

Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH



Instytut Automatyki i Informatyki Stosowanej

Praca dyplomowa inżynierska

na kierunku Automatyka i Robotyka

Projekt i implementacja algorytmu detekcji pasa ruchu
dla autonomicznych pojazdów

Wojciech Pobocha

Numer albumu 318399

promotor
dr inż. Krystian Radlak

WARSZAWA 2025

Projekt i implementacja algorytmu detekcji pasa ruchu dla autonomicznych pojazdów

Streszczenie. W ostatnich latach branża motoryzacyjna przechodzi dynamiczną transformację, napędzaną rozwojem systemów wspomagających kierowcę (ADAS - Advanced Driver Assistance Systems) oraz dążeniem do pełnej autonomizacji pojazdów. Kluczową rolę w tym procesie odgrywa postęp w dziedzinie sztucznej inteligencji, w szczególności wizji komputerowej, która umożliwia tworzenie coraz bardziej zaawansowanych algorytmów przetwarzania obrazu. Jednym z kluczowych elementów innowacji jest detekcja pasa ruchu, która stanowi fundament bezpiecznego i skutecznego sterowania pojazdem. W niniejszej pracy opisano implementację sterowania autonomicznym pojazdem w symulatorze CARLA, wykorzystując między innymi sieć neuronową do detekcji pasów ruchu, wybraną na podstawie analizy dostępnych rozwiązań oraz algorytm sterowania MPC (Model Predictive Control).

Kluczowym elementem rozwiązania jest algorytm detekcji pasów ruchu oparty na metodzie segmentacji semantycznej o nazwie LaneNet. Zaproponowane podejście ma charakter modułowy: każdy komponent systemu - detekcja pasa ruchu, planowanie trasy, sterowanie, wizualizacja - został zaimplementowany jako niezależna klasa w języku Python. Takie rozwiązanie umożliwia niezależne testowanie poszczególnych części, zapewnia elastyczność oraz ułatwia przyszły rozwój.

Oprócz projektu i implementacji, w pracy przedstawiono również proces trenowania i walidacji sieci neuronowej LaneNet na zbiorze danych TuSimple, zawierającym obrazy dróg z oznakowaniem pasów ruchu. W trakcie projektowania i weryfikacji systemu zwracano szczególną uwagę na wydajność algorytmu oraz jego zdolność do działania w czasie rzeczywistym, co jest kluczowym wymaganiem w zastosowaniach autonomicznej jazdy. Zaprezentowano również wyniki, a także omówiono problemy, jakie wystąpiły w trakcie implementacji. Ponadto opisano proces weryfikacji całego rozwiązania w kontekście przygotowanych scenariuszy testowych, obejmujących różne wymagające warunki drogowe, z którymi w większości przypadków algorytm poradził sobie pozytywnie.

Słowa kluczowe: detekcja pasów ruchu, MPC, autonomiczne pojazdy, planowanie tras, CARLA, sieci neuronowe

Design and implementation of a lane detection algorithm for autonomous vehicles

Abstract. In recent years, the automotive industry has been undergoing a dynamic transformation, driven by the development of Advanced Driver Assistance Systems (ADAS) and the pursuit of fully autonomous vehicles. A key factor in this process is the progress in artificial intelligence, particularly in computer vision, which enables the creation of increasingly advanced image processing algorithms. One of the crucial elements in these innovations is lane detection, serving as the foundation for safe and effective vehicle control. This thesis presents the implementation of autonomous vehicle control in the CARLA simulator, leveraging, among other methods, a neural network for lane detection—chosen based on an analysis of available solutions—as well as the Model Predictive Control (MPC) steering algorithm.

The core component of this solution is a lane detection algorithm based on a semantic segmentation method called LaneNet. The proposed approach follows a modular design: each system component—lane detection, path planning, control, and visualization—was implemented as an independent Python class. This architecture facilitates independent testing of each part, ensures flexibility, and simplifies future development.

In addition to the system's design and implementation, this thesis also describes the training and validation process of the LaneNet neural network on the TuSimple dataset, which contains road images with lane markings. Throughout the design and verification stages, particular attention was paid to the efficiency of the algorithm and its ability to operate in real time—both of which are key requirements for autonomous driving applications. The results are presented, along with a discussion of the challenges encountered during implementation. Moreover, the thesis details the verification of the entire solution in the context of the prepared test scenarios, which involved various demanding road conditions. In most cases, the algorithm performed positively under these conditions.

Keywords: lane detection, MPC, autonomous vehicles, path planning, CARLA, neural networks



.....
miejscowość i data

.....
imię i nazwisko studenta

.....
numer albumu

.....
kierunek studiów

OŚWIADCZENIE

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanego z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płycie kompaktowej) oraz treść pracy dyplomowej w module APD systemu USOS są identyczne.

.....
czytelny podpis studenta

Spis treści

1. Wstęp	9
1.1. Wprowadzenie	9
1.2. Motywacja pracy	9
1.3. Struktura pracy	11
2. Wstęp techniczny	12
2.1. Rozwiązania pokrewne	12
2.1.1. Tesla Autopilot	12
2.1.2. Mobileye EyeQ	13
2.1.3. Waymo	14
2.1.4. Autoware	15
2.1.5. Podsumowanie	15
2.2. Algorytm detekcji pasa ruchu	16
2.2.1. Tradycyjne metody przetwarzania obrazu	17
2.2.2. Metody oparte na klasyfikacji obrazów	18
2.2.3. Metody oparte na detekcji obiektów	19
2.2.4. Metody oparte na segmentacji obrazu	21
2.2.5. Podsumowanie	24
3. Projekt systemu	26
3.1. Wybór języka programowania	26
3.2. Klient Carla	27
3.3. Moduł detekcji pasa ruchu	27
3.4. Moduł planowania trasy	28
3.5. Moduł sterowania	29
3.5.1. Model kinematyczny pojazdu	29
3.5.2. Algorytm MPC	30
3.6. Moduł wizualizacji	31
3.7. Moduł główny i plik konfiguracyjny	31
4. Implementacja	32
4.1. Detekcja pasa ruchu	32
4.2. Planowanie trasy	33
4.3. Sterowanie	35
4.4. Dodatkowe informacje	35
4.4.1. System kontroli wersji	36
4.4.2. Dokumentacja	36
4.4.3. Testy	36
4.4.4. Narzędzia CI/CD	38
5. Eksperymenty i wyniki	40

5.1. Model detekcji pasów ruchu	40
5.2. System autonomicznej jazdy	41
6. Podsumowanie	43
Bibliografia	45
Wykaz symboli i skrótów	49
Spis rysunków	50
Spis tabel	50

1. Wstęp

1.1. Wprowadzenie

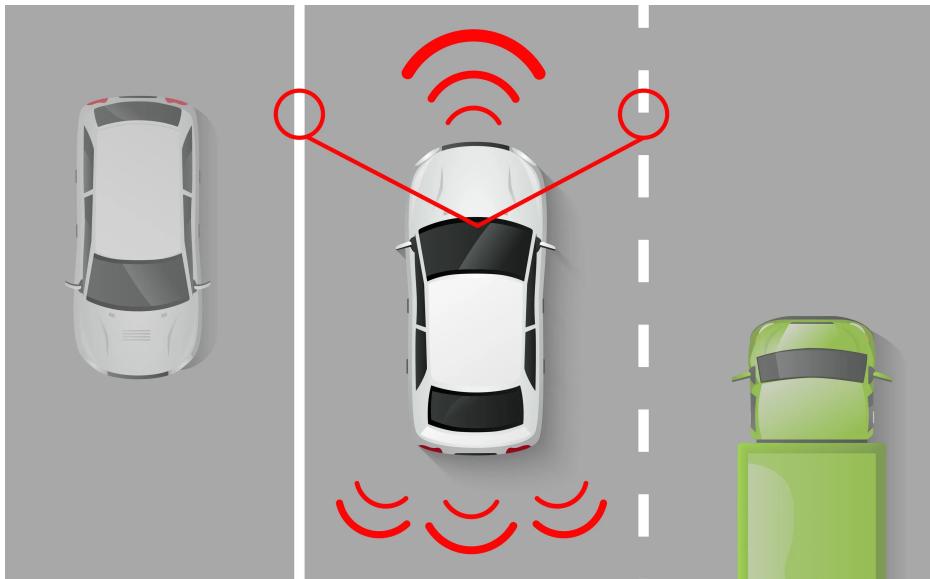
W ostatnich latach rozwój technologii autonomicznych pojazdów zyskał na znaczeniu, stając się jednym z kluczowych aspektów badań i innowacji nowoczesnej motoryzacji. Autonomiczne systemy mają potencjał zrewolucjonizować transport, zwiększając bezpieczeństwo i efektywność [1].

