5.7461	9.7164	12.2637	1.33404	1.3209
35	13.4210	3.36086	6.6735	10.0442	12.0270	1.35385	1.3059
40	13.2226	3.36420	7.7248	10.4495	12.0379	1.34919	1.2768
45	12.8445	3.47436	8.5539	10.8552	12.2773	1.42303	1.4362
50	12.9245	3.58228	9.2702	11.2183	12.3990	1.40922	1.3724

Rozdział 8

Podsumowanie i wnioski

- uzyskane wyniki w świetle postawionych celów i zdefiniowanych wyżej wymagań
- kierunki ewentualnych danych prac (rozbudowa funkcjonalna ...)
- problemy napotkane w trakcie pracy

Bibliografia

- [1] Aditi Bajaj i Jyotsna Sharma. „Computer Vision for Color Detection”. W: *International Journal of Innovative Research in Computer Science & Technology (IJIRCST)* (2020), s. 53–59.
- [2] Ricard Bitriá i Jordi Palacín. „Optimal PID Control of a Brushed DC Motor with an Embedded Low-Cost Magnetic Quadrature Encoder for Improved Step Overshoot and Undershoot Responses in a Mobile Robot Application”. W: *Robotics Laboratory, Universitat de Lleida, Jaume II 69, 25001 Lleida, Spain* (2022).
- [3] Botland.pl. *Botland.pl*. URL: <https://botland.com.pl/silniki-dc-z-przekladnia-i-enkoderami/6288-silnik-katowy-z-przekladnia-sj02-1201-6v-160rpm-enkoder-6959420910199.html>.
- [4] Gary Bradski i Adrian Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. United States of America, Sebastopol: O'Reilly Media, 2008. ISBN: 978-0-596-51613-0.
- [5] B. Cottenceau. *Introduction to Arduino*.
- [6] EBMiA. *EBMiA.pl*. 2023. URL: <https://www.ebmia.pl/wiedza/porady/automatyka-porady/czujnik-halla-co-to-jest-jak-dziala-i-jak-sprawdzic/>.
- [7] Raspberry Pi Ltd. *The Picamera2 Library. A libcamera-based Python library for Raspberry Pi cameras*.
- [8] Octavian Bologa Mihai Crenganis. „PID CONTROLLER FOR A DIFFERENTIAL STEERING MOBILE PLATFORM”. W: *Konferencji ASTR*. 2015, s. 6.
- [9] SHRIPAD G. DESAI PRINCE KUMAR. „Interrupts in The Microcontroller”. W: *IRE Journals* 4.4 (2021), s. 166–169.
- [10] ReearchGate.net. *ResearchGate*. URL: https://www.researchgate.net/figure/a-RGB-Color-Space-7-b-YCbCr-Color-Space-8_fig1_298734907.
- [11] ReearchGate.net. *ResearchGate*. URL: https://www.researchgate.net/figure/HSV-color-space-and-RGB-color-transformation_fig4_312678134.
- [12] ReearchGate.net. *ResearchGate*. URL: https://www.researchgate.net/figure/YCbCr-Color-Space-4_fig3_357594587.

- [13] Wikipedia.com. *Układ regulacji (automatyka)*. URL: https://pl.wikipedia.org/wiki/Uk%C5%82ad_regulacji_%28automatyka%29.
- [14] Djemel Ziou i Salvatore Tabbone. „Edge Detection Techniques-An Overvie”. W: *Pattern Recognition and Image Analysis: Advances in Mathematical Theory and Applications* 8.4 (1998), s. 537–559.

