# Politechnika Wrocławska

Wydział Elektroniki, Fotoniki i Mikrosystemów


KIERUNEK: Automatyka i Robotyka (AIR)

# PRACA DYPLOMOWA INŻYNIERSKA

TYTUŁ PRACY:

Mały robot mobilny do obserwacji otoczenia


AUTOR:

Said Hannoush

PROMOTOR:

Dr inż. Robert Muszyński,
Katedra Cybernetyki i Robotyki



WROCŁAW 2024

# Spis treści

# Rozdział 1

# Wstęp

Robotyka mobilna jest dziedziną nauki i techniki, która koncentruje się na projektowaniu, budowie oraz programowaniu systemów zdolnych do samodzielnego poruszania się i interakcji z otoczeniem. W projekcie mobilnego robota wyposażonego w systemy obserwacji oraz komunikacji kluczowe jest poprawne połączenie komponentów takich jak jednostka obliczeniowa, czujniki i moduły transmisji danych. Skuteczna integracja tych elementów pozwala na realizację zadań związanych z monitorowaniem otoczenia, wykrywaniem przeszkód oraz zdalnym sterowaniem robotem.

Zaprojektowanie systemu sterowania robota wymaga precyzyjnej konfiguracji sprzętu oraz implementacji algorytmów przetwarzania danych. Czujniki, takie jak kamera czy dalmierz, pozwalają na analizę środowiska i wykrywanie przeszkód, co jest podstawą do podejmowania decyzji przez operatora jak i tych autonomicznych. Moduły komunikacyjne, wspierane przez protokoły bezprzewodowe, zapewniają przesyłanie obrazu i danych w czasie rzeczywistym, co umożliwia użytkownikowi nadzór nad robotem nawet z dużej odległości.

Po wstępnych założeniach projektowych, które uwzględniały wyposażenie robota w zaawansowane funkcje autonomiczne, takie jak cykliczne poruszanie się po trajektorii, unikanie przeszkód, śledzenie obiektów czy samoczynne parkowanie, przeprowadzono analizę możliwości technicznych platformy bazowej. Wykorzystana platforma RC okazała się niewystarczająca do implementacji tych funkcji z uwagi na:

- duże rozmiary i niską zwrotność pojazdu,
- obecność poślizgu na osiach i ograniczoną precyzję sterowania silnikiem,
- konstrukcję dostosowaną bardziej do szybkiej jazdy wyścigowej niż kontrolowanej nawigacji.

W związku z tym podjęto decyzję o zmianie zakresu funkcji autonomicznych na bardziej realistyczne i praktyczne. Nowe funkcje obejmują:

- system antykolizyjny – zapobiegający zderzeniom przy różnych prędkościach,
- ograniczanie prędkości – w przypadku wykrycia przeszkody pojazd będzie stopniowo redukował prędkość maksymalną w miarę zbliżania się do niej.

Realizacja projektu wymaga nie tylko przemyślanego projektu sprzętowego, ale również oprogramowania, które synchronizuje wszystkie komponenty jak i pozwala operatorowi na wygodny wgląd w otoczenie robota oraz sterowanie nim.

W ramach realizacji projektu zaplanowano wykonanie następujących działań:

- Zaprojektowanie systemu sterowania, który integruje różne komponenty robota, w tym jednostkę obliczeniową, czujniki oraz moduły komunikacyjne.
- Implementacja algorytmów umożliwiających autonomiczne akcje robota, takie jak system antykolizyjny oraz ograniczanie prędkości w przypadku ciasnych pomieszczeń.
- Instalacja oraz konfiguracja kamery i czujników, które umożliwiają robotowi monitorowanie otoczenia.
- Wdrożenie niezawodnego systemu komunikacji z użytkownikiem w celu przesyłania danych z czujników oraz obrazu wideo w czasie rzeczywistym.
- Testowanie funkcjonalności i skuteczności robota w rzeczywistych warunkach.

W kolejnych rozdziałach pracy omówione zostaną kluczowe aspekty związane z realizacją projektu robota mobilnego. Rozdział drugi obejmuje przegląd i wybór odpowiednich technologii, komponentów sprzętowych oraz narzędzi programistycznych. W rozdziale trzecim opisano platformę sprzętową, w tym zastosowane elementy oraz ich parametry techniczne. Czwarty rozdział poświęcony jest tworzeniu oprogramowania, w tym implementacji systemu sterowania oraz aplikacji klienta. W piątym rozdziale zaprezentowano wyniki testów systemu, przeprowadzono analizę uzyskanych rezultatów oraz oceniono skuteczność przyjętych rozwiązań. Ostatni, szósty rozdział zawiera podsumowanie przeprowadzonych prac, wnioski końcowe oraz sugestie dotyczące dalszego rozwoju projektu.

# Rozdział 2

# Przegląd i dobór rozwiązań

W kontekście projektu zdalnie sterowanego pojazdu istotne są rozwiązania, które zapewniają niezawodną komunikację między jednostką sterującą a aplikacją użytkownika, umożliwiają przesyłanie obrazu w czasie rzeczywistym oraz integrację z czujnikami monitorującymi stan pojazdu. Szczególnie ważne są technologie zoptymalizowane do pracy na urządzeniach o ograniczonej mocy obliczeniowej oraz protokoły, które gwarantują szybki i stabilny przesył danych. W tym rozdziale przedstawiono przegląd dostępnych rozwiązań technologicznych i narzędzi, które mogą być zastosowane w projektach związanych z zdalnym sterowaniem, transmisją danych oraz strumieniowaniem wideo. Omówiono różne platformy sprzętowe, biblioteki programistyczne oraz protokoły komunikacyjne, wskazując na ich kluczowe zalety i ograniczenia.

## 2.1 Technologie do zdalnej komunikacji

W ramach realizacji projektu mobilnego robota do obserwacji otoczenia, kluczowym elementem jest wybór technologii zdalnej komunikacji, która pozwoli na przesyłanie obrazu i danych pomiędzy robotem a komputerem sterującym. Rozważono różne technologie komunikacji bezprzewodowej, takie jak kanałowa komunikacja radiowa typu simplex, Wi-Fi, sieć komórkowa oraz Bluetooth. Każde z tych rozwiązań ma swoje zalety i ograniczenia, które są istotne przy wdrażaniu systemów mobilnych.

Kanałowa komunikacja radiowa typu simplex, mimo że powszechnie stosowana w systemach telemetrycznych, wymaga szczegółowej konfiguracji oraz dodatkowych urządzeń nadawczo-odbiorczych [11]. Praca z falami radiowymi jest bardziej skomplikowana, ponieważ wymaga znajomości niskopoziomowych protokołów transmisji. Dodatkowo, takie rozwiązanie nie zawsze zapewnia, wystarczający do bezpiecznej komunikacji, poziom bezpieczeństwa, a jego przepustowość może być niewystarczająca do obsługi strumienia wideo.

Sieć komórkowa oferuje praktycznie nieograniczony zasięg transmisji, co jest jej największą zaletą, ale wiąże się z dodatkowymi kosztami abonamentu oraz koniecznością wyposażenia robota w moduł LTE lub 5G [10]. Integracja tego typu komunikacji bywa bardziej skomplikowana, a czasami może występować opóźnienie w transmisji danych, co jest istotne w systemach wymagających niskiej latencji.

Bluetooth wyróżnia się łatwością konfiguracji i niskim zużyciem energii, jednak jego ograniczony zasięg, wynoszący zazwyczaj do 10–15 metrów, ogranicza możliwości zastosowania w projektach, gdzie wymagana jest transmisja danych na większe odległości [11]. Ograniczona przepustowość Bluetooth sprawia również, że technologia ta nie nadaje się do przesyłania obrazu w czasie rzeczywistym.

Wi-Fi oferuje stosunkowo duży zasięg transmisji, sięgający do kilkudziesięciu metrów, oraz wysoką przepustowość, co umożliwia przesyłanie danych w czasie rzeczywistym, w tym obrazu wideo [10]. Dodatkowo, Wi-Fi charakteryzuje się łatwością konfiguracji i integracji z istniejącą infrastrukturą sieciową. Dzięki tym cechom znajduje szerokie zastosowanie w projektach, które wymagają stabilnej i wydajnej komunikacji.

### Dobór

Spośród rozważanych technologii komunikacji bezprzewodowej, Wi-Fi okazało się najlepszym wyborem dla mobilnego robota do obserwacji otoczenia. To rozwiązanie zapewnia optymalny kompromis między zasięgiem, przepustowością a łatwością integracji. Możliwość przesyłania obrazu wideo w czasie rzeczywistym oraz stabilność połączenia sprawiają, że Wi-Fi doskonale spełnia wymagania projektu, umożliwiając efektywną transmisję danych między robotem a komputerem sterującym. Dodatkowo, brak stałych kosztów, jak w przypadku sieci komórkowej, oraz znacznie większa funkcjonalność w porównaniu do Bluetooth czy komunikacji radiowej typu simplex, potwierdzają słuszność tego wyboru.

## 2.2 Jednostki sterujące i systemy operacyjne

Wybór jednostek sterujących oraz systemów operacyjnych jest kluczowym elementem w projektowaniu robotów zdalnie sterowanych, w tym tych mobilnych. Jednostka sterująca powinna zapewniać wystarczającą moc obliczeniową dla danego zadania, obsługę różnych komponentów (np. kamer, czujników) oraz umożliwiać komunikację z innymi urządzeniami [11]. Dodatkowo, system operacyjny musi być dostosowany do ograniczeń sprzętowych, takich jak pamięć RAM, procesor oraz zapotrzebowanie na energię [3]. W tym podrozdziale przeanalizujemy dostępne jednostki sterujące, takie jak mikrokontrolery i mikrokomputery, oraz systemy operacyjne, które charakteryzują się niskim zapotrzebowaniem na zasoby, co jest istotne w projektach o ograniczonej mocy obliczeniowej i pamięci [31, 19]. Celem tego przeglądu jest przedstawienie dostępnych opcji, które mogą być wykorzystane w realizacji projektu mobilnego robota, umożliwiającego zdalne sterowanie oraz transmisję danych.

### 2.2.1 Jednostki sterujące

Głównymi kryteriami wyboru jednostki sterującej dla mobilnego robota są analiza możliwości technicznych mikrokontrolerów oraz mikrokomputerów. Oba rozwiązania mają różne właściwości, które wpływają na funkcjonalność systemu, w tym obsługę kamery, zdalną komunikację oraz przetwarzanie danych.

#### Mikrokontrolery

Mikrokontrolery to urządzenia, które znajdują zastosowanie głównie w systemach wbudowanych oraz IoT, charakteryzujące się niskim zużyciem energii i kompaktowymi wymiarami. Do popularnych modeli należą m.in. **ESP32**, **STM32** oraz **Arduino Nano** [1]. ESP32 ma wbudowaną obsługę Wi-Fi i Bluetooth, co czyni go atrakcyjnym wyborem do komunikacji bezprzewodowej, jednak jego ograniczona moc obliczeniowa może być niewystarczająca do przetwarzania obrazu czy obsługi serwerów HTTP. Z kolei STM32, dzięki szerokiemu wachlarzowi modeli, sprawdza się w projektach wymagających wysokiej wydajności w czasie rzeczywistym (RTOS). Arduino Nano, mimo prostszej konstrukcji, doskonale nadaje się do aplikacji, które potrzebują podstawowej logiki sterującej oraz niskiego poboru mocy.

#### Mikrokomputery

Mikrokomputery, takie jak **Raspberry Pi**, **Odroid** czy **Banana Pi**, wyróżniają się możliwością uruchamiania pełnoprawnych systemów operacyjnych oraz obsługą bardziej zaawansowanych aplikacji. Przykłady to:

- **Raspberry Pi Zero 2 W**– kompaktowy, energooszczędny mikrokomputer, idealny do projektów mobilnych oraz posiadający wsparcie dla Wi-fi oraz Bluetooth [8].
- **Odroid C1**– mikrokomputer o nieco większych wymiarach, ale z lepszą wydajnością w porównaniu do Raspberry Pi Zero, co może być istotne przy przetwarzaniu obrazu w czasie rzeczywistym [21].
- **Banana Pi M2 Zero**– alternatywa dla Raspberry Pi, charakteryzująca się podobnymi wymiarami i wydajnością, z dodatkowymi opcjami konfiguracji sprzętowej [26].

W porównaniu do mikrokontrolerów, mikrokomputery zapewniają wyższą moc obliczeniową oraz większą elastyczność dzięki systemom operacyjnym, co umożliwia uruchamianie aplikacji takich jak serwery HTTP, analiza obrazu czy przetwarzanie danych z czujników.

### 2.2.2 Systemy operacyjne

W przypadku urządzeń o ograniczonych zasobach, takich jak Raspberry Pi Zero 2 czy inne mikrokomputery, kluczowe jest wybranie niewymagającego dużej ilości zasobów systemu operacyjnego, który zapewnia równowagę między funkcjonalnością a efektywnością wykorzystania zasobów.

#### Raspbian Lite

Raspbian Lite to uproszczona wersja systemu Raspbian, pozbawiona środowiska graficznego, co znacząco zmniejsza zapotrzebowanie na pamięć RAM i moc obliczeniową [9]. Dzięki wsparciu dla popularnych bibliotek oraz narzędzi konfiguracyjnych, takich jak `raspi-config`, jest często wykorzystywana w projektach opracowywanych na Raspberry Pi.

### Ubuntu Server

Ubuntu Server to wersja systemu Ubuntu, która działa bez interfejsu graficznego [3]. Dzięki szerokiemu wsparciu społeczności oraz bogatym repozytoriom, jest najlepszym rozwiązaniem dla bardziej zaawansowanych projektów, chociaż może wymagać więcej zasobów w porównaniu do Raspbian Lite.