Firmy motoryzacyjne na całym świecie intensyfikują swoje wysiłki w zakresie opracowywania i wdrażania zaawansowanych systemów wspomagania kierowcy (ADAS - *Advanced Driver Assistance Systems*) [2], które zwiększają bezpieczeństwo i komfort jazdy. Warto podkreślić, że Unia Europejska wymaga od producentów wyposażenia nowych pojazdów w określone systemy ADAS, takie jak inteligentny asystent prędkości czy system ostrzegania o niezamierzonej zmianie pasa ruchu, co ma na celu redukcję liczby wypadków drogowych oraz dostosowanie pojazdów do przyszłych standardów autonomii.[3] Systemy te mają na celu nie tylko zwiększenie komfortu jazdy, ale przede wszystkim poprawę bezpieczeństwa poprzez minimalizowanie błędów ludzkich i reagowanie na dynamicznie zmieniające się warunki drogowe. Jednym z kluczowych zadań jest monitorowanie i interpretacja otoczenia, na podstawie tego systemy potrafią podejmować decyzje dotyczące nawigacji i sterowania. Do percepji środowiska wykorzystują różne sensory, takie jak kamery, radary, lidary, systemy GPS, które zapewniają kompleksowy obraz otoczenia. Następnie dane te są przetwarzane i integrowane w celu stworzenia spójnego modelu środowiska. Wiele z tych algorytmów korzysta ze sztucznej inteligencji, w szczególności z technik uczenia maszynowego i głębokiego uczenia, które pozwalają na efektywne analizowanie dużych zbiorów danych pochodzących z sensorów.

Fundamentem takich rozwiązań jest detekcja pasów ruchu. Poprawne wykrywanie i śledzenie pasa pozwala na wyznaczanie odpowiedniej trajektorii poruszania, wykonywanie manewrów, a także płynne reagowanie na dynamicznie zmieniające się otoczenie. Niniejsza praca przedstawia projekt i implementację algorytmu detekcji pasa ruchu w autonomicznym pojeździe, wykorzystując zaawansowane techniki przetwarzania obrazu i sieci neuronowe. Repozytorium z kodem źródłowym znajduje się pod adresem: <https://github.com/Wojtekpob/Autonomous-Driving-System>.

1.2. Motywacja pracy

Celem pracy był projekt i implementacja systemu zdolnego do sterowania pojazdem korzystającego z algorytmu do detekcji pasów ruchu. Podstawę pracy stanowi sieć neuronowa przystosowana do predykcji położenia pasów ruchu bazująca na obrazie z kamery umieszczonej z przodu pojazdu. Kolejny komponent to moduł planowania trasy bazujący na położeniu pasów. Końcowym elementem jest moduł sterowania, który na podstawie wyznaczonej trajektorii oblicza optymalne sterowanie gwarantujące poruszanie się po-



Rysunek 1.1. Lane Keep Assist System – system wspomagania utrzymania pasa ruchu, przykład ADAS. Źródło: [4]

jazdu po wyznaczonej drodze. Opisany system pozwala na poruszanie się samochodu w określonych warunkach - droga ograniczona pasami, bez skrzyżowań, świateł, przejść dla pieszych itd., dzięki czemu może on stanowić podstawowy komponent kompleksowego systemu sterowania. Mając na uwadze szerokie perspektywy rozwoju, system został zaprojektowany w sposób modularny, przez co osobne komponenty można testować, modyfikować, rozszerzać, a także dodawać nowe, zazwyczaj niezależnie od siebie, zachowując jedynie kontrakty pomiędzy poszczególnymi elementami. Do testowania opisanego mechanizmu wykorzystano środowisko symulacyjne CARLA [5]. Dostarcza ono szeroki pakiet czujników, map, możliwości modyfikacji symulacji, a także wszelkich innych narzędzi do rozwoju i testowania systemów dla autonomicznych pojazdów. W ramach integracji ze środowiskiem został zaimplementowany dodatkowy moduł klienta, który zarządza symulacją, pojazdami, a także zbiera dane z czujników.

Motywacją do prowadzenia badań nad tego typu algorytmami jest zwiększenie bezpieczeństwa w ruchu drogowym. Takie systemy zapobiegają niezamierzonym zmianom pasa ruchu. Dodatkowo w połączeniu z innymi modułami, takimi jak adaptacyjny tempostopat czy systemy monitorowania martwego pola, mogą tworzyć kompleksowe metody zwiększania bezpieczeństwa.

Jednym z ważnych aspektów pracy jest unikalne połączenie narzędzi użytych do stworzenia tego systemu, mianowicie symulator CARLA, moduł detekcji pasów ruchu oparty o konwolucyjne sieci neuronowe i moduł sterowania MPC (*Model Predictive Control*). Dzięki temu, a także modularnemu podejściu, rozwiązanie stanowi dobry punkt wyjścia do rozwoju i stworzenia kompleksowego rozwiązania, które będzie zaopatrzone w wielofunkcyjne środowisko testowe.

1.3. Struktura pracy

W niniejszej pracy przedstawiono proces projektowania i implementacji systemu sterowania autonomicznym pojazdem z wykorzystaniem algorytmu detekcji pasa ruchu. Omówiono również napotkane problemy, zastosowane rozwiązania oraz przeprowadzone testy i eksperymenty na różnych etapach projektu.

W rozdziale **Wstęp techniczny** [2] dokonano przeglądu rozwiązań pokrewnych oraz przeglądu algorytmów detekcji pasa ruchu, co stanowiło podstawę do zaprojektowania systemu. Następnie, w rozdziale **Projekt systemu** [3], opisano decyzje projektowe, w tym podział systemu na moduły oraz teoretyczne aspekty ich funkcjonowania. W rozdziale **Implementacja** [4] przedstawiono wykonane zadania, napotkane problemy oraz narzędzia inżynierii oprogramowania użyte podczas realizacji projektu. W kolejnym rozdziale, **Eksperymenty i wyniki** [5], zaprezentowano proces walidacji systemu i algorytmu detekcji pasów ruchu oraz ukazano ich wyniki. W ostatnim rozdziale **Podsumowanie** [6] przedstawiono kluczowe elementy pracy oraz kierunki jej przyszłego rozwoju.

