Politechnika
Śląska

PROJEKT INŻYNIERSKI

Zrobotyzowany 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

Zrobotyzowany 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

Rozdział 1

Wstęp

Niniejsza praca inżynierska stanowi propozycję rozwiązania problemu automatycznego sortowania klocków na podstawie wizji komputerowej, a dokładnie rozpoznania koloru obiektu. Temat pracy jest niezwykle aktualny, ze względu na bardzo dynamicznie rozwijający się przemysł nowych technologii 4.0, a więc powszechną robotyzację i automatyzację procesów. Projekt ten wpisuje się w trend stwarzając możliwość wykorzystania go w szeroko pojętym przemyśle oraz powierzchniach magazynowych.

W rozdziale pierwszym zostanie omówiony problem, którego rozwiązaniem jest robot mobilny, osadzenie problemu w dziedzinie, cel pracy oraz krótki opis poszczególnych rozdziałów.

1.1 Wprowadzenie w problem

Proces sortowania przedmiotów jest zadaniem, do którego poprawnego wykonania konieczni byli ludzie, ze względu na umiejętność klasyfikowania obiektów na podstawie ich wizualnych cech. Wraz z rozwojem wizji komputerowej, możliwe stało się, aby oprogramowanie dokonywało podobnej klasyfikacji. Obecnie zaawansowane algorytmy uczenia maszynowego nierzadko są bardziej skuteczne i wielokrotnie bardziej wydajne od ludzi.

Jednak, aby przeprowadzić proces sortowania w dynamicznym środowisku, wymagana jest obecność medium transportowego. W tym miejscu na przeciw wychodzi robotyka. Połączenie platformy mobilnej wraz z obsługą wizji komputerowej pozwala na uzyskanie bardzo skutecznej formy sortowania obiektów.

Zastosowanie mobilnej platformy wyposażonej w kamerę jest szczególnie przydatne w dynamicznym środowisku przemysłowym lub magazynowym, gdzie istotna jest elastyczność i możliwość dostosowania trasy robota do aktualnych warunków.

Sam fakt automatyzacji procesu sortowania niesie ze sobą dużo korzyści takich jak: zmniejszenie kosztów operacyjnych, zwiększenie efektywności oraz redukcję błędów ludzkich w procesie sortowania.

1.2 Osadzenie problemu w dziedzinie

Projekt można osadzić w dziedzinie robotyki mobilnej wspartej wizją komputerową. Roboty mobilne, zdolne do samodzielnej lub częściowo samodzielnej nawigacji oraz wykonywania wszelakich zadań, znajdują coraz szersze zastosowanie w przemyśle lub logistyce. 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. Technologia ta, oparta na bibliotekach takich jak OpenCV, stała się dostępna nawet dla małych systemów wbudowanych, na przykład Raspberry Pi. W niniejszym projekcie robot wykorzystuje kamerę do rejestrowania obrazów obiektów, które następnie są analizowane w czasie rzeczywistym w celu określenia koloru, a na tej podstawie podejmowana jest decyzja, gdzie bieżący klocek powinien zostać przetransportowany.

1.3 Cel pracy

Celem projektu jest zaprojektowanie oraz zbudowanie platformy jezdnej pracującej w trybie automatycznym. Ponadto celem jest implementacja systemu kontroli robota, umożliwiającego samodzielne podejmowanie decyzji o wyborze miejsca docelowego dla danego obiektu.

Istotnym założeniem poprawności działania platformy jest dokładność i powtarzalność działania, wpływająca na możliwość automatycznej pracy bez ingerencji człowieka.

Dodatkowo system powinien być elastyczny, umożliwiający 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.
- 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.

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: Założenia wstępne** – Opis wymagań projektowych, przedstawienie diagramów funkcjonalnych, opis wykorzystanego sprzętu elektronicznego oraz metodyka pracy.
- **Rozdział 4: Podstawy teoretyczne** – Przedstawienie teoretycznych podstaw działania wykorzystanych technologii.
- **Rozdział 5: Wymagania i specyfikacja użytkowa** – Opis funkcjonalności systemu z perspektywy użytkownika,w tym sposobu instalacji oraz wykorzystania.
- **Rozdział 6: Specyfikacja techniczna** – Szczegółowa analiza implementacji systemu, w tym struktury oprogramowania i integracji z komponentami sprzętowymi.
- **Rozdział 7: Weryfikacja pracy robota** – Opis przeprowadzonych testów, ocena efektywności systemu oraz zgodności z założeniami projektowymi.
- **Rozdział 8: Podsumowanie i wnioski** – Zakończenie projektu, omówienie osiągniętych celów, a także wskazanie możliwych kierunków rozwoju systemu.

Rozdział 2

Analiza tematu

W rozdziale drugim zostanie przeanalizowany temat projektu, z uwzględnieniem aktualnego stanu wiedzy oraz znanych rozwiązań sterowania robotem mobilnym z napędem różnicowym oraz rozpoznawania kolorów z wykorzystaniem wizji komputerowej.

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

Temat projektu osadzić można w dziedzinach robotyki mobilnej oraz wizji komputerowej. W ciągu ostatnich lat dynamiczny rozwój wizji komputerowej zaowocował licznymi artykułami naukowymi oraz książkami poświęconymi tej tematyce.

Robotyka mobilna, kinematyka robotów oraz teoria sterowania obecna jest w środowisku naukowym od wielu lat, dzięki czemu z łatwością można znaleźć wiele materiałów poświęconych tej problematyce.

2.2 Studia literaturowe

Podczas prac nad poszczególnym elementami projektu istotną rolę odgrywała analiza dostępnej literatury naukowej. W sekcjach [2.2.1] oraz [2.2.2] znajduje się analiza artykułów, książek, dokumentacji oraz stron internetowych, których treść okazała się przydatna podczas pracy.

2.2.1 Opracowanie systemu wizyjnego

System wizyjny oparty został na dedykowanej kamerze dla platformy Raspberry Pi oraz na popularnej bibliotece OpenCV, co pozwala na efektywną analizę obrazu i klasyfikację obiektów w czasie rzeczywistym.

W tym celu kluczowa okazała się dokumentacja biblioteki Libcamera2 [12], oferująca szczegółowe informacje na temat instalacji, konfiguracji kamery, oraz wykorzystania łącza CSI (ang. *Camera Serial Interface*) do przesyła obrazu. Dokumentacja dostarczona przez Raspberry Pi Ltd zawiera obszerne wytyczne dotyczące ustawień obrazu, obsługi błędów oraz optymalizacji parametrów obrazu pod kątem specyficznych wymagań, takich jak rozmiar obrazu, automatycznego wyostrzenia obrazu, przestrzeni barw etc.

W celu zaawansowanego przetwarzania obrazu wykorzystano bibliotekę OpenCV. Dla zrozumienia jej funkcjonalności, zwłaszcza w kontekście realizacji projektu, cennym źródłem była literatura specjalistyczna, m.in. książka [6], 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 [3], 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 w celu wykrycia klocka, przy czym szczególna uwaga została poświęcona algorytmom i funkcjom dostępnym w bibliotece OpenCV. Podczas przeprowadzania studiów literaturowych przeanalizowane zostały artykuł [18], 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 oraz praca popełniona przez twórcę metody - John'a Canny [7].

2.2.2 Opracowanie systemu kontroli silników

System kontroli silników 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 [1] oraz [4], który szczegółowo omawia technikę regulacji PID (proporcjonalno-całkująco-różniczkującą) w zastosowaniu do robotów mobilnych z napędem różnicowym.

Istotnym elementem systemu kontroli była również implementacja funkcji zliczających impulsy 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.

Analiza dokumentacji mikrokontrolera ATMega328 [8] oraz literatury opisującej przerwania [13], umożliwiła dobre zrozumienie mechanizmów przerwań oraz wbudowanych modułów liczników. To pozwoliło na rozumienie ograniczeń wynikających z zastosowania kontrolera ATMega328 oraz wybór najlepszego dostępnego rozwiązania.

Rozdział 3

Założenia wstępne

W rozdziale trzecim przedstawiono opis wymagań funkcjonalnych, które określają, jakie zadania musi spełniać system robotyczny 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.

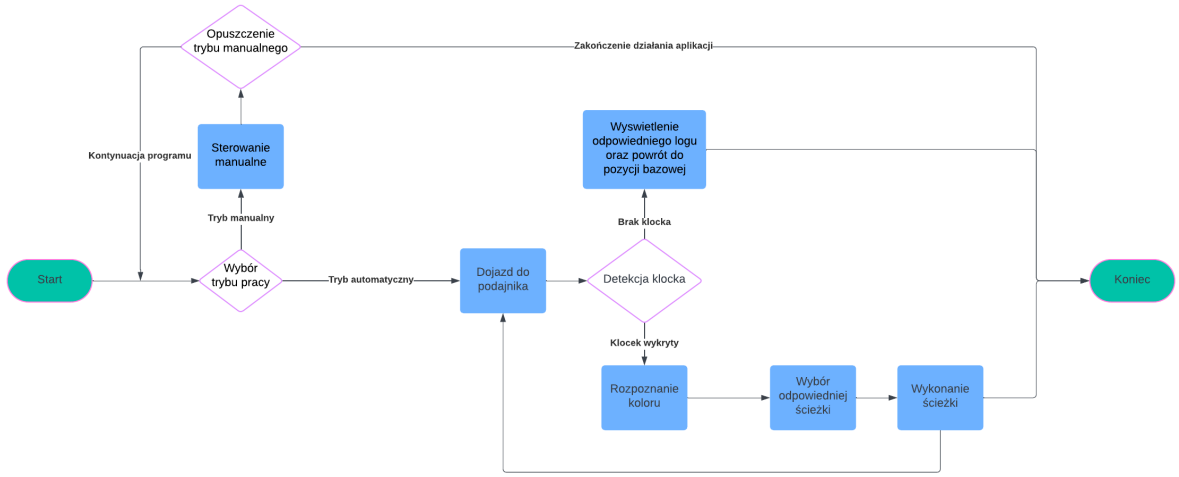
3.1 Wymagania funkcjonalne

Wymagania funkcjonalne odnoszą się do kluczowych funkcji systemu, które muszą zostać spełnione, aby system mógł realizować podstawowe zadania. W przypadku automatycznej 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).
- **Segregacja klocków** – po rozpoznaniu koloru, system transportuje klocek do odpowiedniego pojemnika.
- **Samodzielne poruszanie się** – robot powinien nawigować w wyznaczonej przestrzeni zgodnie z zaprogramowanymi trasami.