### Tiny Core Linux

Tiny Core Linux to niezwykle lekki system operacyjny o minimalnym zapotrzebowaniu na zasoby, co czyni go idealnym wyborem dla mikrokomputerów z ograniczoną pamięcią [20]. Jego modularność pozwala na dostosowanie funkcji do specyficznych wymagań projektu, co jednak wiąże się z większym nakładem pracy podczas konfiguracji.

### FreeRTOS

FreeRTOS to system operacyjny czasu rzeczywistego (RTOS) stworzony z myślą o mikrokontrolerach, takich jak STM32 czy ESP32 [31]. Umożliwia efektywne zarządzanie zadaniami i obsługę.

### Alpine Linux

Alpine Linux to minimalistyczna dystrybucja Linuxa, znana z wysokiej wydajności oraz niskiego zużycia zasobów [19]. Dzięki wsparciu dla kontenerów Docker, może być wykorzystywana w projektach, które wymagają dużej elastyczności w zarządzaniu oprogramowaniem.

### Dobór

Na podstawie przeprowadzonej analizy, jako jednostkę sterującą wybrano **Raspberry Pi Zero 2 W** oraz system operacyjny **Raspbian Lite**. Raspberry Pi Zero 2 W zapewnia wystarczającą moc obliczeniową, kompaktowe wymiary oraz wbudowaną obsługę Wi-Fi i Bluetooth, co czyni go doskonałym wyborem do projektów mobilnych robotów. Jego energooszczędność oraz kompatybilność z szeroką gamą oprogramowania pozwalają na realizację złożonych zadań, takich jak przetwarzanie danych z czujników czy transmisja obrazu.

Raspbian Lite, dzięki swojej lekkości i braku interfejsu graficznego, minimalizuje zużycie zasobów systemowych, co jest szczególnie ważne w projektach o ograniczonej pamięci i mocy obliczeniowej [31]. System ten oferuje wsparcie dla popularnych bibliotek i narzędzi, co ułatwia integrację komponentów i rozwój oprogramowania. Wybór tej kombinacji zapewnia optymalną równowagę między wydajnością a efektywnością energetyczną, co jest kluczowe w realizacji projektu mobilnego robota.

## 2.3 Sposoby tworzenia oprogramowania

Tworzenie oprogramowania, szczególnie w językach takich jak C++, wymaga podejścia dostosowanego do specyfiki urządzeń i środowiska pracy. W zależności od dostępnych zasobów sprzętowych i systemowych, stosuje się różne metody, które różnią się poziomem złożoności konfiguracji i wydajnością. Poniżej przedstawiono przegląd najczęściej wykorzystywanych sposobów tworzenia oprogramowania.

### 2.3.1 Kompilacja lokalna

Kompilacja lokalna to najprostsze podejście, w którym cały proces tworzenia i budowania oprogramowania odbywa się bezpośrednio na urządzeniu docelowym [23, 32]. Jest to wygodne rozwiązanie w przypadku komputerów i urządzeń o wysokich zasobach, takich jak procesor i pamięć RAM. Jednak w przypadku systemów o ograniczonych zasobach, takich jak małe komputery jednopłytkowe, proces ten może być nieefektywny lub wręcz niemożliwy ze względu na długi czas kompilacji i problemy z obsługą dużych bibliotek.

### 2.3.2 Kompilacja zdalna

Kompilacja zdalna polega na korzystaniu z zasobów zewnętrznego komputera, który pełni funkcję serwera kompilacji [12, 6]. Proces ten umożliwia szybkie budowanie aplikacji na urządzeniach o niskich zasobach, które nie byłyby w stanie samodzielnie przeprowadzić tego zadania. Zdalne środowiska kompilacji, takie jak Visual Studio Code, umożliwiają bezpośrednią synchronizację kodu źródłowego i kompilację na zdalnym serwerze.

### 2.3.3 Kompilacja skrośna

Kompilacja skrośna jest to proces, w którym oprogramowanie jest budowane na jednym urządzeniu (np. komputerze PC) z myślą o uruchomieniu go na innym, które ma inną architekturę procesora [7, 15]. To rozwiązanie jest szczególnie popularne w przypadku urządzeń z procesorami ARM, takich jak Raspberry Pi, zwłaszcza w środowiskach o ograniczonych zasobach. Aby przeprowadzić kompilację skrośną, konieczne jest użycie zestawu narzędzi (toolchaina), który jest skonfigurowany dla docelowej architektury.

### Dobór

W ramach realizacji projektu mobilnego robota zdecydowano się na zastosowanie **kompilacji skrośnej (cross-kompilacji)**, która umożliwia tworzenie oprogramowania na komputerze hosta (np. PC) w celu jego uruchomienia na docelowym urządzeniu o innej architekturze, takim jak Raspberry Pi Zero 2. Wybór tej metody jest uzasadniony ograniczonymi zasobami docelowego mikrokomputera, co czyni lokalną kompilację czasochłonną i nieefektywną.

Do zarządzania procesem kompilacji wybrano **CMake**, który dzięki swojej wszechstronności i szerokiemu wsparciu dla kompilacji krzyżowych jest idealnym narzędziem w kontekście tego projektu. CMake pozwala na łatwe zdefiniowanie środowiska kompilacji, zarządzanie zależnościami oraz integrację bibliotek, co jest kluczowe w systemach z wieloma komponentami, takimi jak kamery, czujniki czy moduły komunikacyjne. Dzięki czytelnym plikom konfiguracyjnym i możliwości dostosowania do różnych platform, CMake znacznie upraszcza proces konfiguracji i wdrażania oprogramowania na docelowym urządzeniu. Taki wybór zapewnia wydajność oraz spójność w budowie systemu.

## 2.4 Biblioteki do tworzenia aplikacji graficznych

W tworzeniu graficznych aplikacji klienta, które umożliwiają wyświetlanie obrazu z kamery oraz komunikację sieciową, dostępnych jest wiele bibliotek [2, 4, 29, 35]. Poniżej przedstawiamy przegląd wybranych rozwiązań, które mogą być wykorzystane w takich projektach.

### 2.4.1 Qt Widgets

Qt Widgets to klasyczna biblioteka wchodząca w skład frameworka Qt, napisana w C++ [2, 4]. Umożliwia tworzenie aplikacji z graficznym interfejsem użytkownika (GUI), a jej komponenty pozwalają na łatwe implementowanie standardowych widżetów, takich jak przyciski, pola tekstowe czy okna dialogowe. Dodatkowo, Qt Widgets zapewnia obsługę komunikacji sieciowej oraz możliwość wyświetlania obrazu z kamery, co czyni ją wszechstronnym narzędziem dla aplikacji desktopowych. Dzięki wydajności i bogatej dokumentacji, biblioteka ta jest często wybierana w projektach wymagających tradycyjnego GUI.

### 2.4.2 Qt Quick

Qt Quick to nowoczesne podejście do tworzenia graficznych interfejsów użytkownika, również w ramach frameworka Qt [5]. Wykorzystuje język QML, który jest deklaratywny i umożliwia łatwe definiowanie interfejsów użytkownika, w tym animacji oraz elementów interaktywnych. Qt Quick jest szczególnie przydatne w aplikacjach przeznaczonych na urządzenia mobilne, takie jak smartfony i tablety, ponieważ wspiera responsywność oraz adaptacyjność interfejsów. Biblioteka ta pozwala także na komunikację sieciową oraz wyświetlanie obrazu z kamery, co czyni ją atrakcyjnym wyborem w projektach multimedialnych.

### 2.4.3 GTK+

GTK+ (GIMP Toolkit) to biblioteka open-source do tworzenia graficznych interfejsów użytkownika, wykorzystywana głównie w środowiskach z systemem Linux, choć dostępna jest także na inne platformy [29, 28]. GTK+ jest napisane w języku C, ale oferuje wiązania do innych języków, takich jak Python czy C++. Umożliwia

budowę aplikacji z szerokim zestawem widżetów, a także obsługę komunikacji sieciowej. W przypadku strumieniowania obrazu z kamery, można użyć dodatkowych bibliotek, takich jak GStreamer, które dobrze integrują się z GTK+.

### 2.4.4 FLTK

FLTK (Fast Light Toolkit) to lekka biblioteka GUI napisana w C++, zaprojektowana z myślą o prostocie i wydajności [35]. Dzięki niewielkiemu rozmiarowi i ograniczonym zależnościom, FLTK jest szczególnie skuteczna w przypadku aplikacji, które muszą działać na urządzeniach o ograniczonych zasobach. Oferuje podstawowe widżety, ale nie posiada wbudowanego wsparcia dla zaawansowanych funkcji, takich jak komunikacja sieciowa czy strumieniowanie obrazu z kamery. Niemniej jednak można ją rozszerzyć o dodatkowe biblioteki, aby sprostać takim wymaganiom.

### Dobór

Spośród omówionych bibliotek, do realizacji projektu wybrano **Qt Widgets**. Jest to narzędzie oferujące gotowe wsparcie dla klasycznego GUI, obsługę obrazu z kamery oraz komunikacji sieciowej, co idealnie spełnia wymagania projektu. Dodatkowo, biblioteka ta zapewnia wysoką wydajność, stabilność oraz bogatą dokumentację, co ułatwia szybkie i efektywne tworzenie aplikacji.

## 2.5 Biblioteki do obsługi kamery

W aplikacjach napisanych w języku C++ dostępnych jest wiele bibliotek do obsługi kamer, które różnią się poziomem zaawansowania, wsparciem dla różnych urządzeń oraz możliwościami przetwarzania obrazu [36, 37, 25, 28]. Poniżej przedstawiamy przegląd kilku popularnych bibliotek, które można wykorzystać do sterowania kamerą.

### 2.5.1 OpenCV

OpenCV (Open Source Computer Vision Library) to zaawansowana i wszechstronna biblioteka do przetwarzania obrazów oraz wizji komputerowej. Oferuje funkcje do przechwytywania obrazu z kamery, przetwarzania klatek, analizy ich oraz konwersji formatów. Jest to biblioteka wysokopoziomowa, co sprawia, że jest łatwa w użyciu, zwłaszcza w projektach wymagających szybkiej implementacji funkcji związanych z kamerą i obrazem [36]. Należy jednak pamiętać, że OpenCV wymaga, aby kamera obsługiwała formaty pikseli wspierane przez bibliotekę, co może być ograniczeniem w przypadku niektórych urządzeń. Mimo to, OpenCV cieszy się dużą popularnością dzięki bogatej dokumentacji oraz czynnemu rozwojowi.

### 2.5.2 Libcamera

Libcamera to niskopoziomowa biblioteka stworzona do obsługi kamer wbudowanych w nowoczesne urządzenia, takie jak Raspberry Pi. Stanowi podstawę dla

nowych narzędzi, takich jak `rpicam`, które zastępują starsze rozwiązania, takie jak `raspicam`. Libcamera zapewnia elastyczność oraz precyzyjną kontrolę nad parametrami kamery, takimi jak ekspozycja, balans bieli czy format obrazu [37]. Jest szczególnie polecana do zastosowań na urządzeniach z ograniczonymi zasobami lub niestandardowym sprzętem. Należy jednak pamiętać, że korzystanie z Libcamera wiąże się z większym nakładem pracy, ponieważ jest to biblioteka niskopoziomowa, co oznacza konieczność ręcznej konfiguracji i obsługi formatów obrazu.

### 2.5.3 Video4Linux (V4L2)

Video4Linux (V4L2) to niskopoziomowe API dla systemów Linux, które umożliwia sterowanie urządzeniami wideo, w tym kamerami. Biblioteka ta jest dobrze wspierana w systemach Linux i pozwala na przechwytywanie strumieni wideo oraz konfigurację parametrów urządzenia, takich jak rozdzielczość czy format obrazu [25]. V4L2 stanowi fundament dla wielu innych bibliotek, w tym Libcamera, i może być używana bezpośrednio w aplikacjach wymagających pełnej kontroli nad sprzętem. Warto jednak zauważyć, że jest trudniejsza w użyciu niż rozwiązania wysokopoziomowe, takie jak OpenCV.

### 2.5.4 GStreamer

GStreamer to biblioteka multimedialna, która umożliwia obsługę strumieni wideo i audio, w tym integrację z kamerami. Wspiera różne formaty i kodeki, co czyni ją przydatną w aplikacjach, które wymagają przesyłania lub przetwarzania obrazu w czasie rzeczywistym [28]. GStreamer jest elastyczny i modularny, co pozwala na budowanie zaawansowanych potoków przetwarzania danych multimedialnych. Dzięki dodatkom można go używać do przechwytywania obrazu z kamery oraz przesyłania go do aplikacji klienckiej.

### 2.5.5 Dobór

W ramach realizacji projektu podjęto decyzję o zastosowaniu narzędzia **libcamera-vid** do obsługi kamery, ze względu na jego prostotę oraz pełną zgodność z platformą Raspberry Pi, co czyni je najlepszym wyborem. Narzędzie to umożliwia bezpośrednie przesyłanie obrazu bez konieczności tworzenia skomplikowanych mechanizmów transmisji, co znacząco upraszcza proces integracji systemu [37].

W porównaniu z bardziej rozbudowanymi rozwiązaniami, takimi jak **Libcamera** czy **OpenCV**, które wymagają dodatkowej konfiguracji i programowania w zakresie zarządzania obrazem oraz przesyłania strumieniowego, **libcamera-vid** zapewnia szybkie i sprawne przesyłanie obrazu przy minimalnym nakładzie pracy technicznej. Dzięki temu wybór ten sprzyja zwiększeniu wydajności oraz skróceniu czasu tworzenia systemu.