2. Wstęp techniczny

2.1. Rozwiązania pokrewne

Celem niniejszego projektu była implementacja sterowania autonomicznego pojazdu na podstawie algorytmu detekcji pasa ruchu. W tym podrozdziale przedstawiono przegląd i porównanie rozwiązań, które powszechnie określane są mianem autopilotów. Choć zadania realizowane przez te systemy są bardziej rozbudowane, obejmując między innymi planowanie trasy z uwzględnieniem wielu czynników (np. położenia innych pojazdów lub znaków drogowych), to każde z nich zawiera kluczowe moduły odpowiedzialne za detekcję pasów, sterowanie oraz planowanie trasy. Przybliżenie tych rozwiązań pozwala określić miejsce opracowanego projektu w kontekście istniejących systemów.

2.1.1. Tesla Autopilot

Jednym z najbardziej znanych systemów jest Tesla Autopilot, który opiera się na analizie obrazów z ośmiu kamer, a także radarów i ultradźwięków. Na ich podstawie budowana jest mapa otoczenia, dane z każdej z kamer trafiają do konwolucyjnych sieci neuronowych, w których przeprowadzana jest ekstrakcja cech. Następnie cechy stanowią wejście transformera¹, który tworzy z nich przestrzeń wektorową i umieszcza w perspektywie lotu ptaka (ang. *BEV - Bird's Eye View*). Przestrzeń, na bazie której podejmowane są decyzje, jest wynikiem aktualnej percepji, przestrzeni czasowej oraz przestrzeni otoczenia. Historyczne dane są przechowywane w kolejkach, a system w określonych momentach generuje mapy otoczenia. Mapa otoczenia przydaje się w momentach, gdy pojazd nie porusza się (np. skrzyżowanie, gdy światło jest czerwone).

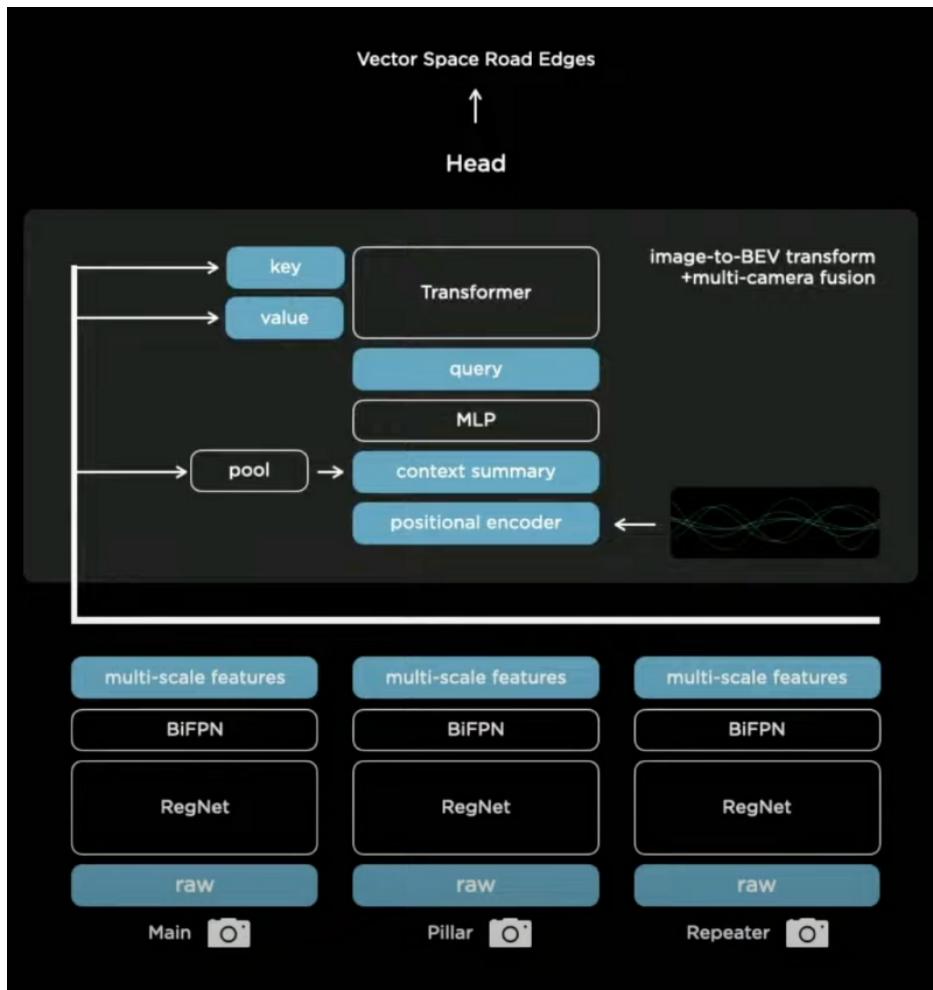
W kolejnym etapie przetwarzania przestrzeni wektorowej Tesla wykorzystuje sieć neuronową do generowania poleceń sterujących pojazdem. Sieć analizuje dane i na ich podstawie bezpośrednio wyznacza działania, takie jak przyspieszanie, hamowanie czy kierowanie. Dzięki temu podejściu system działa w pełni end-to-end, eliminując konieczność ręcznego definiowania reguł oraz nadmiernego przetwarzania danych [7].

Tesla Autopilot nie jest otwartym rozwiązaniem. Algorytmy rozwijane są w ramach zamkniętej platformy. Systemy są trenowane na danych zbieranych z pojazdów, które są używane i których użytkownicy wyrazili na to zgodę [8]. Każdy pojazd wyposażony jest w sensory i przesyła dane do chmury w celu ich przetwarzania i analizy. Dzięki temu zbiory są wyjątkowo pokaźne i zawierają ogromną ilość scenariuszy testowych oraz przypadków brzegowych, co umożliwia trening end-to-end.

Informacje o autopilocie Tesli zostały zaczerpnięte z prezentacji *Tesla AI Day*, dostępnej na platformie YouTube, gdzie przedstawiono szczegóły techniczne działania systemu [9].

¹ Transformer to architektura sieci neuronowej oparta na mechanizmie atencji (ang. *attention*), umożliwia ona modelowanie relacji między danymi wejściowymi, poprzez zmianę ich wag.[6]

Dodatkowe informacje i podsumowanie dotyczące algorytmu autopilota zostały zaprezentowane w artykule na stronie Towards AI [10].



Rysunek 2.1. Architektura percepcji autopilota Tesli. Źródło:[10]

2.1.2. Mobileye EyeQ

System Mobileye EyeQ to kluczowy element architektury autonomicznych pojazdów, stosowany przez wielu producentów samochodów na całym świecie, przykładem jest tutaj grupa Volkswagen, której systemy wspomagania kierowcy oparte są właśnie o produkt Mobileye [11]. Modułowe podejście umożliwia użytkownikom dostosowanie systemu do konkretnych potrzeb. Architektura EyeQ integruje dane z różnych czujników, takich jak kamery, radary czy lidary, aby stworzyć spójny obraz otoczenia pojazdu. [12]