3.2 Przypadki użycia i diagramy UML

Diagram UML (ang. *Unified Modelling Language* - zunifikowany język modelowania) [3.1] zawiera schemat funkcjonalny aplikacji. Wyszczególniono na nim najważniejsze funkcjonalności oraz prawdopodobne zdarzenia. Szczegółowa obsługa błędów na poszczególnych etapach 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 krótkim 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.
- **Ogniwa litowo-jonowe typu 16850** - Pakiet składający się z czterech ogniw połączonych szeregowo odpowiada za zasilanie robota podczas pracy.

- **Płytka stykowa** - 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 (Agile), co umożliwiło sukcesywne rozwijanie i weryfikację funkcjonalności w miarę postępów prac. Na każdym etapie projektowania, implementacji oraz testowania gromadzono informacje zwrotne na temat działania systemu. Na ich podstawie podejmowane były decyzje o wprowadzeniu dodatkowych usprawnień oraz modyfikacji do pierwotnej wersji projektu.

Prace rozpoczęto od przygotowania środowiska programistycznego oraz poprawnego połączenia kamery z mikrokomputerem Raspberry Pi. Na tym etapie skupiono się na konfiguracji narzędzi do programowania i przetestowaniu prawidłowego działania kamery. Był to kluczowy element, gdyż poprawny 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 kontrolera przeprowadzono wstępne testy napędu różnicowego, analizując dokładność jazdy robota oraz zliczania impulsów generowanych przez enkodery.

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

W kolejnych iteracjach skupiono się na dalszym testowaniu i dostosowywaniu parametrów kontrolera oraz poprawkach programowych w celu osiągnięcia wymaganej dokładności.

Po uzyskaniu satysfakcjonujących rezultatów z zakresu precyzji jazdy 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 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 kontrolą mobilnej platformy robotycznej.

Omówione zostaną najważniejsze zasady funkcjonowania enkoderów kwadraturowych, wykorzystanych do monitorowania ruchu robota - 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 z zakresu robotyki, takich jak opisywana platforma mobilna, enkodery umożliwiają śledzenie oraz manipulowanie prędkością lub położeniem robota w przedstzeni dwuwymiarowej. Monitorowanie położenia robota za pomocą między innymi odczytów impulsów z enkoderów stanowi podstawę szerokiej dziedziny robotyki mobilnej - odometrii. W zależości od potrzeb, dodatkowym sensorem możliwym do wykorzystania jest czujnik żyroskopowy IMU - zwracający wartości przyspieszeń liniowych oraz kątowych. W tym projekcie zostały wykorzystane jedynie enkodery, ze względu na sposób poruszania się robota.

Enkodery dzielą się na enkodery optyczne, magnetyczne i mechaniczne, w zależności od zastosowanej technologii detekcji. Szczególna uwaga zostanie poświęcona enkoderom magnetycznym umożliwiających detekcję kierunku ruchu - enkoderom kwadraturowym.

4.1.1 Rodzaje enkoderów

Enkodery klasyfikowane są na podstawie zjawisk fizycznych wykorzystywanych do wykrywania ruchu. Główne typy to[2]:

- **Enkodery optyczne** - podstawę budowy stanowi przeźroczysta tarcza kodowa, przymocowana do wału silnika. Na tarczy znajdują się otwory w określonych miejscach, przez które na układ elementów światłoczułych pada wiązka światła podczerowego generowanego przez diodę IR LED.

- **Enkodery magnetyczne** - działają w oparciu o zmiany pola magnetycznego, zwykle generowanego przez magnes przymocowany do wału silnika. Zastosowanie efektu Hall'a w tych enkoderach pozwala na wykrywanie zmian w polu magnetycznym, poprzez generowanie napięcia w czujnikach Hall'a.

- **Enkodery mechaniczne** - najprostszy rodzaj enkoderów, wykorzystujący mechaniczne styki do generowania impulsów. Ze względu na ograniczoną trwałość są rzadziej stosowane w precyzyjnych aplikacjach.

4.1.2 Efekt Hall'a

Efekt Halla [10] stanowi podstawę działania magnetycznych enkoderów kwadraturowych, wykorzystywanych do pomiaru prędkości i kąta położenia wałów obrotowych. Odkryty przez Edwina Halla w 1879 roku, efekt ten polega na generowaniu tzw. napięcia Halla – różnicy potencjałów w przewodniku umieszczonym w zewnętrznym polu magnetycznym, przez który przepływa prąd. Zjawisko to wynika z działania siły Lorentza na nośniki ładunku w przewodniku, kierującej je w stronę prostopadłą do kierunku przepływu prądu oraz linii pola magnetycznego.

Siła Lorentza działająca na nośniki ładunku jest opisana równaniem:

$$\vec{F} = q \cdot (\vec{E} + \vec{v} \times \vec{B}),$$

gdzie:

- q - ładunek elektryczny nośnika prądu,

- $\vec{E}$ - wektor pola elektrycznego,

- $\vec{v}$ - prędkość nośnika ładunku,

- $\vec{B}$ - wektor indukcji magnetycznej.

Efektem działania siły Lorentza jest przemieszczenie nośników ładunku w kierunku prostopadłym do przepływu prądu i pola magnetycznego, prowadzące do powstania napięcia Halla (U_H) w poprzek przewodnika. Wartość tego napięcia jest opisana wzorem:

$$U_H = \frac{IB}{n \cdot q \cdot d},$$

gdzie:

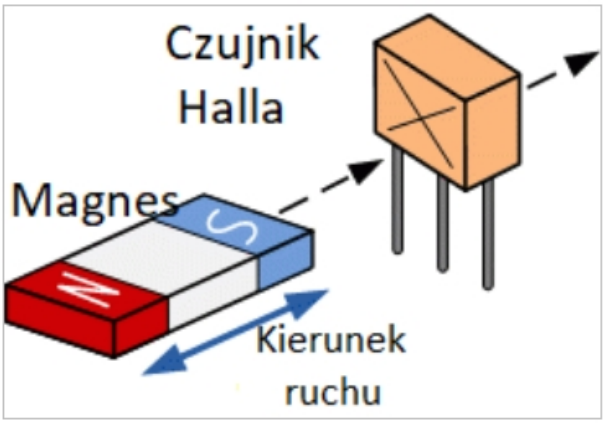
- I - natężenie prądu elektrycznego,
- B – indukcja magnetyczna,
- n – koncentracja nośników ładunku w materiale,
- d – grubość przewodnika.

4.1.3 Enkodery kwadraturowe oparte na efekcie Halla

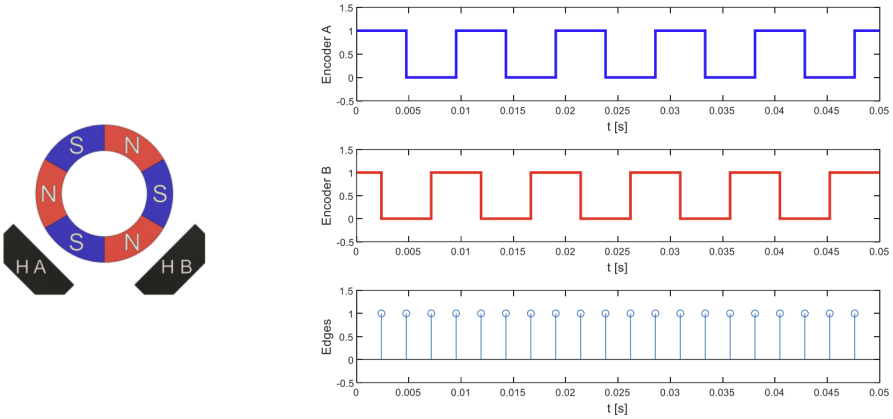
Enkodery kwadraturowe są szeroko stosowane do precyzyjnego pomiaru prędkości i kąta obrotu silnika oraz umożliwiają określanie kierunku ruchu. W magnetycznych enkoderach kwadraturowych, zbudowanych w oparciu o efekt Halla, umieszcza się dwa czujniki Halla rozmieszczone pod kątem 90° względem siebie [4.3]. Podczas obrotu wału silnika wraz z magnesem, pole magnetyczne ulega zmianie wraz z obrotem, co powoduje zmiany w napięciu Halla i dalej do generowania impulsów oznaczjących ruch silnika.

Zastosowanie jednego czujnika umożliwia wykrycie ruchu oraz zliczanie ilości impulsów generowanych przez obrót. Pewnym mankamanetem wykorzystania tylko jednego czujnika jest brak możliwości wykrycia kierunku obrotu. Dzięki dodaniu drugiego sensora Halla, kierunek ruchu można wykryć na podstawie odczytu sygnalu generowanego przez enkoder, ponieważ jeden z kanałów enkodera jako pierwszy zwróci stan wysoki, co będzie oznaczało obrót w konkretną stronę. Wizualizacja sygnalu generowanego przez enkoder kwadraturowy znajduje się na grafice [4.2].

Wysoka odporność na zużycie mechaniczne oraz stabilność działania w trudnych warunkach środowiskowych, takich jak wysoka wilgotność lub zapylenie, czynią magnetyczne enkodery kwadraturowe bardziej niezawodnymi niż ich optyczne odpowiedniki. Cechy te sprawiają, że magnetyczne enkodery kwadraturowe są szczególnie atrakcyjne w zastosowaniach przemysłowych oraz robotycznych, gdzie kluczowe są trwałość i precyzja.