## 2.6 Biblioteki do tworzenia serwera HTTP

Tworzenie serwera HTTP wymaga zastosowania narzędzi, które uproszczą implementację oraz zapewnią wydajność i elastyczność [34, 18, 27, 38]. W tym celu dostępne są zarówno lekkie biblioteki niskopoziomowe, jak i bardziej rozbudowane rozwiązania wysokopoziomowe. Poniżej przedstawiono cztery popularne opcje.

### 2.6.1 Crow

Crow to nowoczesna biblioteka wysokopoziomowa, która ułatwia budowę serwerów HTTP dzięki swojej strukturze *single-header* [34]. Obsługuje routing, asynchroniczne żądania oraz JSON, co czyni ją idealnym wyborem do prostych i średnio zaawansowanych aplikacji. Ze względu na swoją prostotę, często jest wybierana przez początkujących programistów oraz w projektach z ograniczonym czasem realizacji.

### 2.6.2 Boost.Beast

Boost.Beast jest częścią zestawu bibliotek Boost i oferuje zaawansowane możliwości obsługi HTTP oraz WebSocket [18]. Wykorzystuje mechanizmy asynchronicznego wejścia/wyjścia z `Boost.Asio`, co pozwala na efektywną obsługę dużej liczby żądań w czasie rzeczywistym. Biblioteka ta została stworzona dla projektów wymagających dużej elastyczności oraz skalowalności.

### 2.6.3 microhttpd

`microhttpd` to lekka biblioteka napisana w C, która umożliwia tworzenie serwerów HTTP o bardzo małym zużyciu zasobów [27]. Jest to rozwiązanie niskopoziomowe, wymagające ręcznej konfiguracji protokołu HTTP i zarządzania pamięcią, ale doskonale sprawdza się w aplikacjach wbudowanych, gdzie kluczowe są wydajność i minimalne wymagania sprzętowe.

### 2.6.4 Cpp-HTTP-Library

Cpp-HTTP-Library to kolejna opcja dla programistów C++, koncentrująca się na łatwości użycia i szybkim wdrożeniu serwera HTTP [38]. Obsługuje zarówno żądania synchroniczne, jak i asynchroniczne, a także integrację z JSON. Dzięki dobrej dokumentacji i wsparciu społeczności, jest chętnie wykorzystywana w średnio zaawansowanych projektach.

### 2.6.5 Dobór biblioteki do tworzenia serwera HTTP

Do realizacji projektu wybrano bibliotekę **Crow** do tworzenia serwera HTTP. Crow jest nowoczesną, wysokopoziomową biblioteką, która oferuje prostą implementację serwera HTTP, co znacząco ułatwia realizację funkcjonalności związanej z przesyłaniem obrazu z kamery przez HTTP. Dzięki swojej strukturze *single-header*, Crow jest łatwa do integracji, a jej składnia jest czytelna, co pozwala na szybkie

wdrożenie i minimalizację błędów w kodzie [34]. Ponadto, obsługuje asynchroniczne żądania i routowanie, co umożliwia jej obsługę większej liczby równoczesnych zapytań w aplikacjach wbudowanych. W kontekście tego projektu Crow oferuje najprostszą drogę do implementacji funkcjonalności serwera HTTP, umożliwiając szybkie i efektywne zarządzanie komunikacją sieciową.

## 2.7 Biblioteki do sterowania GPIO oraz komunikacji z czujnikami

W dzisiejszych czasach urządzenia IoT i systemy wbudowane często potrzebują efektywnego zarządzania pinami GPIO oraz komunikacji z różnorodnymi czujnikami [30, 16, 22, 17, 13]. Dzięki bogatej ofercie bibliotek w języku C++, możliwe jest skuteczne zarządzanie tymi funkcjami, niezależnie od konkretnego urządzenia.

### 2.7.1 Sterowanie GPIO

GPIO (General Purpose Input/Output) pozwala na kontrolowanie urządzeń zewnętrznych, takich jak przekaźniki, diody LED, przyciski czy serwomechanizmy. Istnieje wiele bibliotek C++ do obsługi GPIO, które różnią się poziomem skomplikowania i łatwością użycia:

- **WiringPi** – Popularna, choć obecnie przestarzała biblioteka, która ułatwia obsługę GPIO oraz podstawowych funkcji PWM (Pulse Width Modulation) [30]. Jest ceniona za prostotę i szybkość prototypowania.
- **pigpio** – Zaawansowana biblioteka do precyzyjnego sterowania GPIO [16]. Oferuje funkcje takie jak generowanie sygnałów PWM, obsługę sygnałów o wysokiej częstotliwości oraz asynchroniczną kontrolę pinów.
- **RPi.GPIO** – Wrapper w C++ dla popularnej biblioteki Pythonowej [22]. Charakteryzuje się prostą konfiguracją i podstawowymi funkcjami do sterowania GPIO, ale jest mniej elastyczna w porównaniu do pigpio.

### 2.7.2 Komunikacja z czujnikami

W interakcji z czujnikami kluczowe są protokoły I2C, SPI i UART. I2C pozwala na komunikację z wieloma urządzeniami na jednej magistrali, SPI oferuje szybszą transmisję danych, a UART jest wykorzystywany w prostych aplikacjach szeregowych. Dzięki bibliotekom C++, takim jak `Wire` (I2C), `SPI` (SPI) i `Serial` (UART), integracja tych protokołów w projektach staje się łatwa i efektywna.

#### I2C (Inter-Integrated Circuit)

Protokół I2C umożliwia podłączenie wielu urządzeń do tych samych pinów komunikacyjnych. Do obsługi I2C w C++ wykorzystuje się m.in.:

- `wiringPiI2C` – Prosta biblioteka umożliwiająca łatwą komunikację z urządzeniami I2C [30].
- `SMBus` – Rozszerzenie protokołu I2C, które oferuje większą kontrolę i bezpieczeństwo przesyłanych danych [13].

#### SPI (Serial Peripheral Interface)

SPI to szybki protokół komunikacji szeregowej, stosowany głównie w połączeniach z urządzeniami o dużej przepustowości, jak wyświetlacze czy przetworniki ADC. Biblioteki wspierające SPI to m.in.:

- `wiringPiSPI` – Lekka biblioteka do szybkiej konfiguracji połączeń SPI [30].
- `bcm2835` – Bardziej zaawansowana biblioteka z dodatkowymi opcjami konfiguracyjnymi dla SPI [22].

#### UART (Universal Asynchronous Receiver-Transmitter)

UART to protokół asynchronicznej komunikacji szeregowej, wykorzystywany do połączeń z modułami Bluetooth, GPS czy terminalami szeregowymi. Do jego obsługi służą m.in.:

- `wiringSerial` – Prosta biblioteka do podstawowej obsługi połączeń UART [30].
- `Boost.Asio` – Uniwersalna biblioteka umożliwiająca asynchroniczną obsługę portów szeregowych i bardziej zaawansowane zarządzanie komunikacją [17].

### 2.7.3 Komunikacja analogowa i cyfrowa z czujnikami

Nie wszystkie czujniki obsługują protokoły I2C, SPI czy UART. W przypadku sygnałów analogowych konieczne jest zastosowanie przetworników analogowo-cyfrowych (ADC), natomiast sygnały cyfrowe mogą być bezpośrednio przetwarzane przez GPIO.

#### Sygnały analogowe

Sygnały analogowe muszą być przekształcone na formę cyfrową przed dalszym przetwarzaniem. Oto przykłady przetworników ADC oraz towarzyszących im bibliotek:

- `MCP3008.h` – Biblioteka do obsługi przetwornika MCP3008, która korzysta z protokołu SPI [13].
- `Adafruit ADS1X15` – Biblioteka do precyzyjnego przetwarzania sygnałów analogowych z użyciem przetwornika ADS1115 przez interfejs I2C [13].

### 2.7.4 Dobór

W projekcie wybrano bibliotekę `pigpio` ze względu na jej wszechstronność i wydajność w obsłudze interfejsów komunikacyjnych, takich jak I2C, SPI, UART oraz GPIO. Zapewnia precyzyjne sterowanie sygnałami cyfrowymi i PWM, co jest kluczowe w aplikacjach wymagających dokładnej kontroli czasowej. Dzięki niskopoziomowemu dostępowi do zasobów sprzętowych Raspberry Pi, `pigpio` pozwala na efektywną integrację urządzeń peryferyjnych. Obszerna dokumentacja i aktywne wsparcie społeczności dodatkowo przemawiały za jej wyborem [16].

# Rozdział 3

# Platforma sprzętowa

W tym rozdziale przedstawiono elementy sprzętowe wykorzystane w realizacji projektu zdalnie sterowanego pojazdu który widać na rysunku 3.1, który ma funkcję strumieniowania obrazu oraz monitorowania danych z czujników. Kluczowym elementem platformy sprzętowej jest **Raspberry Pi Zero 2 W**, pełniące rolę centralnej jednostki sterującej. Dzięki niewielkim wymiarom i energooszczędności, urządzenie to doskonale sprawdza się w pojazdach o ograniczonej przestrzeni oraz wymaganiach dotyczących zużycia energii.

Projekt zakłada zdalne sterowanie pojazdem za pomocą aplikacji komputerowej, która komunikuje się z serwerem HTTP uruchomionym na Raspberry Pi. Jednostka sterująca zapewnia odbiór komend od użytkownika, przesyłanie strumienia wideo z kamery w czasie rzeczywistym oraz odczyt i przesyłanie danych z czujników zamontowanych na pojeździe. W tym celu wykorzystano dedykowane moduły i komponenty sprzętowe, które współpracują z Raspberry Pi, umożliwiając realizację kluczowych funkcji systemu, co jest widoczne na schemacie 3.2.

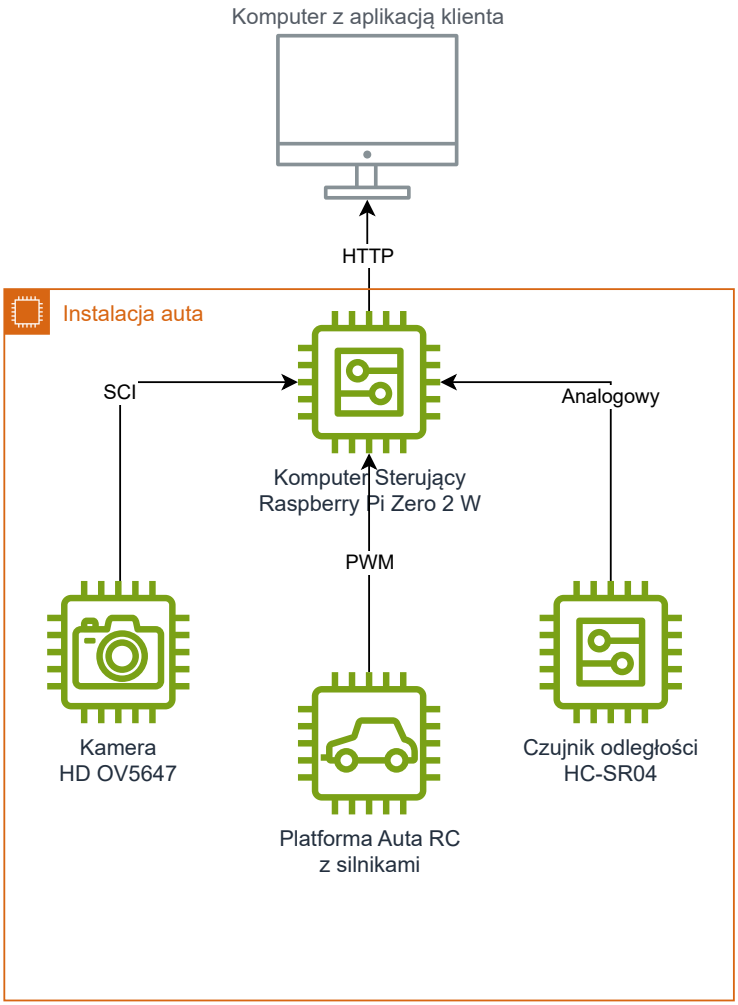
W rozdziale omówiono szczegółowo wybór jednostki centralnej oraz pozostałe elementy platformy sprzętowej, takie jak moduł kamery, czujniki, interfejsy komunikacyjne oraz platformę auta. Opisano również sposób integracji tych elementów oraz ich rolę w zapewnieniu płynnej i niezawodnej pracy pojazdu w kontekście projektu.

## 3.1 Jednostka sterująca – Raspberry Pi Zero 2 W

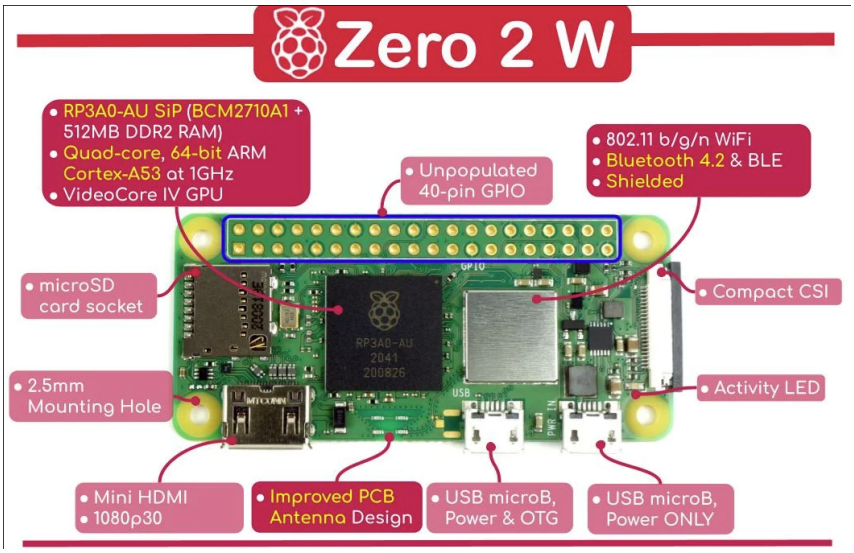
Raspberry Pi Zero 2 W to kompaktowy komputer jedno-płytkowy, który mimo niewielkich rozmiarów oferuje funkcjonalności umożliwiające realizację bardziej rozbudowanych projektów [8]. Stanowi ono ulepszoną wersję swojego poprzednika, Raspberry Pi Zero W. Jest zoptymalizowany pod kątem zadań wymagających niskiego poboru mocy i małych rozmiarów, oferując jednocześnie wysoką wydajność.