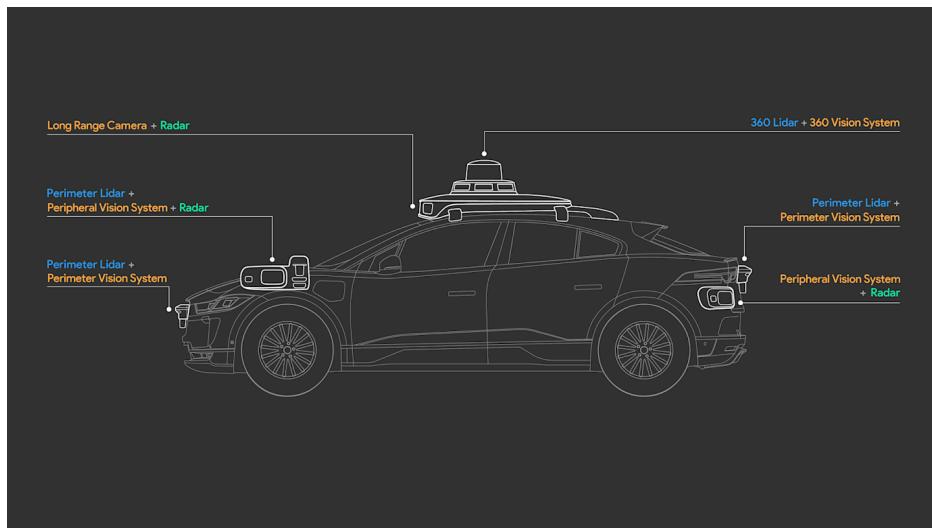
Oprócz systemu Mobileye oferuje również EyeQ Kit – zestaw deweloperski (ang. *SDK - software development kit*), który pozwala producentom samochodów na rozwijanie własnych aplikacji i funkcji w oparciu o podstawową część systemu. Dzięki temu architektura EyeQ jest elastyczna i może być dostosowywana do różnych poziomów automatyzacji – od podstawowych systemów wspomagania kierowcy (ADAS) po w pełni autonomiczną jazdę. [13]

2.1.3. Waymo

Waymo to przedsiębiorstwo zajmujące się rozwojem autonomicznych pojazdów. Projekt rozpoczęła firma Google [14]. System autonomicznego sterowania opiera się na fuzji wielu sensorów, integrowane są dane z lidarów, radarów oraz kamer. Każda grupa z tych sensorów, mimo że współpracuje, to odpowiada głównie za różne zadania. Radarzy wspierają detekcję obiektów i ich prędkości, kamery odpowiadają za rozpoznawanie znaków drogowych, sygnalizacji świetlnej oraz innych uczestników ruchu, takich jak piesi czy inne pojazdy [15]. Kluczową rolę systemu pełnią wysokiej jakości lidary, dostarczające danych z całego otoczenia pojazdu do nawet 300 metrów, w połączeniu z pozostałymi informacjami, umożliwiające tworzenie trójwymiarowych map otoczenia, co pozwala na wykrywanie przeszkód, pieszych i innych pojazdów [16].

Dane z sensorów są przetwarzane przez algorytmy sztucznej inteligencji i uczenia maszynowego, które analizują otoczenie, tworzą predykcje co do przyszłych stanów otoczenia i podejmują decyzje nawigacyjne. System Waymo nie jest otwartoźródłowy, technologie są rozwijane wewnętrznie i nie ma dostępu do kodu źródłowego. Udostępniony jest natomiast zbiór danych Waymo Open Dataset, wspierający badania nad percepcją maszyn i autonomiczną jazdą [17].

Technologie Waymo są wykorzystywane w autonomicznych taksówkach w wybranych miastach USA, firma oferuje usługi przejazdu pojazdami bez kierowcy [18]. Do trenowania systemów Waymo wykorzystywane są dane z rzeczywistych pojazdów oraz symulacje [15]. Warto dodać, że firma pracuje również nad rozwojem modeli end-to-end, takimi jak EMMA (ang. *End-to-End Multimodal Model for Autonomous Driving*), które na bazie danych z sensorów generują bezpośrednio trajektorie pojazdu [19].



Rysunek 2.2. Sensory autonomicznej taksówki firmy Waymo. Źródło:[20]

2.1.4. Autoware

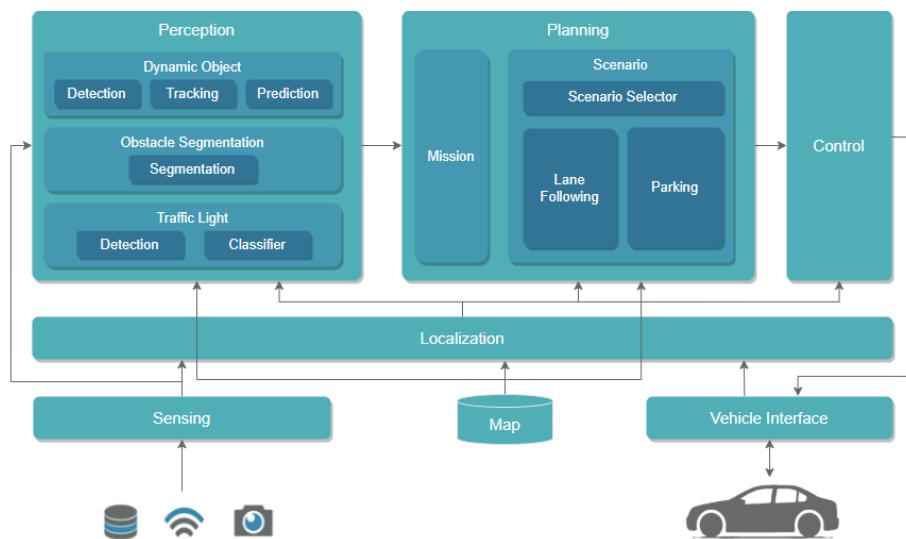
Autoware to otwartoźródłowe oprogramowanie przeznaczone do obsługi pojazdów autonomicznych, rozwijane w oparciu o system ROS (ang. *Robot Operating System*) [21]. Architektura składa się z modułów odpowiedzialnych za percepcję, lokalizację, planowanie trasy oraz sterowanie. Przedstawiona jest na rysunku 2.3.

Do przetwarzania i integracji danych z sensorów Autoware głównie wykorzystywane są sztuczne sieci neuronowe oraz techniki uczenia maszynowego.

Trening systemów Autoware opiera się na danych zbieranych z rzeczywistych pojazdów testowych i publicznych zbiorów danych [22], takich jak KITTI [23]. Dzięki temu możliwe jest doskonalenie systemu w oparciu o różnorodne sytuacje drogowe.

Dzięki modularnej architekturze, Autoware zapewnia rozszerzalność i modyfikowalność modułów, dlatego system można dostosować do specyficznych wymagań, potrzeb, a także różnych zastosowań.

Autoware znajduje zastosowanie w autonomicznym parkowaniu, a także w autonomicznych busach, które poruszają się po ograniczonych obszarach, na przykład kampusach [21].



Rysunek 2.3. Architektura systemu Autoware. Źródło:[24]

2.1.5. Podsumowanie

Opisane systemy posiadają niewątpliwie wiele cech wspólnych, takich jak stosowanie algorytmów sztucznej inteligencji do przetwarzania i integracji danych z sensorów, czy fakt, że każdy z nich ostatecznie spełnia takie same zadania, czyli budowanie mapy otoczenia, planowanie trasy i sterowanie pojazdem. Nie wszystko jednak jest w nich takie samo, różnią się między innymi podejściem do przetwarzania danych z sensorów, źródłem danych treningowych, rodzajem wykorzystywanych sensorów oraz architekturą rozwiązania.

2. Wstęp techniczny

Tesla Autopilot charakteryzuje się podejściem end-to-end, w którym jedna większa sieć odpowiada za tworzenie przestrzeni wektorowej odpowiadającej otoczeniu na podstawie danych z sensorów, natomiast druga, bazując na niej, oblicza ostateczne sterowanie dla pojazdu. Nie są wymagane manualne algorytmy interpretujące i integrujące obrazy z każdej kamery do percepcji otoczenia, ponieważ wszystkim zajmują się wspomniane sieci. Rozwiązanie Tesli nie jest otwartoźródłowe, jest rozwijane wyłącznie dla pojazdów tej marki, a dane wykorzystywane do treningu gromadzone są od użytkowników pojazdów.