Rysunek 4.1: Rysunek przedstawiający zasadę działania czujnika Hall'a [9]



Rysunek 4.2: Rysunek przedstawiający ideową budowę enkodera kwadraturowego oraz sygnały przez niego generowane [4]

Rysunek 4.3: Przykładowy wygląd enkodera opartego o efekt Hall'a z oznaczonymi tranzystorami wykrywającymi zmiany pola magnetycznego podczas obrotu wału [5]

4.2 Regulator PID

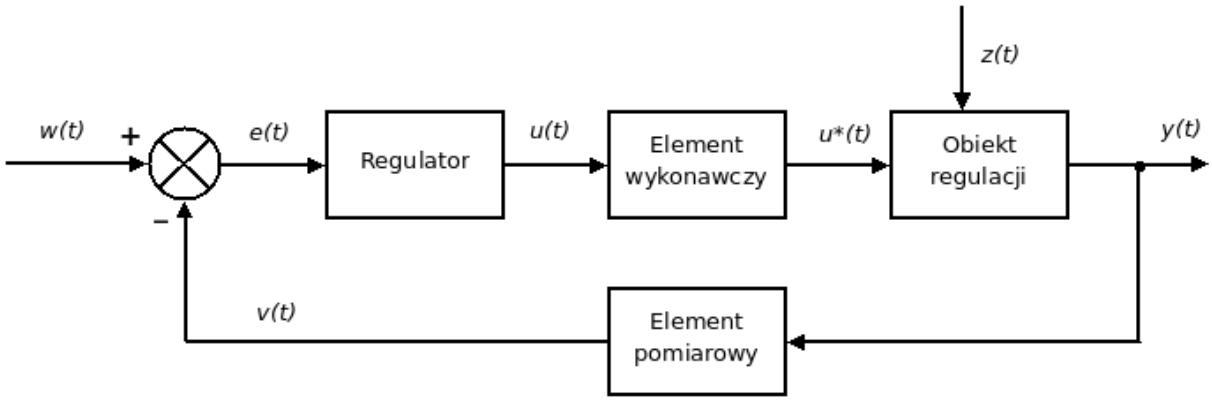
Regulator PID (ang. *Proportional-Integral-Derivative*) jest jednym z najczęściej stosowanych regulatorów w systemach automatycznego sterowania, ze względu na wysoką jakość regulacji gwarantującą uzyskanie zerwej wartości uchybu. Dodatkowo wykorzystanie aż trzech członów - proporcjonalnego, całkującego oraz różniczkującego, daje projektantowi duże możliwości dostosowania charakterystyki odpowiedzi czasowej regulatora, tak aby spełnić wymagania projektowe.

4.2.1 Ogólny schemat układu regulacji

Grafika [4.4] przedstawia ideowy schemat zamkniętego układu regulacji. Oznacza to, że występuje w nim pętla sprzężenia zwrotnego, a więc wartość wyjściowa zostaje dodana na wejściowy węzeł sumacyjny, w celu obliczenia uchybu.

Wartość zadana jest kierowana na wejście regulatora, obliczającego sterowanie, które następnie jest wejściem dla urządzenia wykonawczego. Ostatnim elementem jest odczyt wartości uzyskanych i ponowne przekazanie ich na wejście układu w celu korekcji sterowania.

Na zaprezentowanym schemacie występuje również czynnik $z(t)$, reprezentuje on zakłócenia wpływające na zachowanie wartości regulowanej. Wartość zakłócenia istotnie może zmienić wartość sterowania, która zostanie odpowiednio dostosowana poprzez regulatow w celu kompensacji szumu.



Rysunek 4.4: Rysunek przedstawiający ogólny schemat układu regulacji automatycznej [17]

4.2.2 Działanie regulatora PID

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

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

Uchyb jest jednym z podstawowych wskaźników jakości regulacji, określający jak dobrze wybrany regulator uzyskuje wartość zadaną. Cechą charakterystyczną regulatora PID jest całowika skuteczność uzyskania uchybu równego zero dla stabilnych układów dynamicznych. Zastosowanie innych regulatorów takich jak P (z jedynie członem proporcjonalnym) lub PI (z członem proporcjonalnym i drugim członem całkującym) nie gwarantuje uzyskania uchuby równego zero.

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

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

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. Połączenie ich w jedno równanie pozwala uzyskać bardzo dobrą jakość regulacji.

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:

$$P(t) = K_p e(t) \tag{4.3}$$

Im większa wartość wzmocnienia proporcjonalnego K_p, tym regulator bardziej dynamicznie reaguje na uchyb, co przyspiesza osiągnięcie wartości zadanej, ale przy zbyt dużych wartościach może prowadzić do przeregulowania, a w konsekwencji oscylacji.

4.2.4 Składnik całkujący I

Składnik całkujący sumuje wartości uchybu w czasie, co pozwala na eliminację uchybu ustalonego – różnicy między wartością zadaną a rzeczywiście uzyskaną w stanie ustalonym.

$$e_u = \lim_{t \to \infty} e(t) \tag{4.4}$$

- e_u - uchyb w stanie ustalonym,

- $e(t)$ - funkcja wiążąca wartość uchybu z czasem

Dla stabilnych układów dynamicznych wartość uchybu w granicy, gdzie czas dąży do nieskończoności, osiąga wartość stałą. Im jest mniejsza, tym jakość regulacji jest bardziej dokładna.

Składnik całkujący jest dany równaniem:

$$I(t) = K_i \int_0^t e(\tau)d\tau \tag{4.5}$$

Dzięki temu składnikowi, regulator PID potrafi zniwelować długotrwały uchyb, chociaż przy zbyt dużych wartościach wzmocnienia K_i może wystąpić zjawisko przeregulowania oraz niestabilności układu - opisanej w sekcji [4.2.7].

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:

$$D(t) = K_d \frac{de(t)}{dt} \tag{4.6}$$

Ten składnik regulatora poprawia stabilność i tłumi oscylacje, jednak zbyt duża wartość wzmocnienia 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:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{4.7}$$

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.

Często ze względu na realtywnie prostsze obliczenia, szczególnie podczas analizy stabilności oraz wyznaczania charakterystyk dynamicznych stosuje się postać operatorową (przekształcając równianie w postaci czasowej za pomocą operatora Laplace'a [L]) regulatora PID.

$$U(s) = (K_p + \frac{K_i}{s} + K_d s)E(s), \tag{4.8}$$

gdzie:

- $E(s)$ - uchyb w dziedzinie operatorowej,
- K_p, K_i, K_d wartości wzmocnienień odpowiednich członów.

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 Stabilność układów dynamicznych

Stabilność układów dynamicznych jest kluczowym zagadnieniem podczas projektowania układu regulacji. W analizie stabilności sprawdza się, czy układ dąży do stanu równowagi - minimalizuje uchyb, aby fianlnie osiągnąć wartość minimalną, w przypadku regulatora PID jest to zero. W przypadku układów niestabilnych, wartość uchybu dąży do nieskończoności, co jest szczególnie nieporządane podczas projektowania rzeczywistych układów, ze względu na możliwe uszkodzenie elementów wykonawczych poprzez podanie na ich wejście zbyt dużych wartości napięcia, prądu.

W przypadku regulatora PID analiza stabilności opiera się wyznaczeniu charakterystyk dla zadanych nastaw - wartości parametrów wzmocnienia K_p, K_i, K_d. Najczęściej w celu weryfikacji stabilności układów stosuje się metody:

- **Kryterium Hurwitza** – bazuje na analizie współczynników charakterystycznych układu dynamicznego w dziedzinie operatora Laplace'a. Metoda polega na ocenie znaku podwyznaczników macierzy Hurwitza, aby określić, czy wszystkie pierwiastki wielomianu charakterystycznego mają część rzeczywistą mniejszą od zera.
- **Kryterium Nyquista** – metoda graficzna oparta na analizie funkcji transmitancji układu na wykresie Nyquista. Kryterium to bada, jak charakterystyka amplitudowo-fazowa zamkniętego układu reaguje na różne częstotliwości sygnału wejściowego. Wykres Nyquista pozwala ocenić stabilność systemu poprzez obserwację, czy jego kontur w dziedzinie częstotliwości otacza określony punkt krytyczny. Kryterium Nyquista nadaje się szczególnie do badania stabilności układów nieliniowych i bardziej złożonych systemów z opóźnieniami.

Teoretyczna analiza stabilności jest punktem wyjścia przy projektowaniu regulatorów PID, ponieważ zbyt duże wartości parametrów K_p, K_i, lub K_d mogą prowadzić do niestabilnych oscylacji. Przykładowo, wysoka wartość wzmocnienia proporcjonalnego K_p

zwiększa szybkość reakcji układu, ale zbyt duża może powodować oscylacje wokół wartości zadanej. Z kolei nadmierne wzmocnienie członu różniczkującego K_d prowadzi do niestabilności w obecności szumu. Metody Hurwitza i Nyquista pomagają przewidzieć te reakcje i dobrać parametry regulatora tak, aby uzyskać stabilność i oczekiwane właściwości dynamiczne układu.

4.2.8 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. Uzyskana jakość regulacji jest wystarczająca do spełnienia wymagań projektowych.

4.3 Napęd różnicowy

Napęd różnicowy jest często stosowanym rozwiązaniem w robotach mobilnych do wielu zastosowań, ze względu na stosunkową prostą konstrukcję oraz duże możliwości ruchu. Konstrukcja napędu składa się z dwóch niezależnie napędzanych kół, umieszczonych po przeciwnych stronach osi robota. Dodatkowym elementem konstrukcyjnym może być jedno lub więcej biernych kół podporowych, zapewniających lepszą stabilność platformy - w przypadku poniższego projektu zostały zastosowane dwa elementy bierne podporowe.

4.3.1 Kinematyka napędu różnicowego

Model kinematyczny napędu różnicowego opisuje związek między prędkościami kół a ruchem platformy w przestrzeni dwuwymiarowej. W przypadku idealnym zakładającym brak poślizgu kół oraz brak luzów na przekładni, prędkość liniowa platformy υ oraz prędkość kątowa ω wyrażają się równaniami [4.9].

$$\upsilon = \frac{\upsilon_r + \upsilon_l}{2}, \omega = \frac{\upsilon_r + \upsilon_l}{b}, \tag{4.9}$$

gdzie:

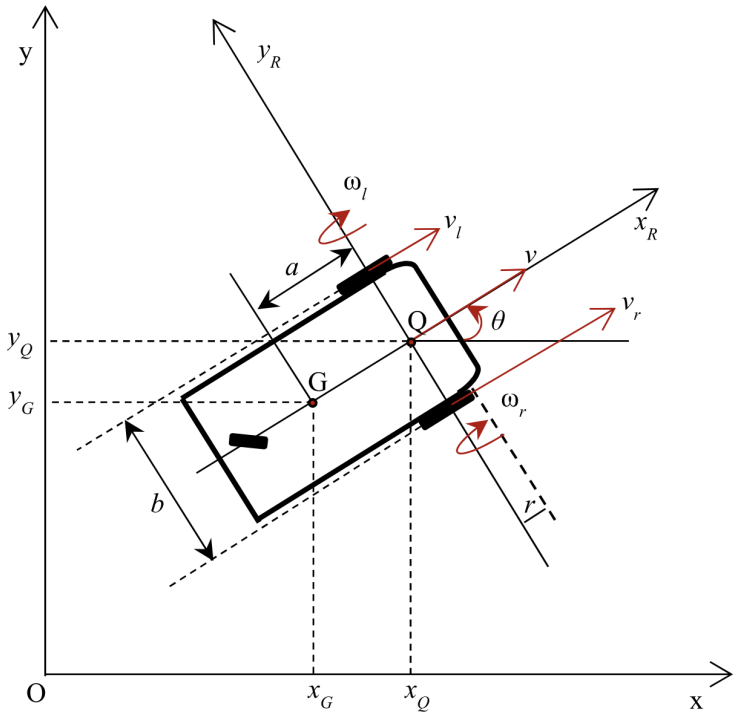
- υ_r oraz υ_l to prędkości odpowiednio prawego oraz lewego koła,
- b oznacza odległość pomiędzy osiami kół.