- **Procesor i wydajność:** Komputer ten posiada czterordzeniowy procesor ARM Cortex-A53, który zapewnia wysoką wydajność przy niskim zużyciu energii. Dzięki temu jest idealnym rozwiązaniem do prostych projektów IoT oraz wbudowanego sterowania.


Rysunek 3.1 Zdjęcie kompletnego robota




Rysunek 3.2 Diagram komponentów projektu



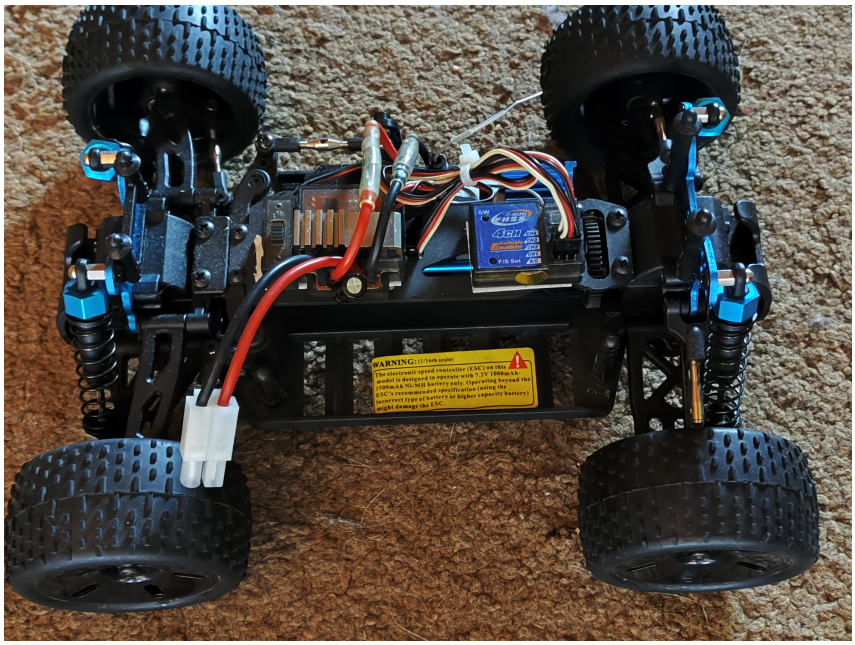

Rysunek 3.3 Raspberry Pi Zero 2 W

- **Pamięć RAM:** Urządzenie zostało wyposażone w 512 MB pamięci RAM, co pozwala na obsługę aplikacji o niewielkich wymaganiach sprzętowych. Taka ilość pamięci umożliwia realizację podstawowych procesów sterujących oraz komunikacyjnych.
- **GPIO:** Raspberry oferuje 40-pinowy interfejs GPIO, co umożliwia łatwą interakcję z zewnętrznymi urządzeniami i czujnikami. Dzięki temu jest wszechstronny w zastosowaniach, takich jak kontrola LED, przekaźników, a także bardziej zaawansowanych modułów komunikacyjnych.
- **Obsługa kamery i wyjścia wideo:** Dzięki złączu CSI (Camera Serial Interface) płytka ta pozwala na podłączenie kamery, co jest przydatne w projektach monitoringu, rozpoznawania obrazów i innych zastosowaniach związanych z wizualizacją.
- **Łączność:** Obsługuje ono moduły Wi-Fi oraz Bluetooth, co pozwala na bezprzewodową komunikację z innymi urządzeniami i integrację w systemach IoT.

Dokładniejszy opis wszystkich elementów płytki widać na obrazku 3.3.

## 3.2 Platforma auta RC

Platforma robota mobilnego wykorzystana w niniejszym projekcie jest konstrukcją zastaną, pochodzącą z modelu samochodu zdalnie sterowanego (RC), jest widoczna na rysunku 3.4. Składa się z solidnej podstawy mechanicznej z czterema kołami oraz układu napędowego z wbudowanym systemem sterowania.



Rysunek 3.4 Platforma auta RC

### 3.2.1 Budowa mechaniczna

Platforma posiada cztery koła, z których dwa przednie są skrętne, umożliwiając precyzyjne sterowanie pojazdem. Wszystkie koła są zamocowane na elementach zawieszenia, co zwiększa stabilność jazdy na nierównych powierzchniach. Mechanizm skrętu przednich kół jest realizowany za pomocą serwomechanizmu (servo), który zmienia ich kąt w zależności od sygnału sterującego.

### 3.2.2 Układ napędowy

Napęd na cztery koła jest realizowany za pomocą pojedynczego szczotkowego silnika elektrycznego, który umożliwia ruch pojazdu do przodu oraz do tyłu. Napęd przenoszony jest na wszystkie koła, co zwiększa przyczepność i poprawia zdolności terenowe.

### 3.2.3 Układ sterowania

Platforma posiada wbudowany układ sterowania bazujący na mostkach H, który umożliwia kontrolę zarówno silnika napędowego, jak i serwomechanizmu skrętu kół. Sterowanie odbywa się za pomocą dwóch wejść PWM o częstotliwości 50 Hz:

- **PWM dla silnika napędowego:**
  - Wypełnienie 5% – 7,5%: pojazd jedzie do tyłu.
  - Wypełnienie 7,5% – 10%: pojazd jedzie do przodu.
- **PWM dla serwomechanizmu skrętu:**
  - Wypełnienie 5% – 7,5%: skręt kół w lewo.

– Wypełnienie 7,5% – 10%: skręt kół w prawo.

Platforma jest zasilana napięciem 7,2 V, co jest typowym standardem dla modeli samochodów RC. Umożliwia to łatwe znalezienie kompatybilnych akumulatorów i ładowarek, co jest istotne z punktu widzenia użytkowania i konserwacji systemu.

### 3.2.4 Podsumowanie

Wykorzystanie gotowej platformy samochodu RC znacznie przyspiesza proces projektowania robota mobilnego, pozwalając skupić się na pozostałych aspektach projektu. Zintegrowane elementy mechaniczne i elektryczne, takie jak układ napędowy, serwomechanizm oraz mostki H, umożliwiają płynną integrację z jednostką sterującą i dalszy rozwój projektu.

## 3.3 Zasilanie

W projekcie zastosowano schemat zasilania przedstawiony na rysunku 3.5. Do zasilania wykorzystano baterię o napięciu 7.2 V i pojemności 5000 mAh 3.6.

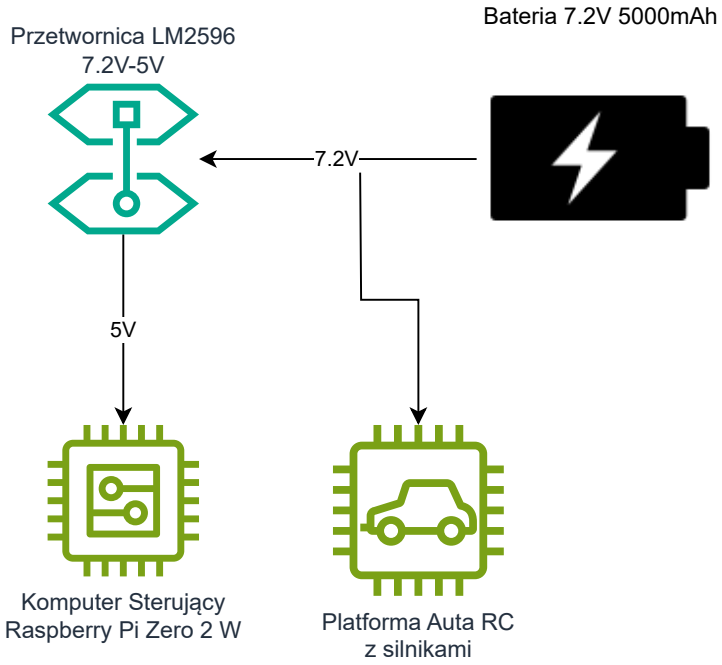
Bateria ta zasila zarówno platformę samochodu RC, która akceptuje bezpośrednio napięcie 7.2 V, jak i komputer jednopłytkowy Raspberry Pi Zero 2 W, który wymaga napięcia 5 V. Aby dostosować do niego napięcie, użyto przetwornicy step-down LM2596 3.7 [14] o zakresie napięcia wejściowego 3,2V-35V i maksymalnym prądzie wyjściowym 3A. Przetwornica LM2596 została skonfigurowana do obniżenia napięcia z 7.2 V do stabilnych 5 V, zapewniając wymagane zasilanie dla komputera. Dzięki temu obie części systemu mogą być efektywnie zasilane z jednej baterii, minimalizując potrzebę dodatkowych źródeł zasilania.

## 3.4 Czujnik odległości

Aby zwiększyć możliwości obserwacji otoczenia samochodu RC, został on wyposażony w ultradźwiękowy czujnik odległości HC-SR04 [33] 3.8. Jest to popularny czujnik, wykorzystywany w wielu projektach robotycznych i IoT do pomiaru odległości. Jego działanie polega na emisji fali dźwiękowej o wysokiej częstotliwości oraz pomiarze czasu, jaki zajmuje dotarcie oraz powrót fali dźwiękowej do czujnika. Na podstawie tego czasu system oblicza odległość do obiektu.

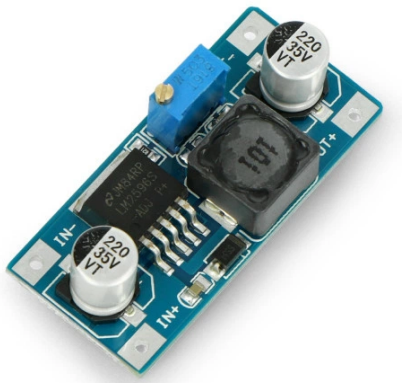
Specyfikacja techniczna

- Zakres pomiaru: 2 cm – 200 cm,
- Częstotliwość ultradźwięków: 40 kHz,
- Napięcie zasilania: 5 V DC,
- Interfejs: cyfrowy, z pinami `Trigger` (do inicjalizacji sygnału) oraz `Echo` (do odbioru sygnału zwrotnego),
- Dokładność pomiaru: ±3 mm.

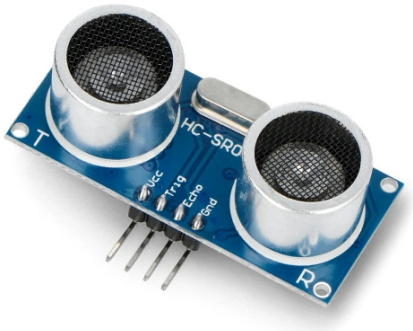



Rysunek 3.5 Diagram systemu zasilania


Rysunek 3.6 Bateria 7.2V 5000mAh




Rysunek 3.7 Przetwornica step-down LM2596 3,2V-35V 3A




Rysunek 3.8 Ultradźwiękowy czujnik odległości HC-SR04 2-200cm


Rysunek 3.9 Kamera OdSeven HD OV5647 5Mpx

**Zastosowanie w projekcie**

Czujnik HC-SR04 został zainstalowany z przodu pojazdu, co umożliwia wykrywanie przeszkód na drodze samochodu RC. Dane z czujnika są wykorzystywane do obliczeń, które pozwalają na:

- implementacje algorytmów antykolizyjnych oraz ograniczających prędkość w ciasnych pomieszczeniach,
- wgląd operatora w obecną odległość od przeszkody, szybciej niż w przypadku kamery, która podlega większym opóźnieniom.

## 3.5 Kamera

W projekcie wykorzystano kamerę OdSeven HD OV5647 5Mpx 3.9 dla Raspberry Pi [24]. Jest to kamera o wysokiej rozdzielczości, wykorzystująca moduł OV5647. Znajduje ona zastosowanie w różnych dziedzinach, od systemów wideo po projekty związane z rozpoznawaniem obrazów. Kamera jest zamontowana za pomocą złącza CSI (Camera Serial Interface) i wymaga kabla o wejściu kompatybilnym z Raspberry Pi. Adapter 300mm ułatwia podłączenie kamery do Raspberry Pi Zero 2 które posiada nieco mniejsze gniazdo.

Specyfikacja kamery

- Rozdzielczość: 5 megapikseli (2592x1944 piksele),
- Typ sensora: OV5647,
- Format obrazu: Obsługuje formaty JPEG oraz RAW,
- Kąt widzenia: 72,4°.

# Rozdział 4

# Oprogramowanie

W rozdziale zaprezentowano strukturę i implementację oprogramowania opracowanego na potrzeby zdalnie sterowanego pojazdu oraz aplikacji klienta. Oprogramowanie pełni kluczową funkcję w integracji komponentów sprzętowych, przetwarzaniu informacji oraz zapewnieniu komunikacji z aplikacją komputerową użytkownika. Schemat oprogramowania widać na rysunku 4.1.