Mobileye EyeQ stosuje podejście modułowe, umożliwia ono integrację danych z różnych sensorów, takich jak kamery, radary i lidary, co zapewnia elastyczność systemu, jednak wymaga bardziej złożonego przetwarzania danych. Głównym celem Mobileye jest wsparcie przemysłu motoryzacyjnego, ich rozwiązania znajdują zastosowanie w systemach ADAS wielu producentów pojazdów. Nie jest otwartoźródłowa, natomiast udostępnia SDK, które umożliwia tworzenie własnych komponentów systemu.

Waymo do percepcji również wykorzystuje kamery, radary i lidary, w swoim rozwiązaniu stawia na jakość zbieranych informacji, ponieważ wykorzystuje kamery o bardzo wysokiej rozdzielczości oraz lidary o dalekim zasięgu. Rozwiązanie samo w sobie nie jest otwartoźródłowe, natomiast część wykorzystywanych danych treningowych jest udostępniona w zbiorze danych Waymo Open Dataset [17]. Firma skupia się na rozwoju autonomicznych taksówek, warto zauważyć fakt, że tylko to rozwiązanie z wymienionych jak do tej pory może poszczycić się skomercjalizowaną jazdą bez udziału kierowcy.

Autoware wyróżnia się jako otwartoźródłowy projekt. Dzięki modułowemu podejściu oferuje elastyczność oraz możliwość modyfikacji i rozwijania systemu przez użytkowników. Wykorzystywane sensory to kamery i lidary, jednak użytkownicy w swoich rozwiązaniach mogą rozszerzać system o dodatkowe źródła. Do treningu wykorzystuje publiczne zbiory danych oraz dane z realnych pojazdów testowych. Dzięki swojej charakterystyce jest odpowiedni do badań i eksperymentów nad autonomicznymi pojazdami.

Tabela 2.1. Porównanie rozwiązań autonomicznej jazdy

Kryterium	Tesla Autopilot	Mobileye EyeQ	Waymo	Autoware
Open source	Nie	Nie	Nie	Tak
End-to-end	Tak	Nie	Częściowo	Nie
Sensory	Kamery, radary, ultradźwięki	Kamery, radary, lidary	Kamery, radary, lidary	Kamery, lidary
Źródła danych	Dane z użytkowników	Dane producentów	Dane rzeczywiste i symulacje	Publiczne zbiory i testy
Zastosowanie	Pojazdy Tesla	Wspomaganie producentów	Taksówki autonomiczne	Badania i eksperymenty
Najważniejszy sensor	Kamery	Kamery	Lidary	Kamery

2.2. Algorytm detekcji pasa ruchu

Kluczowym elementem projektu jest algorytm detekcji pasa ruchu. Ważne, aby takie rozwiązanie cechowało się wysoką precyzją i niezawodnością, ponieważ błędna detekcja może prowadzić do niebezpiecznych manewrów, takich jak niezamierzona zmiana pasa ruchu czy wypadnięcie z drogi. Jest to szczególnie istotne w systemie, w którym nie występuje redundancja w sensorach, przez co algorytm polega wyłącznie na jednym źródle

informacji, w tym przypadku na obrazie z kamery. Oprócz tego algorytm musi być odporny na przesłonięcia pasa wynikające z obecności innych pojazdów, cieni, zanieczyszczeń na drodze lub trudnych warunków atmosferycznych, takich jak mgła czy deszcz. Dodatkowo, detekcja musi działać w czasie rzeczywistym, aby umożliwić natychmiastową reakcję na nieoczekiwane sytuacje na drodze.

Biorąc pod uwagę kluczowe znaczenie wyboru odpowiedniego algorytmu, w tej sekcji zostaną przedstawione i porównane wybrane rozwiązania stosowane w detekcji pasa ruchu na przestrzeni lat. Analiza pozwoli na wybór algorytmu najbardziej odpowiedniego do realizacji postawionego zadania.

2.2.1. Tradycyjne metody przetwarzania obrazu

Tradycyjne metody przetwarzania obrazu stanowiły fundament początkowych algorytmów detekcji pasa ruchu. Korzystając z metod analizy obrazu, takich jak detekcja krawędzi, potrafiły tworzyć celne predykcje na temat położenia pasów.

Jedna z metod opiera się na Transformacji Hougha [25], która jest techniką przetwarzania obrazu stosowaną do wykrywania obiektów matematycznie opisanych (np. linii, okręgów, elips). Algorytm oparty na tej metodzie rozpoczyna się od detekcji krawędzi, w której skład wchodzą: wygładzenie obrazu za pomocą filtra Gaussa w celu redukcji szumów, obliczenie gradientu obrazu, co oznacza znalezienie obszarów zmian jasności (przez co lokalizowane są krawędzie), oraz progowanie, które eliminuje słabe krawędzie i pozostawia jedynie te o wysokim kontraście. Następnie zastosowana jest Transformacja Hougha, która działa poprzez zamianę punktów w przestrzeni obrazowej na przestrzeń parametrów. Linia w obrazie jest opisana wzorem:

$$y = mx + c,$$

transformacja Hougha używa reprezentacji biegunowej:

$$\rho = x \cos \theta + y \sin \theta,$$

gdzie:

- ρ to odległość linii od początku układu współrzędnych,
- θ to kąt nachylenia normalnej do linii względem osi x .

Dzięki tej zamianie, każdy punkt (x, y) na obrazie generuje krzywą w przestrzeni (ρ, θ) , zgodnie z równaniem:

$$\rho = x \cos \theta + y \sin \theta.$$

Następnie tworzona jest dyskretna macierz w przestrzeni (ρ, θ) zwana akumulatorem, w której każda z komórek odpowiada konkretnej kombinacji wartości ρ i θ . Punkty krawędziowe w obrazie (wykryte przez algorytm do detekcji krawędzi) głosują na parametry (ρ, θ) , które definiują możliwe linie przechodzące przez te punkty. Linie są wykrywane

jako lokalne maksima akumulatora, oznacza to, że wiele punktów krawędziowych leży na tej samej prostej. Wykryte maksima są progowane określoną wartością przed ostatecznym wyborem linii.

Problemem tego rozwiązania były trudności w wykrywaniu linii zakrzywionych oraz zawodność w przypadku linii mocno przesłoniętych, na przykład przez samochody. Dodatkowo wyzwaniem był brak odporności na zmienne warunki oświetleniowe, takie jak cienie czy słabe światło w nocy, które prowadziły do błędного wykrywania lub niewykrywania pasów.