W bazowym układzie współrzędnych przemieszczenie robota opisują równania [**??**]:

$$\dot{x} = \upsilon cos(\theta), \dot{y} = \upsilon sin(\theta), \dot{(\theta)} = \omega, \tag{4.10}$$

gdzie:

- x i y - współrzędne środka masy w układzie współrzędnych odniesienia
- θ - orientacja platformy.



Rysunek 4.5: Grafika stanowiąca wizualizację przekształceń kinematyki robota [1]

4.4 Odometria - metoda przyrostowa

Odometria jest metodą znaną od lat, mimo to nadal szeroko stosowaną jako metodę nawigacji w robotyce mobilnej. Cechuje się dobrą dokładnością w krótkim oknie czasowym oraz relatywnie niską ceną wdrożenia. Podstawą odometrii jest całkowanie (sumowanie) przyrostowej informacji pozyskanej na przykład z enkoderów inkrementalnych (przyrostowych) zamontowanych na napędzie, w czasie. Może to prowadzić do dużej akumulacji błędów w przypadku szerokiego okna czasowego, a tym samym do znacznej zmiany orientacji robota względem założonej. Mimo to, metoda ta jest szeroko stosowana, między innymi w połączeniu z innymi metodami orientacji robota w przestrzeni w celu eliminacji nagromadzonych błędów.

Aby wyznaczyć, ile impulsów należy zliczyć w celu wykonania określonego przemieszczenia względem układu bazowego, należy skorzystać ze wzoru 4.11, a następnie 4.12.

$$c_m = \Pi \frac{D_n}{nC_e} \tag{4.11}$$

,gdzie:

- c_m - współczynnik przeliczenia impulsów licznika na przebytą odległość,

- D_n - nominalna średnica kół,
- C_e - rozdzielczość licznika w impulsach na obrót,
- n - współczynnik biegu - przeliczenie obrotów silnika na obroty kół.

$$\Delta U_{R/L,I} = c_m N_{R/L,I} \tag{4.12}$$

,gdzie:

- $\Delta U_{R/L,I}$ - przebyta odległość,
- c_m - współczynnik przeliczenia impulsów licznika na przebytą odległość,
- $N_{R/L,I}$ - ilość zliczonych impulsów przez enkodery koła prawego (R) oraz lewego (L) w krótkim czasie I.

4.5 Podstawy wykrywania krawędzi w wizji komputerowej

Wykrywanie krawędzi to jedna z podstawowych technik przetwarzania obrazów stosowana w wizji komputerowej, umożliwiająca ekstrakcję istotnych informacji strukturalnych z obrazu. Proces ten polega na detekcji nagłych zmian intensywności pikseli, które odpowiadają krawędziom obiektów w przestrzeni obrazu. Krawędzie dostarczają kluczowych danych o kształtach i konturach obiektów, wspomagając zadania takie jak segmentacja obrazu, rozpoznawanie obiektów oraz analiza ruchu. Wysoka jakość detekcji krawędzi jest zatem niezbędna dla dokładności i skuteczności systemów wizji komputerowej.

Jednym z najbardziej uznanych algorytmów wykrywania krawędzi jest metoda Canny'ego, opracowana przez Johna F. Canny'ego w 1986 roku. Algorytm ten jest ceniony za swoje właściwości w zakresie selektywnej detekcji krawędzi przy jednoczesnym tłumieniu szumów, co zapewnia spójną i wyraźną detekcję konturów.

4.5.1 Algorytm Canny'ego

Algorytm Canny'ego [7] to złożona metoda wieloetapowa, której celem jest precyzyjna detekcja krawędzi przy minimalnym wpływie szumów. Składa się on z pięciu kluczowych etapów:

1. **Wygładzanie** (ang. *Smoothing*): W celu redukcji wpływu szumów obraz jest najpierw poddawany filtracji za pomocą splotu z maską Gaussa, co pozwala na zachowanie istotnych struktur obrazu przy jednoczesnym tłumieniu drobnych zaburzeń. Przykładowa maska Gaussa o rozmiarze 3×3 ma postać:

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

2. **Obliczanie gradientu**: Po wygładzeniu obrazu przeprowadza się obliczenia gradientu intensywności dla każdego piksela, co pozwala na identyfikację miejsc o gwałtownych zmianach intensywności. Gradient jest obliczany w kierunkach poziomym (za pomocą maski Sobela S_x) i pionowym (za pomocą maski Sobela S_y):

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

   Na tej podstawie wyznaczana jest wielkość gradientu G:

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

   oraz jego kierunek:

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

3. **Tłumienie nie-maksymalne** (ang. *Non-maximum suppression*): W tym etapie algorytm identyfikuje i zachowuje jedynie lokalne maksima gradientu w kierunku krawędzi, co pozwala na wyostrzenie konturów krawędzi i eliminację nieistotnych pikseli, które nie leżą na krawędzi obiektów.

4. **Podwójne progowanie** (ang. *Double thresholding*): Po wstępnej selekcji krawędzi, algorytm wprowadza dwa progi — wysoki i niski. Piksele o wartości gradientu przekraczającej wysoki próg uznaje się za pewne krawędzie, podczas gdy piksele o wartościach poniżej niskiego progu są eliminowane. Piksele mieszczące się pomiędzy progami są klasyfikowane jako krawędzie słabe, które wymagają dalszej analizy.

5. **Śledzenie krawędzi z histerezą** (ang. *Edge tracking by hysteresis*): W końcowym kroku algorytm wykonuje śledzenie połączonych pikseli, aby zachować spójność krawędzi. Piksele oznaczone jako krawędzie słabe są akceptowane jako część krawędzi tylko wtedy, gdy są połączone z krawędziami silnymi. Ten proces zapobiega przypadkowej detekcji krawędzi i pozwala na eliminację fragmentarycznych lub pojedynczych pikseli wynikających z szumu.

Rysunek 4.6: Zdjęcie przedstawiające skuteczność wykrywania krawędzi algorytmem Canny'ego

4.6 Podstawy rozpoznawania kolorów w wizji komputerowej

Istotnym elementem wizji komputerowej jest możliwość rozpoznawania kolorów. Funkcjonalność ta jest szczególnie przydatna w automatyce, robotyce oraz w metodach sztucznej inteligencji. Pozwala na klasyfikację obiektów przy pomocy prostych algorytmów analizy wizyjnej, jak ma to miejsce w przypadku tego projektu, ale również jest częścią zaawansowanego trenowania sieci neuronowych, tym samym umożliwiając bardzo dokładną klasyfikację obiektów o różnym kształcie i kolorze.

W tej sekcji omówione zostaną podstawowe zasady oraz techniki wykorzystywane w procesie rozpoznawania kolorów.

4.6.1 Teoria koloru

Kolor jest wynikiem interpretacji długości fali światła przez ludzki wzrok. W systemach wizyjnych kolor jest reprezentowany za pomocą modeli kolorów, które pozwalają na matematyczną reprezentację informacji o barwie i jej intensywności. Najczęściej stosowane modele kolorów to przestrzenie RGB (ang. *Red, Green, Blue*), HSV (ang. *Hue, Saturation, Value*) oraz YCrCb (ang. *Luminance, Chrominance*).

- **Model RGB** [4.7] — Kolory reprezentowane są jako kombinacja trzech składowych: czerwonej, zielonej i niebieskiej. Jest to podstawowy model stosowany w systemach cyfrowych do rejestrowania i wyświetlania obrazów.
- **Model HSV** [4.8]— Przestrzeń HSV, zdefiniowana przez odcień, nasycenie i jasność, jest bardziej zbliżona do ludzkiego postrzegania kolorów. Pozwala ona na łatwiejsze odróżnianie odcieni niezależnie od zmian oświetlenia, co czyni ją szczególnie przydatną w detekcji kolorów w wizji komputerowej.
- **Model YCrCb** [4.9]— Oddziela jasność (luminancję) od informacji o kolorze (chrominancję). Ta przestrzeń kolorów jest powszechnie stosowana w kompresji obrazu i w systemach przetwarzania obrazu, gdyż zapewnia stabilność kolorów przy zmiennych warunkach oświetleniowych.

4.6.2 Proces rozpoznawania kolorów

Proces rozpoznawania kolorów obejmuje szereg etapów pozwalających na identyfikację pikseli o określonych wartościach kolorystycznych. Wyróżnić można następujące kroki:

1. **Przechwytywanie obrazu** — Obraz jest rejestrowany przy pomocy kamery, która przekazuje dane w formacie cyfrowym.

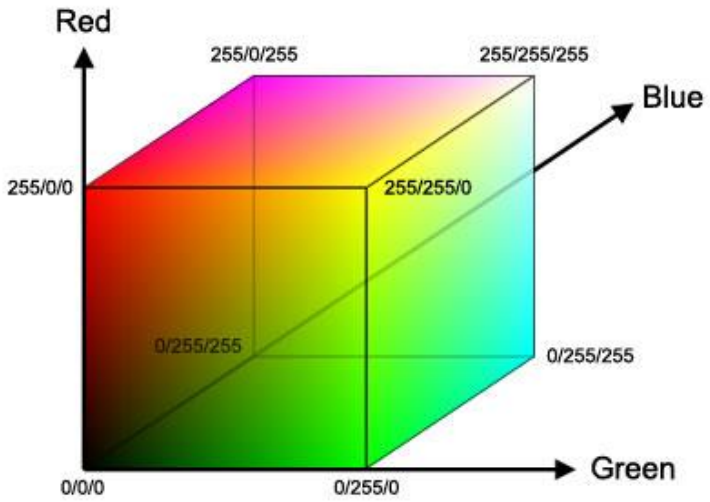
2. **Przetwarzanie obrazu** — Obraz przechodzi przez operacje wstępnego przetwarzania, takie jak filtrowanie i segmentacja, aby wyodrębnić istotne informacje kolorystyczne.

3. **Ekstrakcja cech** — Analizowane są wartości kolorystyczne pikseli, co umożliwia identyfikację obiektów na podstawie ich kolorów.