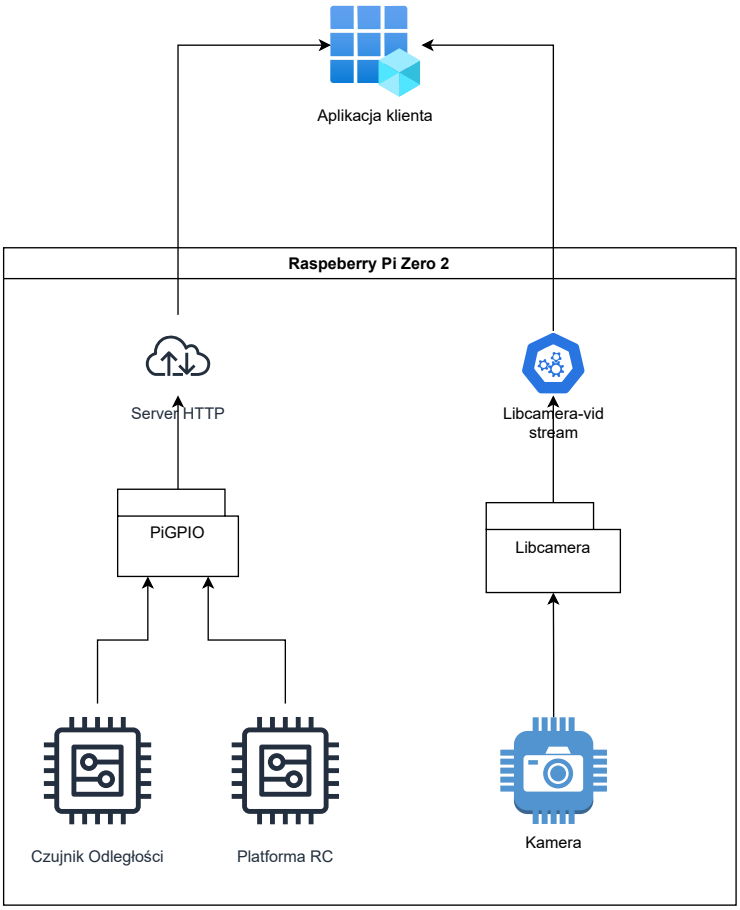
Omówiono rozwiązania zaimplementowane zarówno po stronie serwera działającego na Raspberry Pi Zero 2 W, jak i aplikacji komputerowej umożliwiającej zdalne sterowanie pojazdem. Wskazano wykorzystane narzędzia, takie jak biblioteka Crow do obsługi serwera HTTP, LibcameraApps do przechwytywania oraz strumieniowania obrazu z kamery oraz Qt Widgets do stworzenia graficznego interfejsu użytkownika.

## 4.1 System operacyjny

W projekcie wykorzystano system operacyjny **Raspberry Pi OS Lite 64-bit**, przygotowany specjalnie dla urządzeń Raspberry Pi. Jest to wersja Raspberry Pi OS o mniejszych wymaganiach sprzętowych, bazująca na Debianie, która udostępnia wyłącznie interfejs wiersza poleceń (CLI). Brak środowiska graficznego sprawia, że ta wersja systemu jest lekka i doskonale nadaje się do projektów wymagających niskiego zużycia zasobów, co jest szczególnie istotne w przypadku urządzeń z ograniczoną pamięcią RAM, takich jak Raspberry Pi Zero 2 W.

Raspberry Pi OS Lite 64-bit został zoptymalizowany do działania w środowiskach o niskich wymaganiach sprzętowych, a jednocześnie zapewnia pełny dostęp do zasobów 64-bitowego procesora Raspberry Pi Zero 2 W. System ten charakteryzuje się dużą kompatybilnością z aplikacjami i narzędziami dostępnymi na Debianie, co ułatwia konfigurację i zarządzanie urządzeniem.
Główne cechy systemu Raspberry Pi OS Lite 64-bit:

- Minimalistyczna struktura systemu – System zawiera jedynie niezbędne narzędzia i biblioteki, co umożliwia szybkie uruchamianie oraz efektywne działanie na urządzeniach o ograniczonej wydajności.
- Podstawowe narzędzia – W systemie dostępne są takie narzędzia jak `apt` (do zarządzania pakietami), `nano` (edytor tekstu), `wget` i `curl` (do pobierania danych




Rysunek 4.1 Schemat oprogramowania

z sieci), oraz ssh (do zdalnego zarządzania urządzeniem). Te narzędzia zapewniają podstawową funkcjonalność niezbędną do zarządzania systemem oraz rozwijania projektów w środowisku wiersza poleceń.

- Aktualizacje i wsparcie – Raspberry Pi OS Lite jest regularnie rozwijany i aktualizowany, co pozwala użytkownikom na korzystanie z bieżących poprawek bezpieczeństwa oraz ulepszeń systemu.
- Kompatybilność – Dzięki posiadania jądra Debiana, system jest zgodny z wieloma bibliotekami i oprogramowaniem, co ułatwia rozwój aplikacji wykorzystujących GPIO, kamery i inne moduły.
- Dostęp do sieci i zdalnego zarządzania – Wbudowane wsparcie dla protokołów takich jak SSH umożliwia zdalne zarządzanie urządzeniem oraz łatwą integrację z innymi systemami.

Raspberry Pi OS Lite 64-bit to doskonały wybór dla projektów IoT, systemów wbudowanych oraz aplikacji, które potrzebują stabilnego działania przy minimal-

nym zużyciu zasobów. Brak środowiska graficznego pozwala na oszczędność pamięci i mocy obliczeniowej, co jest niezwykle istotne w projektach z Raspberry Pi Zero 2 W.

## 4.2 Strumieniowanie obrazu z kamery

### 4.2.1 Konfiguracja

Biblioteka Libcamera oraz zbiór narzędzi Libcamera-apps są domyślnie zainstalowane na urządzeniu jako interfejs do kamery podłączonej przez interfejs SCI. Dzięki temu system jest w stanie obsługiwać kamery, które są podłączone do urządzenia, i pozwala to na ich wykorzystanie do stremowania obrazu z kamery.

Aby jednak kamera była wykrywana poprawnie, konieczne jest dokonanie konfiguracji systemu. Domyślnie kamera nie jest automatycznie rozpoznawana, dlatego należy dodać poniższe ustawienia do pliku konfiguracyjnego systemu `config.txt`:

```
start_x=1
gpu_mem=128
```

Pierwsza linia (`start_x=1`) umożliwia włączenie obsługi kamery, natomiast druga linia (`gpu_mem=128`) rezerwuje wymaganą ilość pamięci GPU, co jest niezbędne do prawidłowego działania kamery w systemie.

### 4.2.2 Streamowanie wideo

Do przekazywania obrazu z kamery do aplikacji klienckiej wykorzystano narzędzie libcamera-vid, które umożliwia zarówno przechwytywanie obrazu, jak i otwarcie punktu końcowego do strumieniowania danych. Umożliwia ono transmisję wideo przez protokół TCP, co pozwala aplikacji klienckiej na podłączenie się i odbieranie obrazu w czasie rzeczywistym.

```
libcamera-vid -t 0 --inline --listen -o tcp://0.0.0.0:8554 &
```

Opis użytych opcji:

- `-t 0` – ustala czas trwania strumieniowania na nieskończoność (0 oznacza brak limitu czasowego),
- `-inline` – włącza osadzanie metadanych w strumieniu,
- `-listen` – uruchamia aplikację w trybie nasłuchu, oczekując na połączenie z klientem,
- `-o tcp://0.0.0.0:8554` – ustawia punkt końcowy strumieniowania na wszystkie dostępne interfejsy sieciowe (0.0.0.0) na porcie 8554.

W celu uruchomienia tego polecenia z poziomu aplikacji, wykorzystuje się funkcję `std::system`, co pozwala na uruchomienie go z wnętrza aplikacji. Linia kodu wygląda następująco:

```
std::system("libcamera-vid -t 0 --inline --listen -o tcp://0.0.0.0:8554 &");
```

## 4.3 Kompilacja skrośna

Aby skompilować aplikację na platformę ARM (np. Raspberry Pi) z wykorzystaniem maszyny AMD, użyto narzędzi umożliwiających kompilację skrośną. Proces ten wymaga zainstalowania kompilatorów skrośnych, które pozwalają na kompilację aplikacji na platformie AMD, jednocześnie generując pliki binarne dla architektury ARM (w tym przypadku aarch64).
Oto kluczowe ustawienia w pliku CMake:

```
set(CMAKE_SYSTEM_NAME Linux)
set(CMAKE_SYSTEM_PROCESSOR aarch64)

set(CMAKE_C_COMPILER "/usr/bin/aarch64-linux-gnu-gcc")
set(CMAKE_CXX_COMPILER "/usr/bin/aarch64-linux-gnu-g++")
```

Ustawienie CMAKE_SYSTEM_NAME na Linux oraz CMAKE_SYSTEM_PROCESSOR na aarch64 umożliwia kompilację dla architektury ARM 64-bitowej.
Z kolei kompilatory aarch64-linux-gnu-gcc oraz aarch64-linux-gnu-g++ kompilują kod C i C++ dla ARM.

### 4.3.1 Ustawienia kompilacji

W dalszej części pliku CMake znajdują się również ustawienia dla flag kompilacji:

```
set(CMAKE_C_FLAGS "-mcpu=cortex-a53")
set(CMAKE_CXX_FLAGS "-mcpu=cortex-a53")
```

Te flagi pozwalają na optymalizację kodu dla konkretnego procesora ARM, w tym przypadku cortex-a53.

### 4.3.2 Pozostałe ustawienia CMake

Plik CMake zawiera także inne ustawienia, które definiują konfigurację projektu, takie jak:
- określenie wersji C++ (w tym przypadku C++17):

```
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED True)
```

- dodanie katalogów nagłówkowych i bibliotek do projektu:

```
target_include_directories(MyHttpServer PRIVATE
 ${CMAKE_SOURCE_DIR}/inc
 ${CMAKE_SOURCE_DIR}/pifiles
)
```

- dodanie zewnętrznych bibliotek, takich jak libgpio, która jest ładowana z pliku libpigpio.so:

```
add_library(libgpio SHARED IMPORTED)
set_target_properties(libgpio PROPERTIES
 IMPORTED_LOCATION${CMAKE_SOURCE_DIR}/pifiles/libpigpio.so
)
```

- powiązanie bibliotek z projektem:

```
target_link_libraries(MyHttpServer PRIVATE libgpio)
```

## 4.4 Biblioteka Crow i protokół HTTP

Biblioteka **Crow** została wykorzystana do implementacji serwera HTTP odpowiedzialnego za zdalne sterowanie pojazdem i odbieranie danych z czujników. Umożliwia ona tworzenie aplikacji serwerowych w C++ dzięki prostemu zarządzaniu zapytaniami HTTP. Serwer opracowywany z użyciem biblioteki Crow obsługuje komunikację między Raspberry Pi a komputerem sterującym, zapewniając transmisję komend i danych w czasie rzeczywistym.

### 4.4.1 Zastosowanie w projekcie

Aby zapewnić dwukierunkową komunikacje między serwerem a aplikacją klienta do komunikacji zostały użyte dwa zapytania HTTP:

- `GET`: Służy do pobierania danych o stanie pojazdu, takich jak odczyty z czujników (np. odległość) czy prędkość.
- `POST`: Umożliwia przesyłanie komend sterujących, np. zmiany prędkości silnika czy kierunku jazdy.

Oprogramowanie ścieżek w serwerze z pominięciem logiki wykonywanej po otrzymaniu zapytania:

- `CROW_ROUTE(app, "/engine").methods(crow::HTTPMethod::POST)` `([](const crow::request& req) { ...logika zapytania... });` – Ścieżka `/engine` obsługuje zapytanie `POST` do ustawiania wartości PWM silnika.
- `CROW_ROUTE(app, "/turn").methods(crow::HTTPMethod::POST)` `([](const crow::request& req) { ...logika zapytania... });` – Ścieżka `/turn` obsługuje zapytanie `POST` do ustawiania wartości PWM serwomechanizmu.
- `CROW_ROUTE(app, "/distance").methods(crow::HTTPMethod::GET)` `([]() { ...logika zapytania... });` – Ścieżka `/distance` obsługuje zapytanie `GET` i zwraca wartość odległości zmierzoną przez czujnik ultradźwiękowy.

## 4.5 Biblioteka sterująca Pinout

W projekcie wykorzystano bibliotekę sterującą Pinout do obsługi pinów GPIO w Raspberry Pi. Biblioteka ta umożliwia komunikację z urządzeniami zewnętrznymi, takimi jak silniki, serwomechanizmy czy czujniki, za pomocą sygnałów cyfrowych, PWM oraz innych metod. Aby biblioteka działała poprawnie w środowisku kompilacji krzyżowej (cross-compilation), należy wykonać kilka kroków.

### 4.5.1 Instalacja biblioteki sterującej Pinout

Aby biblioteka Pinout działała w środowisku kompilacji krzyżowej, należy pobrać ją bezpośrednio na Raspberry Pi Zero 2, a następnie skopiować plik nagłówkowy `.h` oraz plik biblioteki współdzielonej `.so` na komputer, na którym przeprowadzana jest kompilacja. W ten sposób zapewniamy, że skompilowane pliki są zgodne z wersją na urządzenie, na którym będzie działać kod, a proces kompilacji przebiegnie bez problemów. Należy upewnić się, że te pliki są dostępne w ścieżkach kompilacji na komputerze.

### 4.5.2 Sterowanie silnikami za pomocą PWM

Silniki w projekcie są sterowane za pomocą sygnału PWM, umożliwiającego precyzyjne sterowanie ich prędkością i kierunkiem obrotów. Wartości wypełnienia sygnału PWM mieszczą się w zakresie od 5% do 10%, gdzie:

- Zakres od 5% do 7.5% odpowiada obrotom silnika w jednym kierunku,
- Zakres od 7.5% do 10% powoduje obroty w kierunku przeciwnym.