2.2.2. Metody oparte na klasyfikacji obrazów

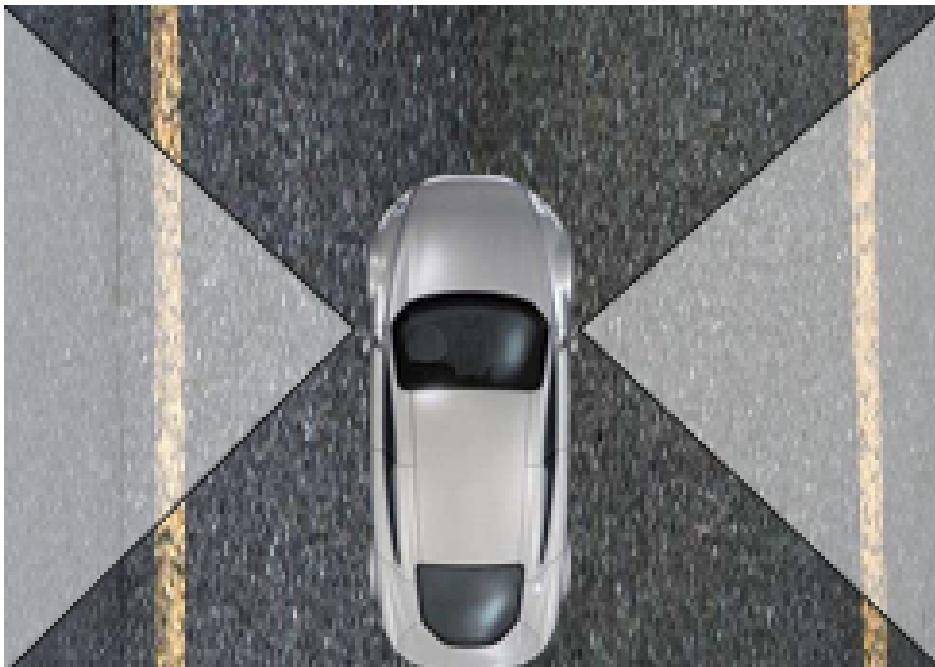
Klasyfikacja obrazu to jedno z wielu zastosowań sieci neuronowych i uczenia maszynowego. Celem klasyfikacji jest nauczenie modelu (np. sieci neuronowej) rozpoznawania cech każdej klasy i poprawne przypisanie jej do jednej z kategorii. Elementem, który pozwala sieci wskazać jedną z klas, jest funkcja wyjściowa softmax, której produktem jest wektor liczb, interpretowany jako wektor prawdopodobieństw, gdzie każdy z indeksów wektora odpowiada konkretnej klasie. Predykcja to indeks maksymalnego elementu wektora.

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^N e^{z_j}}$$

- z_i : i-ta wartość wejściowa.
- Wynik funkcji softmax to rozkład wartości, gdzie wszystkie liczby są znormalizowane do przedziału $[0, 1]$ i ich suma wynosi 1.

DeepLane [26] to algorytm detekcji pasa ruchu oparty na klasyfikacji obrazu za pomocą konwolucyjnej sieci neuronowej (ang. *CNN - Convolutional Neural Network*). Służy do wskazywania pasa ruchu na podstawie obrazu z kamer ustawionych z boków pojazdu, skierowanych w dół, jak na rysunku 2.4. Najważniejsze elementy sieci to 2 warstwy konwolucyjne i wyjściowa warstwa softmax. Oprócz tego w jej skład wchodzą warstwy normalizacji, poolingu i warstwy w pełni połączone (ang. *fully connected*). Wyjście sieci to wektor o wielkości 317, a predykcja to indeks maksymalnej wartości w wektorze; indeksy większe niż 0 odpowiadają położeniu pasa ruchu na wejściowym obrazie, a indeks zerowy oznacza brak wykrytego pasa ruchu.

Algorytm charakteryzuje się wysoką skutecznością, poprawnie przewidując aż 97,85% położień pasa ruchu przy założeniu 2-pikselowej granicy błędu oraz 99,05% przy 5-pikselowej granicy. Sieć jest niewielka i mało wymagająca obliczeniowo, co pozwala na szybkie i efektywne predykcje. Jej głównym ograniczeniem jest jednak specyficzne zastosowanie — potrafi wykrywać jedynie proste linie na podstawie danych pochodzących z odpowiednio ułożonych kamer. W efekcie nie nadaje się do sterowania pojazdem wyłącznie w oparciu o jej predykcje. Dodatkowo sieć jest w stanie wykrywać tylko jedną linię jednocześnie.



Rysunek 2.4. Ułożenie kamer dla DeepLane. Źródło:[26]

2.2.3. Metody oparte na detekcji obiektów

Detekcja obiektów to kluczowe zagadnienie w dziedzinie wizji komputerowej, łączące zadanie klasyfikacji oraz lokalizacji. Celem detekcji jest określenie, jakie elementy znajdują się na obrazie i przewidzenie współrzędnych prostokątów ograniczających (ang. *bounding boxes*) danego obiektu. To podejście ma szerokie zastosowania w dziedzinie autonomicznych pojazdów, wykorzystuje się je do detekcji pieszych, pojazdów, znaków i innych elementów ruchu.

EELane [27] to algorytm bazujący na detekcji obiektów, jest to implementacja rozwiązania OverFeat [28] dla problemu detekcji pasów ruchu i pojazdów. Algorytm składa się z sieci neuronowej, a twórcy dzielą funkcjonalność na trzy - klasyfikację, detekcję i lokalizację. Każda z tych gałęzi korzysta ze wspólnych warstw konwolucyjnych odpowiedzialnych za ekstrakcję cech, a różnice pojawiają się w końcowych warstwach odpowiedzialnych za konkretny rodzaj wyniku. W OverFeat sieć korzysta z techniki okna przesuwnego (ang. *sliding window*), które polega na analizowaniu po kolejnych mniejszych częściach obrazu w celu umożliwienia wykrycia wyłącznie jednej klasy w danym kroku. Algorytm jest jednak zoptymalizowany tak, by nie powielać obliczeń dla każdego okna, dlatego takie same działania są wykorzystywane ponownie w kolejnych krokach. Oprócz tego wykorzystane jest przetwarzanie wieloskalarnie, co oznacza, że ten sam proces okna przesuwnego przeprowadzany jest w różnych rozdzielczościach obrazu. Dzięki temu sieć może wykrywać obiekty różnej wielkości i odległości od kamery.

- Gałąź klasyfikacji wskazuje jaka klasa znajduje się w konkretnym wycinku obrazu

2. Wstęp techniczny

- Gałąź lokalizacji zwraca współrzędne obiektu, jeśli został wykryty na obrazie
- Gałąź detekcji akumuluje wyniki, poprzez redukowanie pokrywających się detekcji i łączy ze sobą wykryte ramki pochodzące z różnych kroków

EELane był trenowany na stworzonym przez autorów zbiorze danych nieudostępnionym publicznie. Ostateczne wyjście sieci odpowiedzialne za predykcję położenia pasa ruchu to sześcioelementowy wektor, pierwsze cztery wartości to współrzędne x, y początku i końca pasa ruchu, natomiast dwie ostatnie to odległość początku i końca pasa od kamery.

VPGNet - Vanishing Point Guided Network [29] to kolejne wyróżniające się podejście z rodziny algorytmów opartych o detekcję obiektów. Algorytm spełnia funkcje detekcji oznaczeń drogowych i pasów ruchu. Nowość wprowadzona przez twórców to predykcja punktu zbieżności pasów ruchu, stąd pochodzi nazwa. Sieć najpierw przewiduje, gdzie znajduje się wspomniany punkt zbieżności, a następnie wykorzystuje go do poprawienia detekcji pasów i oznaczeń drogowych. VPGNet, podobnie jak opisywany wyżej EENet, to rozwiązanie wielozadaniowe, którego struktura posiada wiele gałęzi, natomiast dzielą one wspólne warstwy ekstrakcji cech. Zalicza się do nich predykcja punktu zbieżności, lokalizacja pasów i znaków oraz klasyfikacja obiektów.

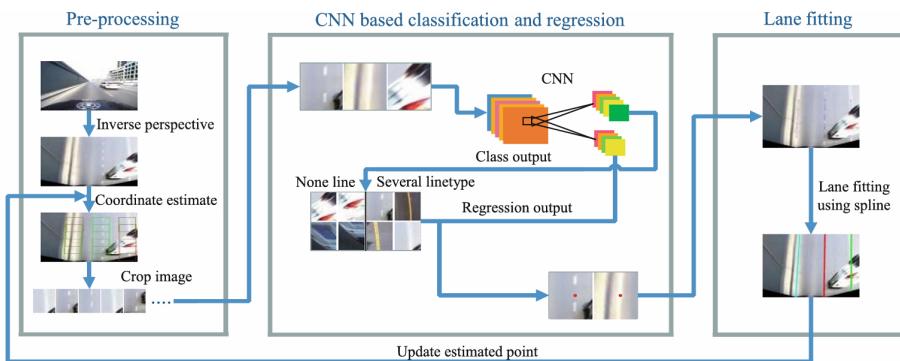
Zbiór treningowy został stworzony przez autorów, nie jest udostępniony publicznie. Wymagany jest specyficzny format danych treningowych, ponieważ w predykcji przewidywany jest punkt zbiegu, który nie jest oznaczony w większości zbiorów do treningu systemów percepcji autonomicznych pojazdów.