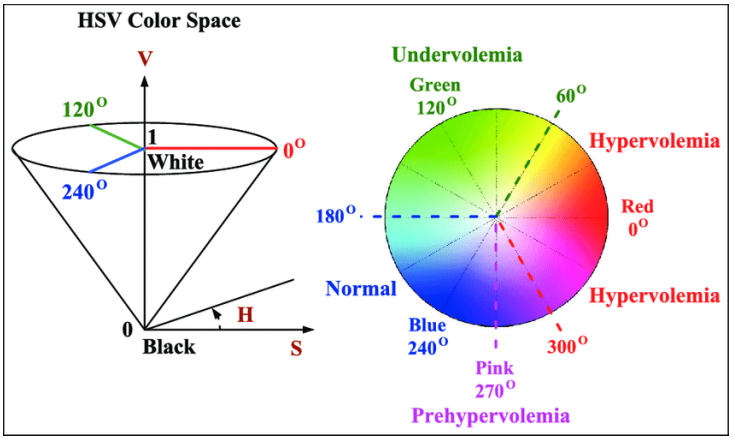
4. **Decyzja** — Na podstawie zebranych informacji o kolorze podejmowana jest decyzja o klasyfikacji obiektów.

4.6.3 Algorytmy rozpoznawania kolorów

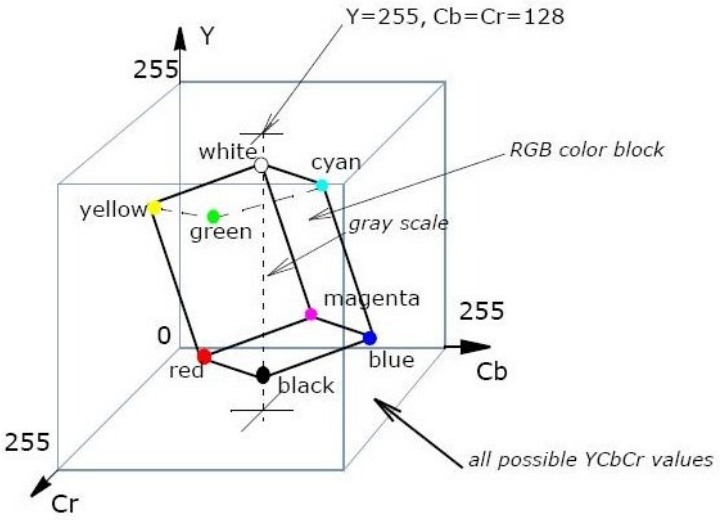
W literaturze przedmiotu występuje wiele podejść do rozpoznawania kolorów, takich jak analiza histogramu, segmentacja obrazu oraz analiza cech. Histogramy kolorów pozwalają na szybkie określenie dominujących kolorów na obrazie, co może być użyteczne w aplikacjach czasu rzeczywistego. Segmentacja obrazu natomiast umożliwia wyodrębnienie obiektów o określonym kolorze, co może być realizowane przy pomocy algorytmów takich jak k-means clustering lub metoda watershed.



Rysunek 4.7: Wizualizacja przestrzeni kolorów RGB [14]



Rysunek 4.8: Wizualizacja przestrzeni HSV [15]



Rysunek 4.9: Wizualizacja przestrzeni YCrCb [16]

Rozdział 5

Wymagania i specyfikacja użytkowa

W poniższym rozdziale szczegółowo opisano wymagania sprzętowe i programowe, sposób instalacji bibliotek oraz inne informacje niezbędne użytkownikowi do poprawnego uruchomienia i korzystania z systemu.

5.1 Wymagania sprzętowe i programowe

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

Wymagania sprzętowe

- Mikrokomputer Raspberry Pi – wybrany model Raspberry Pi 4 Model B, charakteryzuje się czterordzeniowym procesorem, zapewniającym 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 - do 16MPx. 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 kontrolowanie ruchu robota.

- Zasilanie – robot zasilany jest z pakietu czterech ogniw litowo-jonowych, dostarczających wystraczającą ilość napięcia oraz umożliwiających pracę przez odpowiedni czas.

- Chwytak - 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

Dobrą 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.

```
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:

```
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[[12]]):

```
    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.

```
    sudo apt-get update
    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 Instalacja środowiska Python Flask

Struktura programowania Flask umożliwia proste utworzenie aplikacji internetowej bazującej na szablonie HTML oraz kaskadowym arkuszu stylów (ang. *Cascading Style*

Sheets [CSS]) zapewniającym pożądany wygląd aplikacji.

Instalacja przebiega za pomocą menadżera pakietów *pip*:

```
pip install flask
```

Po instalacji możliwe jest utworzenie lokalnego serwera, do którego dostęp mają wszystkie urządzenie sieci lokalnej, co jest bardzo wygodne biorąc pod uwagę specyfikę wykorzystania tej aplikacji.

5.2.8 Weryfikacja instalacji bibliotek

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:

```
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:

```
python3
```

Następnie zaimportuj bibliotekę OpenCV, wpisując:

```
import cv2
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:

```
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.

5.2.9 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 jakoś 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.

5.2.10 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.

5.2.11 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.

5.2.12 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 wymogi projektu, z bardzo dobrą dokładnością, w celu wyeliminowania błędów wynikających z konstrukcji mechanicznej.

W fazie prototypowej zdecydowano się na wykorzystanie sklejki brzozowej – materiału taniego, lekkiego oraz łatwego w obróbce. Platforma bazowa prototypu miała wymiary: 240 mm długości oraz 220 mm szerokości. Silniki zostały osadzone na osi poprzecznej robota, zgodnie z wymogami nakładanymi przez sposób napędu.

Taka konfiguracja umożliwiła szybką weryfikację działania elementów wykonawczych oraz systemu sterowania, jeszcze przed przystąpieniem do bardziej zaawansowanych prac konstrukcyjnych. Zdjęcie [**??**] przedstawia opisany prototyp.

W toku prac nad wczesną wersją, 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 wprowadzeniu niezbędnych modyfikacji konstrukcyjnych w stosunku do prototypu. Ze względu na ograniczenia przestrzeni roboczej urządzenia drukującego, wymiary podstawy platformy zostały przeniesione na plan kwadratu o boku 220 mm. Przednia ściana została zaprojektowana jako część podłogi, w celu zwiększenia sztywności konstrukcji. Najcięższym elementem całego robota jest chwytak wraz z serwomechanizmem, przez co wytrzymałość przedniej ściany oraz podłogi była kluczowa. Zgodnie z osią do wewnątrz robota została poprowadzona przegroda, pełniąca funkcję podpory dla górnej części platformy. Finalny projekt podstawy, przygotowany w programie Autodesk Fusion 360, został przedstawiony na rysunku [6.5].

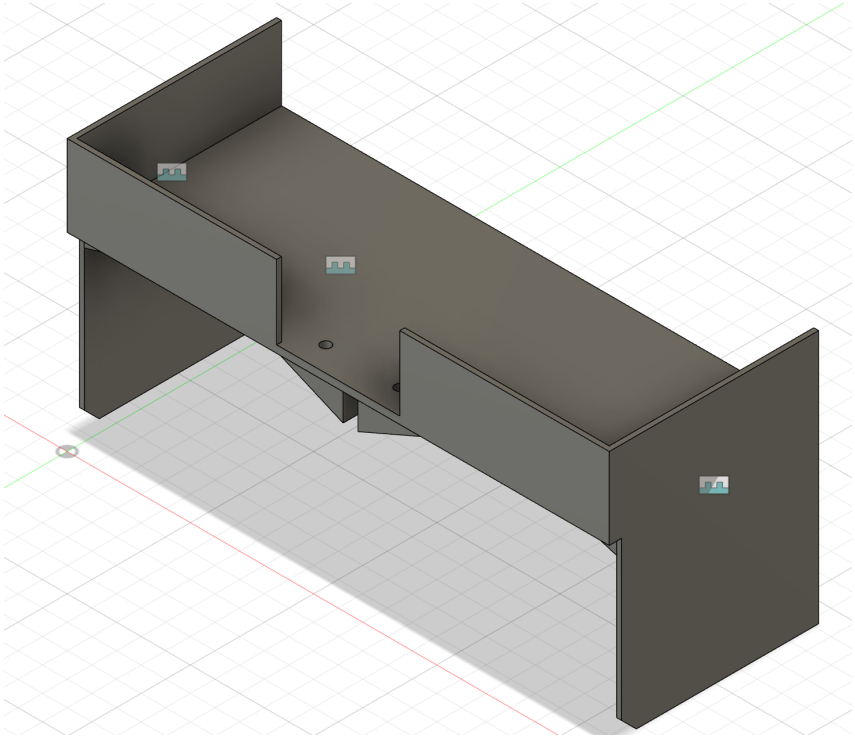
Ostatni etap obejmował zaprojektowanie górnej części obudowy, na której zostanie osadzony komputer Raspberry Pi oraz przymocowany uchwyt na kamerę. Sposób osadzenia górnej części na dolnym segmencie opiera się na precyzyjnie wykonanej przerwie, w którą wsuwa się wspornik bazy. Dzięki takiemu rozwiązaniu, w prosty i szybki sposób można zdjąć górną część i dokonać potrzebnych zmian lub modyfikacji. Te atrybuty są szczególnie przydatne podczas fazy testów i ciągłego ulepszania robota.

W wersji docelowej, przeznaczonej do komercyjnego użytkowania, omawiane mocowanie zostałoby zastąpione przez standardowe połączenia śrubowe lub sklejenie, co zwiększyłoby trwałość obudowy.

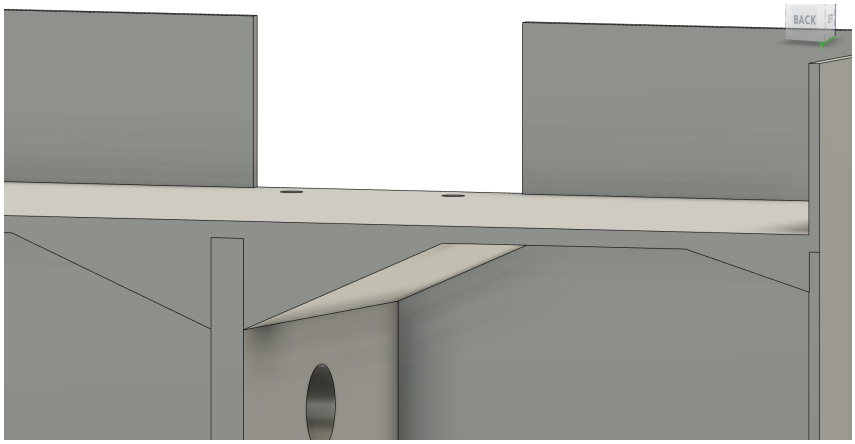
Ostateczna wersja platformy mobilnej została zaprezentowana na zdjęciu [**??**] oraz jest widoczna na nagraniach załączonych do pracy. Model obudowy stanowi w pełni funkcjonalny, zgodny z założeniami, robot, jednak część mocowań w wersji przeznaczonej do sprzedaży należy zmienić.

W tym celu zastosowano mocowania, umożliwiające łatwe zdejmowanie i zakładanie górnej części obudowy na podstawę, co znacząco ułatwia konserwację oraz serwisowanie wewnętrznych komponentów.

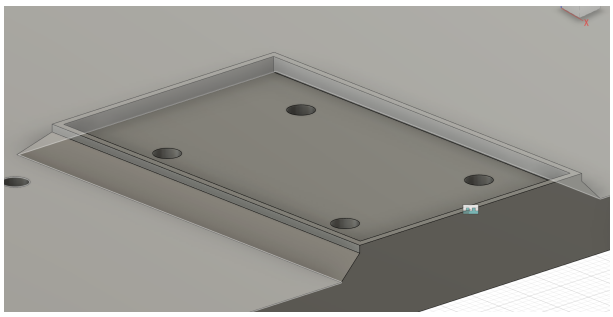
Finalna wersja platformy mobilnej została zaprezentowana na ilustracji 6.5, a także na załączonych nagraniach.



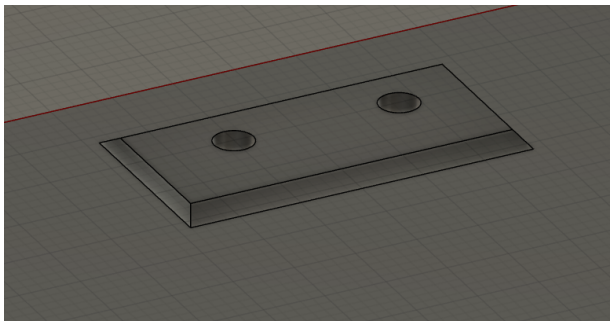
Rysunek 6.1: Model górnej części obudowy



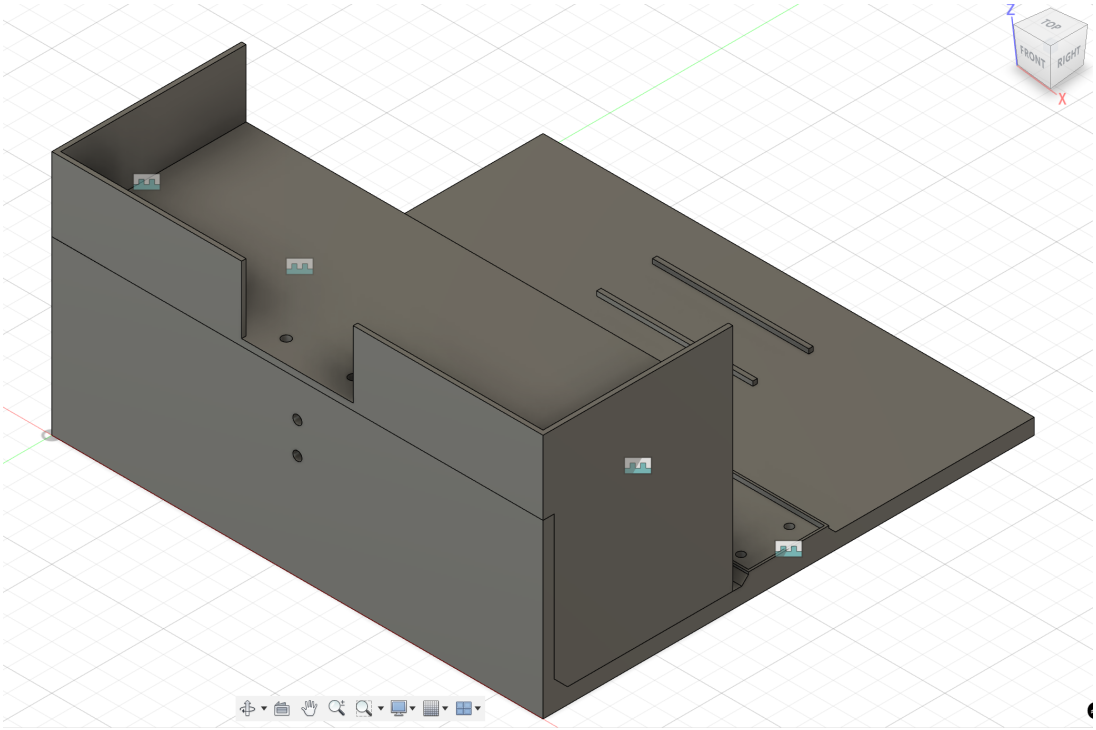
Rysunek 6.2: Przybliżenie sposobu połączenia górnej części obudowy z dolną. Widoczny jest również otwór przeznaczony na przewody połączeniowe



Rysunek 6.3: Przybliżenie miejsca na mocowania silników



Rysunek 6.4: Przybliżenie miejsca mocowania kulek podporowych w osi robota



Rysunek 6.5: 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:

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

Rysunek 6.6: Fragment przedstawiający konfigurację pinów

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 (`Kp`), całkujące (`Ki`) oraz różniczkujące (`Kd`), 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:

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

Rysunek 6.7: 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 6.8: Fragment przedstawiający realizację regulacji PID

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 skutkował nieskończonym działaniem pętli.

6.2.3 Funkcja zliczająca impulsy enkoderów

Dla uzyskania dokładnych odczytów pozycji, w kontrolerze zastosowano obsługę przerwań 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:

```
1  void readEncoder1() {
2    int MSB1 = digitalRead(ENCA1);
3
4    if (MSB1) {
5      posi1++;
6    }
7  }
8
9  void readEncoder2() {
10   int MSB2 = digitalRead(ENCA2);
11
12   if (MSB2) {
13     posi2++;
14   }
15 }
```

Rysunek 6.9: Funkcja zliczająca impulsy enkodera

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 odkreśla 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

```
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 6.10: Fragment przedstawiający odczyt ramki bitowej z magistrali szeregowej

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     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"
          , P1, P2, E1, E2);
20      Serial.println(res);
21      targetCounts=0;
22      targetCountsWheelRight = 0;
23      targetCountsWheelLeft = 0;
24      status = false;
25    }
```

Rysunek 6.11: 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:

```
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 6.12: Importowanie bibliotek i modułów sterujących aplikacją

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:

```
1  if choice == '1':
2      try:
3          auto_control_thread = threading.Thread(target=
               start_auto_control_task, args=(serial_com, status))
4          auto_control_thread.start()
5          print("Auto mode has started")
6
7          # Monitoring the auto mode until finished
8          while not status.is_set():
9              time.sleep(0.01)
10         auto_control_thread.join()
11         print("Auto mode has finished")
12     except:
13         print("Exception occurred during attempt to perform auto
               control")
```

Rysunek 6.13: Wybór i uruchomienie trybu automatycznego

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

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:

```
elif choice == '3':
    try:
        camera_module_task = threading.Thread(target=
            start_web_app_task)
        camera_module_task.start()
        print("Camera task has started correctly")
    except:
        print("Exception occurred during attempt to start camera
            Task")
```

Rysunek 6.14: Uruchomienie panelu operatorskiego z wizualizacją

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.

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

Rysunek 6.15: Kod zamykający wątki i kończący działanie 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, przesyłając komendy dotyczące prędkości i kierunku poruszania się.

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

6.3.2 Opis sposobu komunikacji szeregowej

Komunikacja szeregowa między jednostką sterującą (Raspberry Pi) a mikrokontrolerem (Arduino) jest realizowana za pomocą dedykowanej klasy `SerialCommunication`.

```
def start_auto_control_task(serial_com, status):
    auto_module = AutoModeModule(serial_com, status)
    auto_module.selectPath()

def start_web_app_task():
    web_app = WebAppVisu()
    web_app.run()
```

Rysunek 6.16: Kod funkcji uruchamiających zadania kontrolne robota

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 przesyłanie danych w czasie rzeczywistym. Podczas inicjalizacji sprawdzane jest, czy port został otwarty poprawnie, a następnie następuje reset buforów wejściowego i wyjściowego w celu zapewnienia spójności przesyłanych danych.

```
def __init__(self):
    self.ser = serial.Serial('/dev/ttyUSB0', 250000, timeout=1)

    if self.ser.is_open:
        print("Serial port is open")

    self.ser.reset_input_buffer()
    self.ser.reset_output_buffer()
```

Rysunek 6.17: Kod inicjalizujący połączenie szeregowe

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.

```
def send_data(self, data):
    try:
        # Wy lij ramk  danych
        self.ser.write(data)
        print(f"The binary message {data} has been sent")
        time.sleep(4)
```

Rysunek 6.18: Kod odpowiedzialny za wysyłanie danych do Arduino

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`.

```
# Sprawd , czy s  dost pne dane do odczytu
if self.ser.in_waiting > 0:
    dane = b""

    while self.ser.in_waiting > 0:
        dane += self.ser.read(self.ser.in_waiting)

    dane_str = dane.decode('utf-8', errors='ignore')
    print("Received data: ", dane_str)

    if "FINISH" in dane_str:
        print('Arduino confirmed movement completion.')
        return True
else:
    print("No incoming data!")
    return False
```

Rysunek 6.19: Kod odbierający dane potwierdzające zakończenie zadania

Obsługa błędów w komunikacji szeregowej

Komunikacja szeregowa 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.

```
1     except Exception as e:
2         print(f'Exception␣occurred␣during␣serial␣communication:␣{e
              }')
3         return False
```

Rysunek 6.20: Obsługa błędów komunikacji szeregowej

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. 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`.

```
colors = {
    'blue': [np.array([95, 255, 85]), np.array([120, 255, 255])],
    'red': [np.array([161, 165, 127]), np.array([178, 255, 255])],
    'yellow': [np.array([16, 0, 99]), np.array([39, 255, 255])],
    'green': [np.array([33, 19, 105]), np.array([77, 255, 255])],
    'white': [np.array([0, 0, 200]), np.array([180, 50, 255])],
}
```

Rysunek 6.21: Definicje zakresów kolorów dla identyfikacji klocków

```
def find_color(frame, points):
    mask = cv2.inRange(frame, points[0], points[1])  # Create mask with boundaries
    cnts = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)

    for c in cnts:
        area = cv2.contourArea(c)  # Calculate the area of the contour
        if area > 5000:
            M = cv2.moments(c)
            if M['m00'] != 0:
                cx = int(M['m10'] / M['m00'])  # X position
                cy = int(M['m01'] / M['m00'])  # Y position
                return c, cx, cy
    return None
```

Rysunek 6.22: Kod odpowiedzialny za detekcję kolorów klocków

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.