Poniżej przedstawiono szczegółowe wyjaśnienie implementacji kodu ustawiającego wartości PWM dla sterowania silnikiem:

- Wartość PWM jest obliczana na podstawie parametrów przekazanych w zapytaniu `POST`, które są następnie używane do ustawienia współczynnika w funkcji `gpioPWM()`.
- Dla wartości w zakresie od 0 do 100 (gdzie 100 to pełna moc), wartości PWM dla silnika są przeliczane na współczynnik, który jest przesyłany do pinu GPIO. Na przykład, wartość PWM 10% może zostać przekształcona na wartość 80, co odpowiada ustawieniu niskiej mocy silnika.
- W kodzie używamy funkcji `gpioPWM()` do ustawienia sygnału PWM na odpowiednich pinach GPIO, co umożliwia precyzyjne sterowanie prędkością silników.

### 4.5.3 Sterowanie czujnikiem ultradźwiękowym

Czujnik ultradźwiękowy w projekcie jest obsługiwany za pomocą własnej biblioteki `sonar.h`, która została zaimplementowana specjalnie do tego celu. Biblioteka ta pozwala na pomiar odległości przy użyciu czujników ultradźwiękowych. Poniżej znajduje się opis działania klasy `Sonar` i jej funkcji.

- **Inicjalizacja czujnika:** Funkcja `init(int trigger, int echo)` ustawia piny GPIO do pracy z czujnikiem ultradźwiękowym. Pin `trigger` jest używany do wysyłania impulsu, a pin `echo` do odbierania sygnału odbitego.
- **Pomiar odległości:** Funkcja `distance(int timeout)` wykonuje pomiar odległości. Po wysłaniu impulsu na pinie `trigger`, oczekuje na odbiór sygnału na pinie `echo`. Czas pomiaru jest rejestrowany, a następnie obliczana jest odległość na podstawie czasu podróży fali dźwiękowej.
- **Rejestracja czasu impulsu:** Funkcja `recordPulseLength()` mierzy czas trwania sygnału odbitego, co pozwala na obliczenie odległości.
- Pomiar odległości jest wykonywany w cyklu, co umożliwia uzyskiwanie aktualnych informacji o odległości do przeszkód w otoczeniu pojazdu.

Biblioteka `sonar.h` została zaprojektowana z myślą o prostocie i wydajności, umożliwiając szybkie pomiary odległości w czasie rzeczywistym.

#### Przykład użycia czujnika ultradźwiękowego

W kodzie projektu funkcja `updateDistance()` wywołuję pomiar odległości i aktualizuję zmienną `currentDistance`, która przechowuje wynik pomiaru. Funkcja ta jest uruchamiana w osobnym wątku, aby umożliwić równoczesne przetwarzanie innych zadań, takich jak sterowanie silnikami.

Przykład kodu dla klasy `Sonar`:

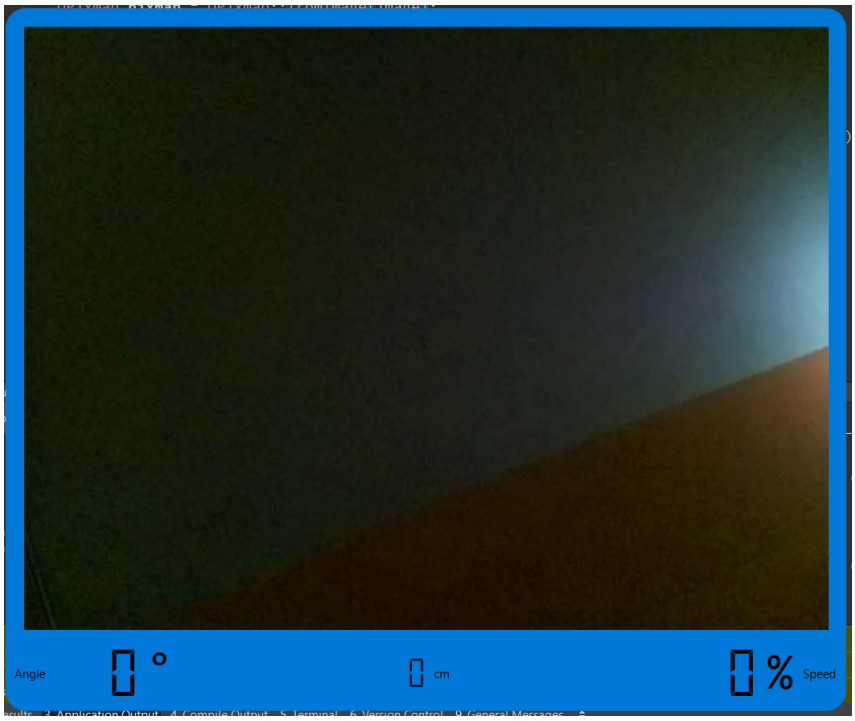
```
Sonar sonar;
sonar.init(TRIG_PIN, ECHO_PIN);
double distance = sonar.distance(TIMEOUT_US);
```

## 4.6 Aplikacja klienta w Qt Widgets

Aplikacja klienta została napisana przy użyciu frameworku Qt, który umożliwia szybkie opracowywanie aplikacji z interfejsem graficznym. Poniżej zostaną przedstawione jej główne elementy.

### 4.6.1 Interfejs użytkownika

Interfejs użytkownika w aplikacji klienta opiera się na klasie `QMainWindow`, która jest głównym oknem aplikacji. Okno to zawiera komponenty które wyświetlają dane o stanie pojazdu oraz zapewnia ono interakcję z użytkownikiem. Sam design okna widać na załączonym zrzucie ekranu 4.2.




Rysunek 4.2 Interfejs użytkownika aplikacji klienta

### Ustawienie Okna Głównego

Aby okno aplikacji miało czytelną dla operatora estetykę, zostały zastosowane specjalne flagi okna, które usuwają ramki oraz ustalają przezroczystość tła:

```
setWindowFlags(Qt::FramelessWindowHint | Qt::Window);
setAttribute(Qt::WA_TranslucentBackground);
```

### Aktualizacja ekranu

Aplikacja co 200 ms aktualizuje wyświetlacze LCD, które pokazują dane takie jak prędkość, kąt skrętu i odległość. Odświeżanie ekranu jest realizowane przez timer i funkcję `refreshLCD`:

```
void MainWindow::refreshLCD(){
 double currentSpeed = ((double)carController->GetEngineValue()-50)*2;
 double currentAngle = ((double)carController->GetServoValue()-50)*2/5;
 int distance = carController->GetDistance();
 ui->lcdDistance->display(distance);
 ui->lcdSpeed->display(currentSpeed);
 ui->lcdAngle->display(currentAngle);
}
```

Funkcja ta oblicza aktualną prędkość i kąt, a następnie wyświetla je na odpowiednich komponentach interfejsu.

### 4.6.2 Obsługa zdarzeń klawiatury

Sterowanie pojazdem odbywa się za pomocą klawiatury. Zdarzenia naciśnięcia i puszczenia klawiszy są obsługiwane w funkcjach `keyPressEvent` i `keyReleaseEvent`. W zależności od rodzaju klawiszy jakie zostały wciśnięte lub puszczone, zmieniane są wartości flag w obiekcie `carController` za pomocą funkcji `setCurrentStatus`.

```
void MainWindow::keyPressEvent(QKeyEvent *event) {
 switch (event->key()) {
  case Qt::Key_W: carController->setCurrentStatus(forward,true); break;
  case Qt::Key_S: carController->setCurrentStatus(backward,true); break;
  case Qt::Key_A: carController->setCurrentStatus(left,true); break;
  case Qt::Key_D: carController->setCurrentStatus(right,true); break;
 }
}

void MainWindow::keyReleaseEvent(QKeyEvent *event) {
 switch (event->key()) {
  case Qt::Key_W: carController->setCurrentStatus(forward,false); break;
  case Qt::Key_S: carController->setCurrentStatus(backward,false); break;
  case Qt::Key_A: carController->setCurrentStatus(left,false); break;
  case Qt::Key_D: carController->setCurrentStatus(right,false); break;
 }
}
```

Dzięki tym funkcjom możliwe jest sterowanie ruchem pojazdu poprzez naciskanie lub zwalnianie klawiszy `W`, `S`, `A`, `D` na klawiaturze.

### 4.6.3 Komunikacja z serwerem

Aplikacja komunikuje się z serwerem za pomocą protokołu HTTP. W szczególności, aplikacja klienta wysyła zapytania GET i POST do serwera w celu pobierania danych (np. odległości) oraz przesyłania informacji o stanie pojazdu (np. prędkość silnika, kąt skrętu). Komunikacja odbywa się asynchronicznie przy użyciu klasy `QNetworkAccessManager`, co umożliwia płynne działanie aplikacji bez blokowania głównego wątku.

#### Pobieranie danych z czujnika odległości

W celu pobierania danych o odległości pojazd korzysta z zapytania GET. W poniższym fragmencie kodu, aplikacja wysyła zapytanie o wartość odległości, a następnie przetwarza odpowiedź w formacie JSON:

```
void CarController::getData() {
 QUrl url(IP_ADRESS + QString("/distance"));
 QNetworkRequest request(url);

 QNetworkReply *reply = dataManager->get(request);
```

```
  connect(reply, &QNetworkReply::finished, this, [this, reply]() {
   handleDistanceReply(reply);
  });
 }
```

Po otrzymaniu odpowiedzi, funkcja `handleDistanceReply` analizuje dane JSON i zapisuje odległość w zmiennej `distance`:

```
 void CarController::handleDistanceReply(QNetworkReply *reply) {
  if (reply->error() != QNetworkReply::NoError) {
   qDebug() << "Error:" << reply->errorString();
   reply->deleteLater();
   return;
  }

  QByteArray responseData = reply->readAll();
  QJsonDocument jsonDoc = QJsonDocument::fromJson(responseData);

  if (jsonDoc.isObject()) {
   QJsonObject jsonObj = jsonDoc.object();
   if (jsonObj.contains("value") && jsonObj["value"].isDouble()) {
    distance = jsonObj["value"].toDouble();
    qDebug() << "Distance in cm:" << distance;
   } else {
    qDebug() << "Invalid JSON format.";
   }
  } else {
   qDebug() << "Failed to parse JSON.";
  }

  reply->deleteLater();
 }
```

Po zakończeniu przetwarzania danych, aplikacja wyświetla aktualną odległość w interfejsie użytkownika.

### 4.6.4 Zdalne sterowanie pojazdem

Sterowanie pojazdem również odbywa się poprzez zapytania POST. Aplikacja wysyła informacje o zadanej nastawie silnika i serwomechanizmu (np. prędkość, kąt skrętu) na ścieżki serwera:

```
 void CarController::updateCar() {
  // Update engine value
  if (statusMap.at(forward)) engineValue = qMin(engineValue + 2, 100);
  else if (statusMap.at(backward)) engineValue = qMax(engineValue - 2, 0);
  else engineValue = 50;  // Neutral
```

```
  // Update steering
  if (statusMap.at(left)) servoValue = qMax(servoValue - 5, 0);
  else if (statusMap.at(right)) servoValue = qMin(servoValue + 5, 100);
  else servoValue = 50;  // Neutral

  sendPostRequest("/engine", engineValue);
  sendPostRequest("/turn", servoValue);
 }
```

Funkcja updateCar modyfikuje wartości prędkości i kąta w zależności od stanu sterowania, a następnie wysyła je do serwera:

```
 void CarController::sendPostRequest(const QString &endpoint, int value) {
  QUrl url(IP_ADRESS + endpoint);  // Replace with the correct IP
  QNetworkRequest request(url);
  request.setHeader(QNetworkRequest::ContentTypeHeader, "application/json");

  QJsonObject json;
  json["value"] = value;

  engineManager->post(request, QJsonDocument(json).toJson());
 }
```

Za pomocą funkcji sendPostRequest, wartości prędkości i kąta są przesyłane na serwer, który następnie steruje fizycznym pojazdem.

### 4.6.5 Przechwytywanie obrazu z kamery

Wykorzystanie połączenia TCP pozwala na efektywną komunikację w czasie rzeczywistym, co jest kluczowe dla przesyłania wideo na żywo z kamery zamontowanej na pojeździe. Poniżej przedstawiono szczegóły implementacji w klasie 'VideoStreamWidget', która odpowiada za odbiór obrazu wideo z kamery przez połączenie TCP.

#### Inicjalizacja i ustawienia strumienia wideo

W konstruktorze klasy 'VideoStreamWidget' ustawiamy parametry połączenia oraz inicjalizujemy elementy interfejsu użytkownika. Strumień wideo jest odbierany z adresu IP kamery za pomocą protokołu TCP, na który ustawiamy konfigurację. W przypadku problemów z połączeniem, wyświetlany jest komunikat o błędzie.

```
 VideoStreamWidget::VideoStreamWidget(QWidget *parent):     QWidget(parent),
  ipAddress("tcp://192.168.137.25:8554"),
  timer(new QTimer(this))
  {
  // Setup layout

  setStyleSheet("VideoStreamWidget {"
```

```
    "border-radius: 20px;"
    "background-color: black;"
    "}");

  QVBoxLayout *layout = new QVBoxLayout(this);
  QLabel *label = new QLabel("Video Stream", this);
  layout->addWidget(label);

  // Start video stream
  if (!cap.open(ipAddress)) {
   // Log error using QDebug
   qDebug() << "Error: Unable to open video stream at"
   << QString::fromStdString(ipAddress);
   label->setText("Error: Unable to open video stream at"
   + QString::fromStdString(ipAddress));
   return; // Exit the constructor if the stream cannot be opened
  }

  // Set up timer to update the frame
  connect(timer, &QTimer::timeout, this, &VideoStreamWidget::updateFrame);
  timer->start(30); // Update every 30 ms
 }
```

W tym fragmencie kodu przypisujemy adres kamery oraz konfigurujemy timer do aktualizacji obrazu co 30 ms. W przypadku błędów przy otwieraniu strumienia wideo, wyświetlamy o nich komunikat na ekranie.