STLNet - Spatial Temporal Based Lane Detection Network (Sieć neuronowa oparta na czasie i przestrzeni) [30] to sieć neuronowa służąca do detekcji i klasyfikacji pasów ruchu. STLNet kładzie nacisk na integrację aktualnych danych z tymi pochodzącymi z przeszłości, co zwiększa dokładność i szybkość detekcji w kolejnych sekwencjach predykcyjnych. Wyróżniającym elementem rozwiązania jest preprocessing obrazu poprzez transformację do widoku z lotu ptaka (ang. *BEV - Bird's Eye View*). Ta operacja pomaga zachować proporcje geometryczne i spójność wizualną, przez co detekcja zakrętów i dalszych elementów drogi jest ułatwiona.

Jak wszystkie z wymienionych do tej pory sieci, STLNet składa się z warstw konwolucyjnych odpowiedzialnych za ekstrakcję cech. Podobnie jak w VPGNet [29] i EELane [27] wyodrębnione cechy są wykorzystywane przez kilka gałęzi odpowiedzialnych za różne zadania:

- klasyfikacja typu pasa (ciągły, przerywany, podwójny, żółty, biały itp.),
- lokalizacja współrzędnych pasa,
- ocena kontekstu czasowo-przestrzennego, czyli jak pozycja pasa może zmieniać się w czasie.

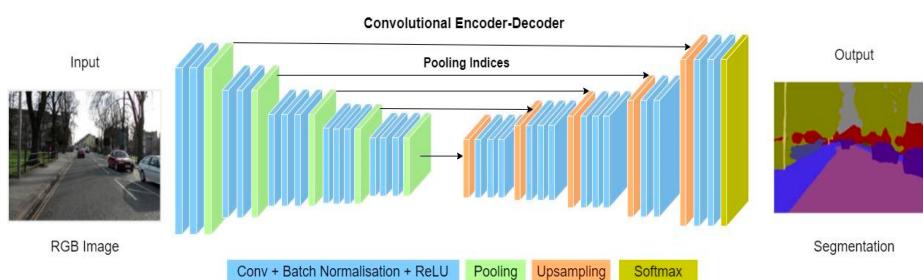
Przebieg algorytmu składa się z kilku kroków i został zaprezentowany na 2.5. Najpierw przeprowadzana jest transformacja do widoku z lotu ptaka oraz oszacowanie przybliżonego położenia granic pasów ruchu. Następnie, przy użyciu sieci neuronowej CNN, wykonywana jest klasyfikacja typu pasów ruchu (np. linia ciągła, przerwywana) oraz regresja precyzyjnych współrzędnych pozycji granic pasów. W ostatnim kroku wyniki predykcji CNN są optymalizowane i odbywa się dopasowanie pasów, co pozwala uzyskać spójne i dokładne odwzorowanie pasa ruchu [30].



Rysunek 2.5. Przebieg algorytmu STLNet. Źródło:[30]

2.2.4. Metody oparte na segmentacji obrazu

Segmentacja obrazu to jedno z popularnych zadań sztucznej inteligencji, opiera się na podziale obrazu na znaczące regiony według zdefiniowanych wcześniej klas. W praktyce oznacza to klasyfikację każdego z pikseli. Architektura takich rozwiązań zazwyczaj składa się z enkodera, który odpowiada za ekstrakcję cech, i dekodera, który tworzy z nich predykcje. Przykład takiej architektury został zaprezentowany na rysunku 2.6. Zadanie segmentacji jest powszechnie używane w systemach autonomicznych pojazdów, pełniąc kluczową rolę w percepcji otoczenia. Może służyć do detekcji pieszych, pojazdów, świateł drogowych, dróg (np. oddzielania poboczy od pasów ruchu).



Rysunek 2.6. Architektura sieci typu enkoder-dekoder używana do segmentacji na przykładzie SegNet [31]. Źródło:[32]

2. Wstęp techniczny

CNN-LSTM [33] to pierwszy z wybranych do analizy algorytmów, którego celem jest segmentacja obrazu, algorytm łączy konwolucyjne sieci neuronowe i komórki pamięci LSTM². Architektura opiera się na wspomnianym wcześniej enkoderze i dekoderze, których struktura wywodzi się z UNet [34] lub SegNet [31] (obie były testowane), ale wyróżnia się zastosowaniem komórek LSTM pomiędzy tymi dwoma strukturami. Dzięki temu sieć uwzględnia dane nie tylko z bieżącego obrazu, lecz także z kilku poprzednich klatek. Sprawia to, że algorytm jest przystosowany do detekcji pasów ruchu w filmach lub w czasie rzeczywistym, gdzie kluczowe jest uwzględnienie informacji z kilku kolejnych momentów wideo. Dzięki temu możliwe jest zachowanie spójności predykcji oraz poprawa dokładności detekcji w przypadku chwilowego zasłonięcia pasów ruchu przez inne pojazdy, zmienne warunki pogodowe lub słabą widoczność. Taka funkcjonalność sprawia, że algorytm jest szczególnie przydatny w systemach autonomicznych pojazdów.

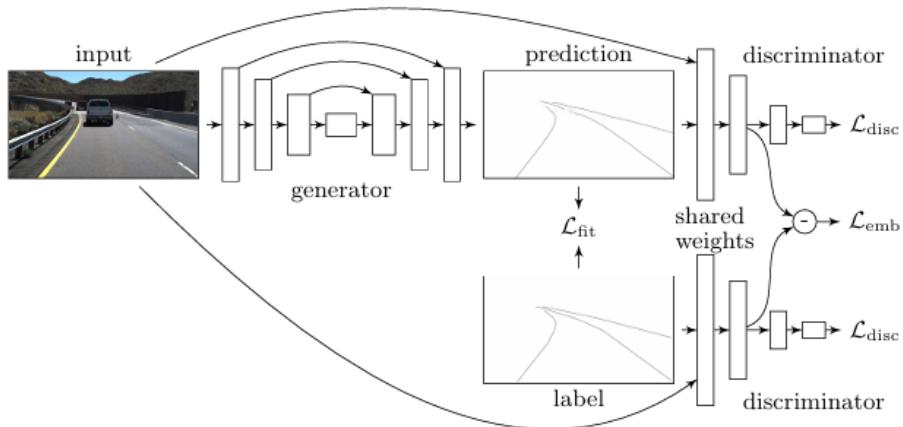
Do treningu sieci autorzy użyli zbioru danych TuSimple [35] rozszerzonego o własne oznaczone dane.

EL-GAN [36] to kolejne rozwiązańe detekcji pasa ruchu oparte o segmentację obrazu, nazwa pochodzi od architektury GAN - Generative Adversarial Network [37], czyli generatywna adwersarialna sieć neuronowa, na której opiera się EL-GAN. Głównym założeniem GAN-ów jest podział sieci na generator, którego celem jest tworzenie danych jak najbardziej zbliżonych do rzeczywistych, oraz dyskryminatora, którego zadaniem jest odróżnianie danych syntetycznych wygenerowanych przez generator od realnych. Typowym zastosowaniem takich architektur jest generowanie obrazów, gdzie wytrenowany generator, na podstawie losowego wektora o określonej długości, tworzy nowy obraz związany z danymi, na których został przeszkolony. W przypadku EL-GAN, zamiast standardowego generowania obrazów, sieć wykorzystuje obraz do stworzenia maski segmentacji z oznaczonymi pasami ruchu.