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

Rysunek 6.23: Kod detekcji i klasyfikacji kształtów kwadratowych

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.

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

Rysunek 6.24: Kod wizualizacji wyników detekcji kolorów i kształtów

6.4 Implementacja panelu operatorskiego

Implementacja panelu operatorskiego wybiega poza minimalne założenia projektowe, jednak ze względu na sposób pracy robota, została uznana za pomocną. Dzięki temu, użytkownik w czasie rzeczywistym może obserwować jakość działania systemu oraz w razie potrzeby stosownie zareagować.

System podglądu pracy robota został utworzony bazując na strukturze programowania Flask. Jest to popularny schemat tworzenia aplikacji internetowych, ponieważ zapewnia on możliwość utworzenia lekkiego i szybkiego serwera działającego lokalnie na maszynie. Jest to istotna kwestia, biorąc pod uwagę, komputer na którym aplikacja pracuje.

Struktura projektu jest relatywnie prosta, zawiera trzy główne składniki:

- folder *static* - zawiera wszystkie statyczne zasoby strony internetowej, takie jak pliki CSS odpowiadające za wygląd, obrazy oraz inne pliki multimedialne, które są ładowane po stronie klienta bez potrzeby ponownego przetwarzania przez serwer,
- folder *templates* - przechowuje pliki HTML, które definiują strukturę i wygląd interfejsu użytkownika. Flask wykorzystuje mechanizm szablonów Jinja2, co pozwala na dynamiczne wstawianie zmiennych i generowanie treści w zależności od danych otrzymywanych z robota w czasie rzeczywistym,
- plik *app.py* - pełni rolę głównego pliku aplikacji, obsługując zapytania HTTP oraz zarządzając komunikacją pomiędzy częścią serwerową a front-endem aplikacji. Plik ten inicjalizuje serwer Flask, odpowiada za odbieranie danych z robota oraz wysyłanie ich do szablonów HTML.

Rozdział 7

Weryfikacja pracy robota

7.1 Sposób testowania

Testy platformy mobilnej można podzielić na dwa etapy. Pierwszy z nich polegał na weryfikacji ilości impulsów zliczonych przez enkodery podczas ruchu robota. Był to kluczowy etap, pozwalający na wczesne wykrycie ewentualnych niedokładności związanych z algorytmem sterującym.

Drugi etap można nazwać właściwym procesem jakości algorytmów sterowania, ponieważ odbywał się on w warunkach zbliżonych do rzeczywistych. Robot poruszał się zgodnie z preprogramowaną ścieżką, natomiast weryfikacja była możliwa dzięki naklejonej taśmie malarskiej zgodnie z wytyczoną drogą. Był to prosty i efektywny sposób testowania jakości algorytmów.

7.2 Wykryte i usunięte błędy

7.2.1 Błąd nieskończonej pracy silników

Ważnym aspektem wykrytym w pierwszej fazie testów było niepoprawne zatrzymanie silników. Jeżeli silniki nie pracowały jednakowo, to znaczy, ich odczyty impulsów enkoderów nie były do siebie zbliżone, istniało duże ryzyko nieskończonej pracy silników.

Powodem takiego niepożądanego działania był warunek zakończenia pracy silników - fragment kodu znajduje się w spisie [6.8]. Widoczna instrukcja warunkowa sprawdza czy wartość bezwzględna błędu ruchu jest mniejsza niż wartość stałej tolerancji niedokładności. Jeżeli wartość błędu jest wyższa, naturalnie silniki pracują zgodnie z mocą wyznaczoną przez regulator PID, jeżeli natomiast wartość jest niższa, silniki zostają zatrzymane. Ze względu na pewną bezwładność elementów wykonawczych, mimo ich zatrzymania w pewnej iteracji *n*, w kolejnej iteracji *n+1* wartość błędu będzie wynosić więcej niż stała tolerancji. Z tego powodu silniki ponownie rozpoczną pracę nieskończenie zwiększając wartość błędu.

W związku z powyższym, została dodana flaga widoczna w spisie [6.8], zapewniająca poprawność działania kodu w każdych warunkach. Jeżeli silnik raz osiągną wartość założoną, flaga uniemożliwia ponowne wejście programu do wznowienia pracy silnika. Stan zmiennej binarnej zostaje zmieniony dopiero po otrzymaniu nowych instrukcji z komputera nadrzędnego.

7.2.2 Błąd odbierania i wysyłania poleceń

Podczas testów przeprowadzonych w fazie pierwszej, zaobserwowane zostało problematyczne zachowanie komunikacji między SBC Raspberry Pi a kontrolerem silników Arduninо UNO R3. Błąd polegał na zbyt wczesnym wysyłaniu kolejnych poleceń. Z pozoru łatwy do rozwiązania problem, okazał się nie tak trywialny.

Pierwsza wersja komunikacji polegała na stałym opóźnieniu z wykorzystaniem metody dostarczanej przez klasę *time - sleep()*. Metoda ta umożliwia uśpienie wątku na określony czas (w sekundach). To podejście ma bardzo poważną wadę - jest nieelastyczne względem wysyłanych komunikatów. Oznacza to, że niezależnie od odległości jaką robot ma przebyć, kolejna komenda zostanie wysłana na przykład po czterech sekundach.

Kolejnym podejściem, było zastosowanie pewnej flagi lub znacznika, który umożliwi określenie zakończenia pracy robota oraz wykorzystanie go jako warunku kontynuowania. W tym celu dotychczas istniejący kod został umieszczony wewnątrz pętli *while*, co umożliwiło poprawne oczekiwanie na ukończenie zadania.

Po zastosowaniu wyżej opisanego podejścia, opóźnienie było tak małe, że w warunkach rzeczywistych nieosiągalne, co zmusiło autora do dodania pewnego dodatkowego krótkiego opóźnienia między kolejnymi komendami.

7.2.3 Błąd dokładności jazdy

Poważnym problemem okazała się dokładność jazdy platformy mobilnej. Mimo, że względem wczesnych wersji algorytmów zostały poczynione znaczne postępy, robot nie osiągnął idealnej dokładności, w szczególności podczas obrotów.

Pierwsze wersje sterowania opierały się na obrocie względem układu współrzędnych, którego środek znajdował się w punkcie styku jednego koła z podłożem. Innymi słowy, robot obracał się o zadany kąt poruszając się tylko jednym kołem, co nie do końca spełniało wymaganie projektowe, jakim było zastosowanie napędu różnicowego. Dokładność osiągnięta w tej konfiguracji była jednak zadowalająca.

Po wprowadzeniu napędu różnicowego, pojawiły się większe błędy wynikające z mniej dokładnego zliczania impulsów o znaku ujemnym, a więc podczas ruchu kół do tyłu. Wymagania projektowe zostały spełnione, jednak kosztem dokładności, która we wczesnych wersjach była nienajlepsza.

Ostatecznie została podjęta decyzja o zlikwidowaniu jednego kanału enkodera. Ze względu na charakter sterowania robotem, rozróżnienie kierunku jazdy może zostać zrealizowane programowo, zgodnie z tym, co zostało zaprezentowane w kodzie [6.9]. Decyzja ta była spowodowana sposobem zliczania impulsów - za pomocą przerwań globalnych. Przy dużej ilości generowanych przerwań, szybkość mikrokontrolera mogła być niewystarczająca. Dzięki zredukowaniu zliczanych impulsów o połowę, wpływ późnienia wynikającego z dużej ilości generowanych przerwań również został zmniejszony.

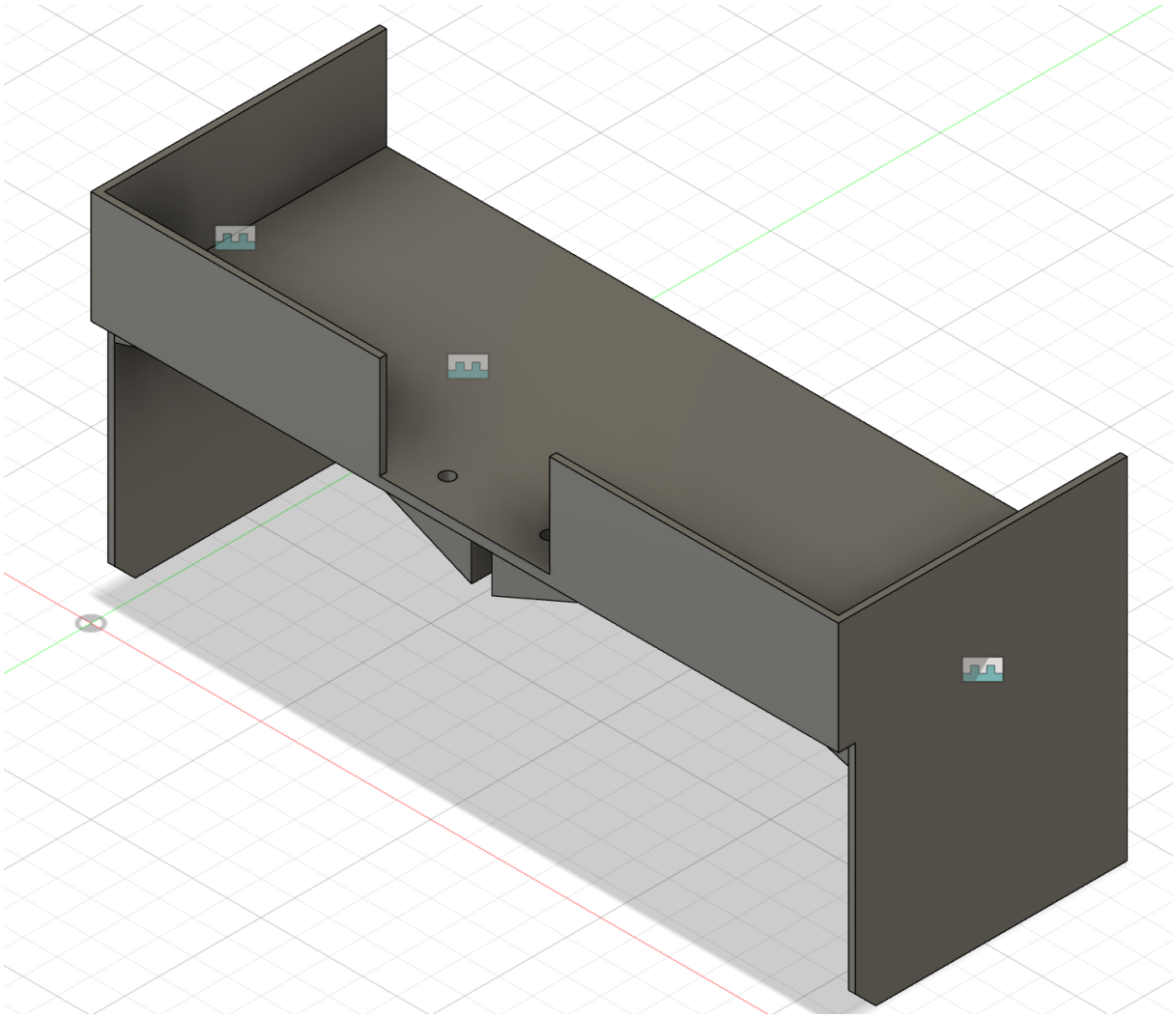
Szczególnie mocno różnica zarysowuje się w przypadku obrotów. Wynika to z faktu, że wcześniej z nieznanych do końca przyczyn, silniki obracały się o nieco mniejszy kąt do tyłu niż do przodu. Efekt pozostawał bez zmian nawet po próbach bezpośredniej ingerencji w postaci dodania stałej ilości impulsów w przypadku obrotu do tyłu lub zwiększenia wartości sygnału PWM. Zachowanie to może wynikać między innymi z jakości wykonania silnika.