Dodatki

Spis skrótów i symboli

CSI szeregowy interfejs kamery (ang. *Camera Serial Interface*)

OpenCV otwarta biblioteka wizji komputerowej(ang. *Open Source Computer Vision Library*)

N liczebność zbioru danych

μ stopień przyleżności do zbioru

\mathbb{E} zbiór krawędzi grafu

\mathcal{L} transformata Laplace’a

Źródła

Jeżeli w pracy konieczne jest umieszczenie długich fragmentów kodu źródłowego, należy je przenieść w to miejsce.

```
1 if (_nClusters < 1)
2     throw std::string ("unknown_number_of_clusters");
3 if (_nIterations < 1 and _epsilon < 0)
4     throw std::string ("You should set a maximal number of
        iteration or minimal difference — epsilon.");
5 if (_nIterations > 0 and _epsilon > 0)
6     throw std::string ("Both number of iterations and minimal
        epsilon set — you should set either number of iterations
        or minimal epsilon.");
```

Lista dodatkowych plików, uzupełniających tekst pracy

W systemie do pracy dołączono dodatkowe pliki zawierające:

- źródła programu,
- dane testowe,
- film pokazujący działanie opracowanego oprogramowania lub zaprojektowanego i wykonanego urządzenia,
- itp.

Spis rysunków

3.1	Schamat UML zawierający podstawowe funkcjonalności	10
4.1	Rysunek przedstawiający zasadę działania czujnika Hall'a [6]	14
4.2	Rysunek przedstawiający ideowo budowę enkodera kwadraturowego oraz sygnałów przez niego generowanych [2]	15
4.3	Rysunek przedstawiający przykładowy wygląd enkodera opartego o efekt Hall'a. Czerwonymi kwadratami są zaznaczone tranzystory wykrywające zmianę pola magnetycznego podczas obrotu wału [3]	15
4.4	Rysunek przedstawiający ogólny schemat układu regulacji automatycznej [13]	16
4.5	Rysunek przedstawiający wizualizację przestrzeni kolorów RGB [10]	21
4.6	Rysunek przedstawiający wizualizację przestrzeni HSV [11]	21
4.7	Rysunek przedstawiający wizualizację przestrzeni YCrCb [12]	21
6.1	Przybliżenie miejsca na mocnowania silników	33
6.2	Przybliżenie miejsca mocowania kulek podporowych w osi robota	33
6.3	Projekt podwozia robota wraz z większością elementów w celu zobrazowania gotowej wersji	33
6.4	Przybliżenie miejsca mocowania kulek podporowych w osi robota	34
6.5	Przybliżenie sposobu połączenia górnej części obudowy z dolną. Widoczny jest również otwór przeznaczony na przewody połączeniowe	34
6.6	Prezentacja finalnej wersji obudowy robota	35
6.7	Fragment przedstawiający konfigurację pinów	38
6.8	Wartości parametrów regulatora PID	38
6.9	Fragment przedstawiający realizację regulacji PID	39
6.10	Funkcja zliczająca impulsy enkodera	39
6.11	Fragment przedstawiający odczyt ramki bitowej z magistrali szeregowej . .	40
6.12	Fragment przedstawiający logikę zarządzania otrzymanym zadaniem	40
6.13	Importowanie bibliotek i modułów sterujących aplikacją	42
6.14	Wybór i uruchomienie trybu automatycznego	42
6.15	Uruchomienie panelu operatorskiego z wizualizacją	44

6.16	Kod zamykający wątki i kończący działanie aplikacji	44
6.17	Kod funkcji uruchamiających zadania kontrolne robota	44
6.18	Kod inicjalizujący połączenie szeregowo	46
6.19	Kod odpowiedzialny za wysyłanie danych do Arduino	46
6.20	Kod odbierający dane potwierdzające zakończenie zadania	46
6.21	Obsługa błędów komunikacji szeregowej	46
6.22	Definicje zakresów kolorów dla identyfikacji klocków	48
6.23	Kod odpowiedzialny za detekcję kolorów klocków	48
6.24	Kod detekcji i klasyfikacji kształtów kwadratowych	50
6.25	Kod wizualizacji wyników detekcji kolorów i kształtów	50

Spis tabel

7.1	Nagłówek tabeli jest nad tabelą.	52
-----	--	----