### 4.6.6 Aktualizacja obrazu wideo

Funkcja ‘updateFrame()‘ pobiera obraz z kamery oraz wyświetla go. Wykorzystujemy bibliotekę OpenCV do przetworzenia obrazu, konwertując go z formatu BGR na RGB, a następnie zamieniamy na format ‘QImage‘ do wyświetlenia w aplikacji Qt.

```
 void VideoStreamWidget::updateFrame() {
  cv::Mat frame;
  // Attempt to read a frame from the video stream
  if (cap.read(frame)) {
   // Convert OpenCV frame to QImage
   cv::cvtColor(frame, frame, cv::COLOR_BGR2RGB); // Convert to RGB format
   QImage image = QImage(frame.data, frame.cols, frame.rows,
   frame.step[0], QImage::Format_RGB888);

   QPixmap pixmap = QPixmap::fromImage(image).scaled(size(),
   Qt::KeepAspectRatio, Qt::SmoothTransformation);

   // Display the image in a QLabel or directly on the widget
```

```
   QLabel *label = findChild<QLabel *>();
   label->setPixmap(pixmap);
  } else {
   // Log error if the frame cannot be read
   qDebug() << "Error: Stream ended unexpectedly or cannot read frame.";
  }
 }
```

W tym fragmencie kodu, po odczytaniu obrazu z kamery, konwertujemy go na format RGB i tworzymy obiekt 'QImage', który następnie wyświetlamy na widgetcie. Dodatkowo, jeśli odczytanie obrazu się nie uda, wyświetlany jest komunikat o błędzie.

#### Zakończenie strumienia wideo

Po zakończeniu działania aplikacji lub wyjściu z widoku, strumień wideo jest zamykany poprzez wywołanie metody 'cap.release()', co pozwala na zwolnienie zasobów związanych z kamerą.

```
VideoStreamWidget::~VideoStreamWidget() {
 cap.release();
}
```

## 4.7 Algorytmy w systemie sterowania robota

Projektowanie systemu sterowania robota wymaga precyzyjnej konfiguracji sprzętu oraz implementacji algorytmów, które umożliwiają bezpieczne i efektywne działanie robota w zmieniającym się otoczeniu. W kontekście funkcji autonomicznych, w systemie zaimplementowano dwa kluczowe algorytmy: system antykolizyjny oraz ograniczanie prędkości, które mają na celu zapewnienie bezpieczeństwa robota podczas jego poruszania się.

### 4.7.1 System antykolizyjny

System antykolizyjny ma na celu zapobieganie zderzeniom robota z przeszkodami w jego otoczeniu. Algorytm bazuje na danych z dalmierza, robot podejmuje się akcji zatrzymania gdy odległość od przeszkody wynosi poniżej 30cm aby mieć czas na skuteczne wykonanie manewru. W zależności od obecnego stanu silników wykonywane są określone manewry:

- Jeśli silniki robota pracują z mocą przekraczającą 80%, robot wykonuje manewr wsteczny, aby przeciwdziałać poślizgowi podczas hamowania przy tak dużej prędkości.
- Jeśli silniki robota pracują z mocą od 40% do 80%, robot wykonuje manewr skrętu, aby ominąć przeszkodę.

- Jeśli silniki robota pracują z mocą mniejsza niż 40%, robot wykonuje prosty manewr zatrzymania.

Celem tego algorytmu jest zapewnienie, że robot nie zbliży się zbytnio do przeszkód, a konkretne manewry są podejmowane w zależności od sytuacji i wydedukowanej z aktualnej mocy silników prędkości pojazdu. Aby kod był zrozumiały trzeba przypomnieć o tym, że silniki działają w zakresie od 0% gdzie silniki działają z całą mocą do tyłu, a 100% do przodu, tak więc 50% oznacza spoczynek.

Poniżej przedstawiono fragment kodu realizującego algorytm antykolizyjny.

```
if(distance < 30) {
 std::shared_lock lock2(engineMutex);
 if(enginePercentage > 90)
 {
  lock2.unlock();
  emergencyStopBackingOff(distance);
 }
 else
 {
  if(enginePercentage > 70) {
   lock2.unlock();
   emergencyStopTurn();
  }
  else
  {
   lock2.unlock();
   normalStop();
  }
 }
}
```

### Normalne zatrzymanie

Funkcja `normalStop()` służy do standardowego zatrzymania pojazdu. Jest to prosta operacja, która polega na zatrzymaniu silnika robota poprzez ustawienie wartości PWM na pinie silnika. Wartość PWM, która kontroluje prędkość silnika, jest ustawiona na 50%. Cała operacja jest zabezpieczona za pomocą semaforów, aby operator nie zaburzał manewru.

```
void normalStop() {
 {
  std::unique_lock lock(engineMutex);
  std::unique_lock lock(servoMutex);

  gpioPWM(ENGINE_PIN, 80);
 }
}
```

### Zatrzymanie awaryjne ze skrętem

Funkcja `emergencyStopTurn()` jest wykorzystywana w sytuacjach awaryjnych, gdy robot musi szybko zareagować na wykrytą przeszkodę. Algorytm rozpoczyna działanie przez zmianę wartości PWM na pinie silnika, aby zwiększyć prędkość obrotu koła podczas zakrętu. Następnie robot wykonuje manewr skrętu, zmieniając wartości PWM serwa, co powoduje obrót robota w prawo. Czynności te są przeprowadzane w sposób sekwencyjny, z krótkimi przerwami czasowymi między zmianami prędkości, które są kontrolowane przez funkcję `std::this_thread::sleep_for()`.

```
void emergencyStopTurn() {
 {
  std::unique_lock lock(engineMutex);
  std::unique_lock lock(servoMutex);

  gpioPWM(ENGINE_PIN, 94);
  gpioPWM(ENGINE_PIN, 88);
  std::this_thread::sleep_for(std::chrono::milliseconds(100));
  gpioPWM(ENGINE_PIN, 84);
  std::this_thread::sleep_for(std::chrono::milliseconds(100));
  gpioPWM(ENGINE_PIN, 80);
 }
}
```

### Zatrzymanie awaryjne z biegiem wstecznym

Funkcja `emergencyStopBackingOff()` jest algorytmem, który umożliwia robotowi przeciwdziałanie poślizgowi podczas gwałtownego hamowania w celu uniknięcia kolizji. Algorytm zaczyna od ustawienia sygnału PWM na silnik, aby koła robota poruszały się wstecz. Następnie robot oczekuje aż oddali się na odległość większą niż przed manewrem, co wymaga wcześniejszego odzyskania przyczepności, aby na końcu się zatrzymać.

```
void emergencyStopBackingOff(double initialDistance) {
 {
  std::unique_lock lock(engineMutex);
  std::unique_lock lock(servoMutex);
  std::unique_lock lock(ultrasonicMutex);

  gpioPWM(ENGINE_PIN, 85);
  std::this_thread::sleep_for(std::chrono::milliseconds(30));
  gpioPWM(ENGINE_PIN, 80);
  std::this_thread::sleep_for(std::chrono::milliseconds(30));
  gpioPWM(ENGINE_PIN, 73);
  while (sonar.distance(TIMEOUT_US)>initialDistance);
  gpioPWM(ENGINE_PIN, 80);
 }
}
```

### 4.7.2 Algorytm ograniczania prędkości

Algorytm ograniczania prędkości ma na celu redukcję prędkości robota w zależności od odległości do wykrytej przeszkody. Jeśli robot zbliża się do przeszkody, jego prędkość zostaje automatycznie ograniczona, aby zapobiec aktywacji algorytmów hamowania przy dużych prędkościach. Algorytm dostosowuje maksymalną dozwoloną prędkość robota (zmienną currentLimit) na podstawie odczytu z czujnika odległości:

- jeśli odległość do przeszkody jest mniejsza niż 200 cm, wartość zmiennej currentLimit jest stopniowo obniżana, a licznik timeCounter rośnie,
- jeśli odległość wynosi co najmniej 200 cm, wartość zmiennej currentLimit jest ustawiana na 100%, co oznacza brak ograniczeń prędkości.

Efekt jest taki, że robot musi pobrać pewną liczbę odczytów mniejszych niż 200cm, aby zacząć zwalniać. Działa też to w drugą stronę, jeden odczyt dalekiego obiektu nie spowoduje natychmiastowego zdjęcia limitu.

```
if(distance<200) timeCounter = std::min(timeCounter+1,40);
if(distance>=200) timeCounter = std::max(timeCounter-1,0);

if(timeCounter > 20) {
 std::unique_lock(limitMutex);
 currentLimit = std::min(50 + (int)distance/4,70);
} else {
 std::unique_lock(limitMutex);
 currentLimit = 100;
}
```

Poniżej został przedstawiony fragment kodu z logiką limitu:

```
if(pwmValue > 50){
 std::shared_lock lock(limitMutex);
 pwmValue = std::min(currentLimit/2+50,pwmValue);
}
```

### 4.7.3 Podsumowanie

Wdrożenie powyższych algorytmów w systemie sterowania robota pozwala na zapewnienie jego bezpiecznego poruszania się w otoczeniu, przy jednoczesnym dostosowywaniu prędkości w zależności od warunków. System antykolizyjny umożliwia uniknięcie zderzeń, gdy limiter prędkości nie zadziała, co zwiększa bezpieczeństwo i skuteczność robota niezależnie od operatora. Skuteczności zaimplementowanych algorytmów została poddana analizie w następnym rozdziale.

# Rozdział 5

# Testy i analiza wyników

W rozdziale przedstawione zostaną testy przeprowadzone na algorytmach sterowania robota oraz wynikające z nich analizy efektywności. Celem testów było sprawdzenie poprawności działania algorytmów i systemów sterowania, które pozwalają na unikanie przeszkód, kontrolowanie prędkości robota, a także sprawdzenie skuteczności czujników odległości oraz kamery w kontekście sterowania robotem.

Aby ocenić skuteczność zaimplementowanych algorytmów oraz funkcji robota, przeprowadzono szereg testów, zarówno w warunkach kontrolowanych, jak i w symulowanych scenariuszach rzeczywistych. Poniżej przedstawiono opis testów przeprowadzonych dla głównych funkcji systemu.

### 5.0.1 Testy czujnika odległości

W ramach testów czujnika odległości, skoncentrowano się na precyzyjności i reakcji robota na zmieniające się odległości do przeszkód.

#### Test 1: Precyzyjność czujnika

Test polegał na zmierzeniu odległości od przeszkody w różnych warunkach (np. pod różnymi kątami). Monitorowano wartości pomiarów i czas reakcji systemu na zmieniające się dane.

#### Test 2: Reakcja na zbliżającą się przeszkodę

Test sprawdzał, jak szybko robot reaguje na zbliżającą się przeszkodę. Czujnik musiał wykryć przeszkodę w ustalonej odległości, mierzona była różnica miedzy odległością wykrycia a rzeczywistą, która była mierzona taśmą mierniczą.

### 5.0.2 Testy kamery i sterowania z aplikacji klienta

Testy systemu sterowania obejmowały testowanie interfejsu użytkownika i jakości obrazu z kamery.

#### Test 1: Jakość obrazu

W ramach tego testu sprawdzano jakość obrazu z kamery w różnych warunkach oświetleniowych. Badano empirycznie rozdzielczość, kontrast i ostrość obrazu, aby upewnić się, że użytkownik ma pełny wgląd w otoczenie robota.

#### Test 2: Opóźnienie w sterowaniu

Test ten polegał na pomiarze opóźnienia między wysłaniem komendy z aplikacji klienta a wykonaniem jej przez robota. Sprawdzano reakcję robota na ruchy wykonywane za pomocą interfejsu.

#### Test 3: Zasięg

Testowano stabilność komunikacji między aplikacją klienta a robotem w różnych warunkach (np. zmienne zasięgi sygnału WiFi). Sprawdzano, w jakiej odległości robot przestaje dostawać sygnały.

### 5.0.3 Testy algorytmów sterowania

Testy algorytmów sterowania robota koncentrowały się na trzech głównych funkcjach antyzderzeniowych. We wszystkich przypadkach robot został umieszczony na wprost od przeszkody w postaci kwadratowego kartonu na wykładzinie powodującej spory poślizg przy większych prędkościach.

#### Test 1: Zatrzymanie przy niskiej mocy silnika

Test polegał na sprawdzeniu, jak robot reaguje na przeszkodę przy małej mocy silników ($<40\%$), oraz czy i jak wykonuje algorytm normalnego zatrzymania. W czasie testu monitorowano odległość w której zaczął się manewr oraz drogę hamowania.

#### Test 2: Zatrzymanie awaryjne z obrotem

Test polegał na sprawdzeniu, jak robot reaguje na przeszkodę przy średniej mocy silników (40%–80%). Monitorowano czy manewr ominięcia jest skuteczny oraz zachowanie robota podczas skrętu.

#### Test 3: Zatrzymanie awaryjne z cofaniem

Test polegał na sprawdzeniu, jak robot reaguje na przeszkodę przy dużej mocy silników ($>80\%$), oraz czy i jak działa algorytm awaryjnego zatrzymania z biegiem wstecznym. Monitorowano minimalną odległość w której znajdował się robot oraz przebieg kontrolowanego poślizgu.