Po zastosowaniu jednego kanału enkodera, opisane problemy zostały zmniejszone, a dokładność znacznie się poprawiła. Nie jest ona jednak idealna. Autor uważa, że największe różnice wprowadziłaby zmiana elementów wykonawczych - silników. Zastosowane silniki firmy Pololu stanowią solidną bazę do rozwoju budżetowych projektów, jednak w przypadku, gdy robot ma realnie funkcjonować niezawodnie w pewnym niekrótkim horyzoncie czasu, należy zastosować zdecydowanie bardziej dokładne silniki z mniejszymi luzami.

Dodatkowo ze względu na nieidealną dokładność, autor wprowadził koncepcję stacji dokujących [7.1] znajdujących się bezpośrednio przed podajnikiem oraz miejscami docelowymi. Umożliwiają one korekcję toru jazdy, tym samym niwelując błędy powstałe podczas pracy robota. Podobne podejście zostało wymienione w znanej pracy naukowców z Uniwersytetu w Michigan pod tytułem "Where am I", co po polsku znaczy "Gdzie jestem?"[11]. Wynika z niej, że podczas implementacji algorytmów sterowania opierających się jedynie na odczytach z enkoderów małe błędy, w przypadku powyższego projektu poniżej 1%, potęgują się w czasie, co uniemożliwia długofalową pracę bez wsparcia na przykład wcześniej wymienionych stacji dokujących.

7.2.4 Warunki początkowe

Ze względu na charakter pracy oraz sposób nawigacji oparty na uproszczonej odometrii przy użyciu enkoderów inkrementalnych, robot wykazuje dużą wrażliwość na warunki początkowe. Warunki te, definiujące początkowe położenie robota względem środowiska, mają kluczowe znaczenie dla dokładności jego działania. Nieprawidłowe ustawienie w pozycji bazowej może prowadzić do znaczących błędów w dalszej pracy, takich jak nieprecyzyjne podjechanie do stacji podajnikowej. Nawet niewielkie odchylenia, np. kilka stopni



Rysunek 7.1: Zdjęcie przedstawiające stację dokującą oraz miejsce docelowe

od linii zerowej, mogą skutkować krytycznymi problemami, takimi jak brak możliwości pobrania obiektu.

W celu minimalizacji tych błędów robot korzysta ze stacji dokujących przy punktach kontrolnych, które umożliwiają korekcję pozycji, jednak zakres ich działania jest ograniczony i nie rozwiązuje problemu dużych odchyleń. Rozwiązaniem tego problemu jest zastosowanie wyraźnego szablonu lub prowadnicy, który zapewni precyzyjne ustawienie robota w pozycji bazowej. Takie podejście znacząco zmniejsza ryzyko błędów wynikających z nieścisłości w ręcznym ustawianiu robota.

Rozdział 8

Podsumowanie i wnioski

8.1 Podsumowanie

Finalna wersja platformy mobilnej sortującej klocki zgodnie z ich kolorem spełnia wszystkie wymagania projektowe w sposób dostateczny. Po zastosowaniu dodatkowych szyn naprowadzających robot wykonuje swoją pracę w sposób dobry.

Mimo napotkanych trudności i błędów wymagania projektowe zostały spełnione. Bazując na zdobytym doświadczeniu kolejna generacja robota z całą pewnością zostałaby znacznie poprawiona, a sam czas prototypowania skróciłby się drastycznie.

W dalszej części rozdziału zostały opisane usprawnienia, które zdaniem autora warto wprowadzić w kolejnych iteracjach projektu.

8.2 Kierunki ewentualnych przyszłych prac

Platforma będąca przedmiotem niniejszej pracy umożliwia bardzo szeroki wachlarz możliwości dalszego rozwoju. Obecnie robot wykorzystuje podstawowe przetwarzanie informacji wizyjnej ze względu na charakter sortowanych przedmiotów. Rozpoznawanie kolorów wystarcza, aby robot pomyślnie określił gdzie należy umieścić klocek. W przypadku bardziej zaawansowanych przedmiotów, takich jak części produkcyjne lub paczki magazynowe opisane na przykład kodem lub specjalną etykietą, warto rozważyć wprowadzenie bardziej zaawansowanej formy wizji komputerowej z wykorzystaniem uczenia maszynowego. Dzięki dużym możliwościom obliczeniowym mikrokomputera Raspberry Pi, wdrożenie uczenia maszynowego nie będzie wymagało zmiany układu SBC.

Dodatkowym aspektem wizji komputerowej, który warto rozważyć jest wprowadzenie systemu bezpieczeństwa, wykrywającego przedmioty na swojej drodze oraz ludzi. Umożliwi to pracę platformy mobilnej w sposób bezpieczny w dynamicznym środowisku niekoniecznie odizolowanym.

Kolejnym ulepszeniem, tym razem bardziej wymagającym, jest zastosowanie techno-

logii mapowania środowiska pracy za pomocą metody SLAM z użyciem czujnika LIDAR. Robot za pomocą tego systemu mógłby autonomicznie obliczać trasę, co znacznie poprawiłoby jego elastyczność.

Następnym rozwiązaniem, które warto rozważyć jest migracja systemu kontroli robota do środowiska programistycznego ROS2 (ang. *Robot Operating System*). Jest to środowisko umożliwiające przystępne skalowanie projektu, dzięki jasno określonym zasadom komunikacji oraz dużej dostępności gotowych rozwiązań. Na przykład wyżej wymienione kwestie mapowania oraz autonomicznej nawigacji relatywnie prosto można wdrożyć wykorzystując biblioteki takie jak "slam_toolbox"oraz "Nav2". Są to rozwiązania dostarczone przez zespół programistyczny rozwijający środowisko ROS2.

Wielką zaletą tego środowiska jest jego głęboko zakorzenione nawiązanie do idei wolnego oprogramowania (ang. *open source*), wywodzącego się z jądra systemu operacyjnego jakim jest Linux. Oznacza to, że cała społeczność może bezpłatnie korzystać z rozwiązań dostarczonych nie tylko bezpośrednio przez producenta, ale również przez użytkowników.

Jeżeli migracja systemu kontroli robota istotnie miałaby miejsce, rozsądnym byłoby, aby umieścić aplikację wewnątrz kontenera Docker. Jest to kolejne środowisko programistyczne, umożliwiające bardzo dobrą spójność działania aplikacji, dzięki "spakowaniu"wszystkich zależności oraz bibliotek do kontenera. Dzięki temu zabiegowi aplikacja będzie działała w ten sam sposób na wszystkich urządzeniach mających zainstalowano Docker'a.

Ostatnim usprawnieniem, które zadaniem autora warto rozważyć jest rozwinięcie panelu operatorskiego jako aplikacji internetowej. Aktualna wersja aplikacji nie umożliwia wystarczającej kontroli na robotem oraz nad parametrami jego pracy. Rozwinięcie tego rozwiązania umożliwi znacznie bardziej przystępną obsługę robota, wraz z programowaniem nowych ścieżek lub nowych zadań.

Bibliografia

[1] Vaclav Krivanek Khac Lam Pham Alexandr Stefek Thuan Van Pham. „Energy Comparison of Controllers Used for a Differential Drive Wheeled Mobile Robot". W: *IEEE Access*. 2020, s. 13.

[2] apautomatyka.pl. *Zasada działania enkoderów magnetycznych i optycznych POSITAL*. URL: https://apautomatyka.pl/zasada-dzialania-enkoderow-magnetycznych-i-optycznych-posital/.

[3] 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.

[4] 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).

[5] 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.

[6] 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.

[7] John Canny. „A Computational Approach to Edge Detection". W: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (1986).

[8] B. Cottenceau. *Introduction to Arduino*.

[9] EBMiA. *EBMiA.pl*. 2023. URL: https://www.ebmia.pl/wiedza/porady/automatyka-porady/czujnik-halla-co-to-jest-jak-dziala-i-jak-sprawdzic/.

[10] encyklopedia.pwn.pl. *Halla zjawisko*. URL: https://encyklopedia.pwn.pl/haslo/;3909634.

[11] H.R. Everett J. Borenstein i L. Feng. *""Where am I?"Sensors and Methods for Mobile Robot Positioning"*. University of Michigan, 1996.

[12] Raspberry Pi Ltd. *The Picamera2 Library. A libcamera-based Python library for Raspberry Pi cameras.*

[13] SHRIPAD G. DESAI PRINCE KUMAR. „Interrupts in The Microcontroller". W: *IRE Journals* 4.4 (2021), s. 166–169.

[14] ReearchGate.net. *ResearchGate*. URL: https://www.researchgate.net/figure/a-RGB-Color-Space-7-b-YCbCr-Color-Space-8_fig1_298734907.

[15] ReearchGate.net. *ResearchGate*. URL: https://www.researchgate.net/figure/HSV-color-space-and-RGB-color-transformation_fig4_312678134.

[16] ReearchGate.net. *ResearchGate*. URL: https://www.researchgate.net/figure/YCbCr-Color-Space-4_fig3_357594587.

[17] Wikipedia.com. *Układ regulacji (automatyka)*. URL: https://pl.wikipedia.org/wiki/Uk%C5%82ad_regulacji_%28automatyka%29.

[18] Djemel Ziou i Salvatore Tabbone. „Edge Detection Techniques-An Overvie". W: *&#1085 / 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 stopnień przyleżności do zbioru

$\mathbb{E}$ zbiór krawędzi grafu

$\mathcal{L}$ transformata Laplace'a

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