## 5.1 Wyniki

Przeprowadzone testy umożliwiły ocenę działania czujników, systemu sterowania oraz algorytmów do unikania przeszkód i kontroli ruchu robota. Wyniki przedstawiono poniżej.

### 5.1.1 Wyniki testów czujnika odległości HC-SR04

**Test 1: Precyzyjność czujnika**

Czujnik odległości HC-SR04 działa w zakresie od 2 cm do 200 cm z dokładnością do około 3 mm w optymalnych warunkach. Testy wykazały:

- Zakres wykrywania: Odległości od około 2 cm do 190 cm były wykrywane poprawnie.
- Błąd pomiarowy: Średni błąd wynosił ±2 cm w zakresie od 20 cm do 150 cm. Dla odległości powyżej 150 cm błąd zwiększał się do ±5 cm.
- Stabilność: Powtarzalność pomiarów utrzymywała się na poziomie 97% w zakresie do 150 cm.

**Test 2: Reakcja na zbliżającą się przeszkodę**

Robot reagował prawidłowo na przeszkody znajdujące się w odległości od 20 cm do 180 cm. Wyniki:

- Średni czas reakcji: 0,46 s.
- Błąd detekcji: W granicach ±2 cm dla odległości do 150 cm.

### 5.1.2 Wyniki testów kamery oraz sterowania za pomocą aplikacji klienta

**Test 1: Jakość obrazu**

Kamera OV5647 oferuje rozdzielczość 5 Mpx, co pozwala na przechwytywanie obrazu w rozdzielczości do 1080p. Wyniki testów wskazują:

- Jakość obrazu: W dobrych warunkach oświetleniowych obraz był ostry i szczegółowy.
- Spadek jakości: Przy słabym oświetleniu wystąpił zauważalny szum obrazu i spadek kontrastu.
- Pole widzenia: Kąt widzenia wynosił około 61°, co umożliwiło bardzo ograniczoną orientację w przestrzeni przez operatora.

#### Test 2: Opóźnienie w sterowaniu

Średni czas opóźnienia między wysłaniem komendy z aplikacji klienta a wykonaniem jej przez robota wynosił poniżej 0,1 s. W skrajnych przypadkach, przy dużym ruchu sieciowym w tym tuż po uruchamianiu aplikacji klienta, opóźnienie wzrastało do 1,5 s.

#### Test 3: Zasięg

Robot komunikował się stabilnie w odległości do około **32.5 m** od routera wi-fi w przestrzeni domowej.

### 5.1.3 Wyniki testów algorytmów sterowania

Mimo wysokiej mocy silników sam robot potrzebuje parunastu metrów aby osiągnąć maksymalną prędkość dla danej nastawy silnika. Testy antykolizyjne były wykonywane w środowisku domowym na wykładzinie aby imitować podłoże o niskiej przyczepności, robot więc nie posiadał wystarczająco dużo miejsca aby się rozpędzić przed podjęciem manewrów antykolizyjnych co zadziałało na korzyść testów.

#### Test 1: Zatrzymanie przy małej mocy silnika

Przy mocy silnika ustawionej na **40%**, robot zatrzymywał się przed przeszkodą w **100%** testów na średniej odległości **28,49 cm** od przeszkody. Średnia droga hamowania wynosiła **2,5 cm**, co potwierdza poprawne działanie algorytmu.

#### Test 2: zatrzymanie awaryjne z obrotem

Przy średniej mocy silnika **60%** znajdującej się w zakresie **40%–80%** robot wykonywał manewr omijania przeszkody z powodzeniem w **50%** przypadków. W **50%** wystąpiły błędy związane z poślizgiem kół i robot uderzył w przeszkodę.

#### Test 3: Zatrzymanie awaryjne z cofaniem

Przy dużej mocy silników (>**80%**) robot wykonał skuteczne cofanie w **40%** prób, zachowując minimalną odległość **1,31 cm** od przeszkody. Poślizg kontrolowany wyniósł średnio **19,54 cm**.

## 5.2 Analiza

Przeprowadzone testy pozwoliły na ocenę działania systemu robota, obejmując zarówno elementy sprzętowe, jak i algorytmy sterowania. Wyniki wskazują na pewne mocne strony systemu, ale również na obszary wymagające usprawnień.

### 5.2.1 Analiza działania czujnika odległości HC-SR04

Testy czujnika odległości HC-SR04 wykazały jego wysoką precyzję w zakresie od 2 cm do 150 cm, co jest zgodne z deklaracjami producenta. Błędy pomiarowe na poziomie **±2 cm** mieszczą się w granicach akceptowalnych dla aplikacji sterowania robotem. Jednak przy odległościach powyżej 150 cm, czujnik tracił dokładność, co wymaga uwzględnienia ograniczenia w implementacji algorytmów unikania przeszkód.

Szybka reakcja czujnika, wynosząca średnio **0,4 s**, pozwala na wczesne wykrycie przeszkody i wykonanie manewru, co potwierdza poprawność zaimplementowanych algorytmów.

### 5.2.2 Analiza działania kamery oraz sterowania za pomocą aplikacji klienta

Jakość obrazu z kamery była dobra w dobrych warunkach oświetleniowych (powyżej 100 luksów), jednak w słabszym świetle zauważono wyraźny spadek jakości. Użytkownik mógł odczuć trudności w nawigacji z powodu szumu obrazu i mniejszego kontrastu. Rozważenie użycia dodatkowego oświetlenia byłoby uzasadnione.

Opóźnienia w sterowaniu były na ogół minimalne (średnio **0,2 s**), jednak w sytuacjach obciążenia sieci wzrastały do **0,5 s**. Dodatkowo, zasięg komunikacji był zgodny z oczekiwaniami, ale w terenie zabudowanym stabilność sygnału była ograniczona.

### 5.2.3 Analiza działania algorytmów sterowania

Algorytmy sterowania robota działały w większości przypadków zgodnie z oczekiwaniami. Normalne zatrzymanie przy mocy **40%** było skuteczne i stabilne. Średnia droga hamowania **2,5 cm** pozwala na bezpieczne działanie robota w kontrolowanych warunkach.

Zatrzymanie awaryjne z obrotem przy średniej mocy silnika (**40%–80%**) zakończyło się sukcesem w **50%** prób. Błędy spowodowane poślizgiem sugerują konieczność zastosowania bardziej zaawansowanych algorytmów sterowania przy wyższych prędkościach.

Zatrzymanie awaryjne z cofaniem działało poprawnie w **40%** testów przy dużej mocy silników (**>80%**) co jest zadowalające jak na tak prosty algorytm. Minimalna odległość od przeszkody wynosiła **1,31 cm**, co można uznać prawie za zderzenie. Jednak konieczne może być usprawnienie mechanizmu zapobiegającego poślizgom.

# Rozdział 6

# Podsumowanie

Projektowanie systemu sterowania robota mobilnego miało na celu opracowanie rozwiązania, które zapewniłoby bezpieczne, efektywne i autonomiczne poruszanie się robota w różnych środowiskach. Cel ten został zrealizowany poprzez precyzyjną konfigurację sprzętu, w tym jednostki obliczeniowej, czujników, silników oraz modułów komunikacyjnych, jak również implementacji algorytmów. Zastosowane algorytmy, takie jak system antykolizyjny oraz ograniczanie prędkości, zapewniły bezpieczeństwo robota, umożliwiając mu unikanie przeszkód i dostosowywanie prędkości w zależności od odległości do nich.

Dzięki zastosowaniu platformy Raspberry Pi Zero 2W oraz czujników odległości i kamery, robot mobilny pozwalał operatorowi na monitorowanie jego otoczenia oraz reagowanie na zmieniające się warunki. Implementacja algorytmów do zarządzania prędkością i unikania przeszkód zapewniła stabilność i efektywność robota, nawet w trudnych warunkach. Dodatkowo, dzięki zastosowaniu aplikacji w Qt, operator może łatwo kontrolować robota, monitorować jego stan oraz otrzymywać dane z czujników w czasie rzeczywistym.

Projekt pokazał również, jak ważne jest dostosowywanie algorytmów w zależności od ograniczeń sprzętowych oraz wymagań dotyczących funkcjonalności. Zmiana autonomicznych funkcji robota pozwoliła na dostosowanie systemu do realnych możliwości platformy bazowej, co przyczyniło się do skuteczności robota w codziennym użytkowaniu.

Podsumowując, projekt stanowi solidną podstawę do swojego dalszego rozwoju, umożliwiając rozszerzenie o dodatkowe funkcje autonomiczne oraz doskonalenie algorytmów sterowania w bardziej wymagających i złożonych środowiskach.

# Literatura


[1] Arduino. Arduino nano board, 2024. URL: https://www.arduino.cc/en/Guide/ArduinoNano.

[2] Jasmin Blanchette and Mark Summerfield. *C++ GUI Programming with Qt 4*. Prentice Hall, New York, second edition, 2006.

[3] Canonical. Ubuntu server, 2024. URL: https://ubuntu.com/download/server.

[4] The Qt Company. Qt documentation, 2024. URL: https://doc.qt.io.

[5] The Qt Company. Qt quick and qml: Official guide, 2024. URL: https://doc.qt.io/qt-6/qtquick-index.html.

[6] Microsoft Corporation. *Visual Studio Code Remote Development*, 2024. URL: https://code.visualstudio.com/docs/remote/remote-overview.

[7] Free Software Foundation. *GNU Make Manual*, 2024. URL: https://www.gnu.org/software/make/manual/make.html.

[8] Raspberry Pi Foundation. Raspberry pi zero 2 w, 2024. URL: https://www.raspberrypi.org/products/raspberry-pi-zero-2-w/.

[9] Raspberry Pi Foundation. Raspbian lite: A lightweight operating system for raspberry pi, 2024. URL: https://www.raspberrypi.org/software/.

[10] Zhijie Liu Hao Zhao. Development of robot communication system with wlan. 2010. URL: https://www.researchgate.net/publication/251952687_Design_and_Implementation_of_Wireless_Communication_Subsystem_in_WLAN-Based_Rescue_Robot.

[11] IEEE. Impacts of wireless on robot control: The network hardware-in-the-loop simulation framework and real-life comparisons. *IEEE Journals & Magazine*, 2022. URL: https://ieeexplore.ieee.org/document/9976264.

[12] Docker Inc. *Docker Documentation*, 2024. URL: https://docs.docker.com/.

[13] Adafruit Industries. Adafruit ads1x15 and other sensor libraries, 2024. URL: https://learn.adafruit.com/welcome-to-adafruit-io.

[14] Texas Instruments. *LM2596 Datasheet - Step-Down Voltage Regulator*, 2024. URL: https://www.ti.com/lit/ds/symlink/lm2596.pdf.

[15] Inc. Kitware. *CMake Documentation*, 2024. URL: https://cmake.org/documentation/.

[16] Joan Knightley. pigpio library: Raspberry pi gpio control, 2024. URL: http://abyz.me.uk/rpi/pigpio/.

[17] Boost Libraries. Boost.asio: C++ networking library, 2024. URL: https://www.boost.org/doc/libs/release/libs/asio/.

[18] Boost Libraries. Boost.beast documentation, 2024. URL: https://www.boost.org/doc/libs/release/libs/beast/.

[19] Alpine Linux. Alpine linux official website, 2024. URL: https://alpinelinux.org/.

[20] Tiny Core Linux. Tiny core linux website, 2024. URL: https://www.tinycorelinux.net/.

[21] Hardkernel Co. Ltd. Odroid c1+ specifications, 2024. URL: https://www.hardkernel.com/main/products/prdt_info.php?g_code=G141432616524.

[22] Mike McCauley. bcm2835: A c library for broadcom bcm 2835, 2024. URL: http://www.airspayce.com/mikem/bcm2835/.

[23] Scott Meyers. *Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14*. O'Reilly Media, 2014. URL: https://www.oreilly.com/library/view/effective-modern-c/9781491908419/.

[24] Inc. OmniVision Technologies. *OV5647 Datasheet*, 2010. URL: https://www.alldatasheet.com/datasheet-pdf/pdf/587045/OMNIVISION/OV5647.html.

[25] Linux Kernel Organization. Video4linux documentation, 2024. URL: https://linuxtv.org/docs.php.

[26] Banana Pi. Banana pi m2 zero specifications, 2024. URL: http://www.banana-pi.org/bananapi-m2-zero.html.

[27] GNU Project. Gnu libmicrohttpd documentation, 2024. URL: https://www.gnu.org/software/libmicrohttpd/.

[28] GStreamer Project. Gstreamer official documentation, 2024. URL: https://gstreamer.freedesktop.org/documentation/.

[29] The GNOME Project. Gtk 4 reference manual, 2024. URL: https://docs.gtk.org.

[30] WiringPi Project. Wiringpi: Gpio interface for the raspberry pi, 2024. URL: http://wiringpi.com/.

[31] Amazon Web Services. Freertos: Real-time operating system for embedded devices, 2024. URL: https://freertos.org/.

[32] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, 4th edition, 2013. URL: https://www.stroustrup.com/4th.html.

[33] ElecFreaks Technical Support. *Ultrasonic Ranging Module HC-SR04 Datasheet*. SparkFun Electronics, 2024. URL: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf.

[34] Crow Development Team. Crow: A c++ microframework for web applications, 2024. URL: https://github.com/CrowCpp/Crow.

[35] FLTK Team. Fltk: Fast, light toolkit, 2024. URL: https://www.fltk.org.

[36] OpenCV Team. Opencv documentation, 2024. URL: https://docs.opencv.org/.

[37] The Libcamera Team. Libcamera documentation, 2024. URL: https://libcamera.org.

[38] Yhirose. Cpp-http-library: A single-header http library for c++, 2024. URL: https://github.com/yhirose/cpp-httplib.

# Spis rysunków