Drugi nowatorski aspekt EL-GAN stanowi specjalna funkcja straty osadzeń (ang. *embedding loss*), polegająca na obliczaniu straty na podstawie różnicy w cechach obrazów, czyli po etapie ekstrakcji cech, a nie różnicy w samych obrazach. Taka funkcja straty ma pomóc w uwzględnianiu strukturalnych danych obrazów, powoduje ona, że gdy jeden piksel był oznaczony jako fragment pasa ruchu, a algorytm nie wykryje go, tylko wykryje piksel obok, to kara za to będzie maksymalna, gdy jednak zastosowana jest ta funkcja, po ekstrakcji cech prawdopodobnie ten element zostanie zaklasyfikowany jako poprawnie wykryty.

Algorytm był trenowany i testowany wyłącznie na zbiorze danych TuSimple [35].

² LSTM - Long Short-Term Memory to rodzaj rekurencyjnej sieci neuronowej używany do przetwarzania sekwencji danych, takich jak język naturalny, czy szeregi czasowe (np. dane giełdowe). Kluczową cechą LSTM jest umiejętność zapamiętywania i zapominania informacji z przeszłości oraz wykorzystywanie ich do predykcji.

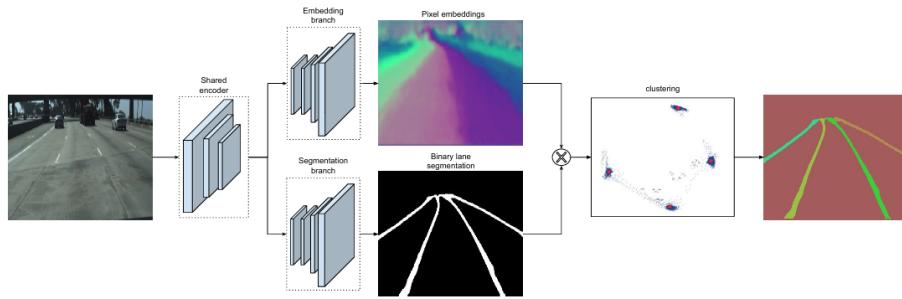


Rysunek 2.7. Architektura EL-GAN, ilustrująca trening generatora i dyskryminatora. Źródło:[36]

LaneNet [38] jest kolejnym algorytmem w dziedzinie segmentacji obrazu, służącym do predykcji położenia pasów, wykorzystując segmentację instancji. Oprócz wyznaczenia tła i pasów ruchu, algorytm kategoryzuje każdy pas jako osobny obiekt. Charakterystycznym elementem rozwiązania jest podział funkcjonalności sieci na gałąź segmentacji, odpowiadającą generowaniu binarnej maski predykcji, gdzie pasy ruchu oddzielone są od reszty obrazu, i gałąź przydzielania instancji, która grupuje w klasy piksele wykryte przy segmentacji binarnej. Sieć wykorzystana do segmentacji to ENet [39].

Dodatkowym usprawnieniem, jakie wprowadza LaneNet, jest sieć neuronowa H-Net. Jej zadaniem jest dostosowanie obrazu w taki sposób, aby linie pasów ruchu mogły być modelowane wielomianami drugiego lub trzeciego stopnia. H-Net działa podobnie do transformacji BEV. Dzięki niej pasy, które przed zmianą perspektywy zbiegają się na horyzoncie, są od siebie oddzielone. H-Net gwarantuje lepszą transformację perspektywy niż klasyczna metoda BEV, ponieważ jest dostosowywana do konkretnego obrazu. W przeciwieństwie do BEV, która opiera się na statycznych parametrach kamery (takich jak kąt nachylenia czy wysokość), H-Net uczy się transformacji perspektywy bezpośrednio na danych treningowych. Dzięki temu może uwzględniać zmienne warunki, takie jak nachylenie drogi czy zakręty, które mogą wpływać na geometrię pasów ruchu. Produktem treningu sieci H-Net jest macierz transformacji. Dzięki tak przygotowanym danym algorytm w ostatnim kroku może metodą aproksymacji wyznaczyć wielomiany odpowiadające pasom ruchu. Proces z architekturą sieci przedstawiony jest na rysunku 2.8.

Podsumowując, kroki działania LaneNet to: segmentacja binarna pikseli pasów, zgrupowanie pikseli do odpowiednich instancji pasów, transformacja perspektywy za pomocą H-Net, dopasowanie wielomianów do pasów w przekształconej przestrzeni oraz transformacja do oryginalnej przestrzeni. Do treningu został wykorzystany zbiór danych TuSimple [35].



Rysunek 2.8. Schemat architektury i działania sieci LaneNet. Źródło:[38]

2.2.5. Podsumowanie

W niniejszej sekcji zaprezentowano szereg algorytmów detekcji pasów ruchu, zaczynając od tradycyjnych metod opartych na przetwarzaniu obrazu (Transformacja Hougha), przez sieci neuronowe rozwiązujące zadanie klasyfikacji (DeepLane), sieci detekcyjne (EELane, VPGNet, STLNet), po podejścia zajmujące się segmentacją (CNN-LSTM, EL-GAN, LaneNet). Każdy z opisanych algorytmów posiada charakterystyczne podejście do zadania detekcji i ma unikalne elementy oraz mechanizmy pomagające w osiągnięciu celu.

Wspólnym wyzwaniem pozostaje kompromis między wysoką precyzją a wydajnością obliczeniową. Tradycyjne metody przetwarzania obrazu oferują niskie koszty obliczeniowe, jednak są zawodne w wymagających scenariuszach, takich jak zakręty czy ograniczona widoczność. Podejścia wykorzystujące sieci neuronowe lepiej radzą sobie z trudnymi warunkami, ale wymagają dużej ilości danych treningowych oraz specjalnego sprzętu do działania w czasie rzeczywistym. Wybrane cechy opisanych algorytmów zostały przedstawione w tabeli 2.2.

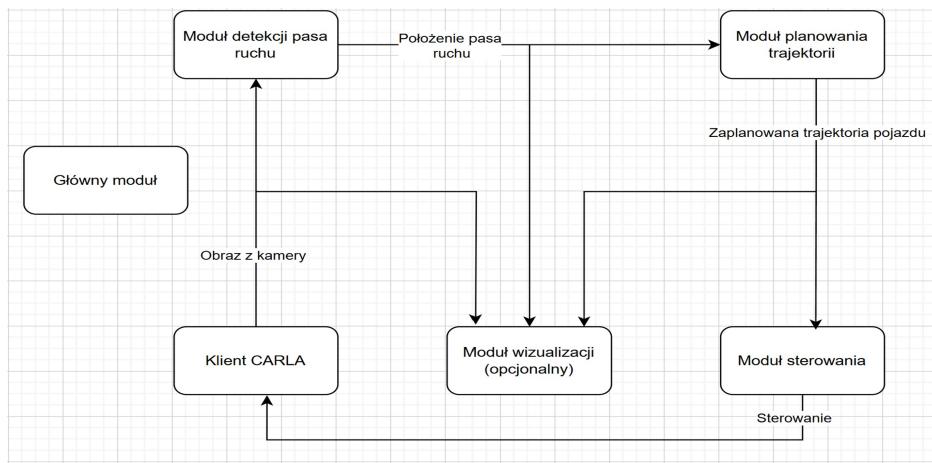
Tabela 2.2. Porównanie wybranych algorytmów detekcji pasa ruchu.

Algorytm	Metoda	Charakterystyczny element	Zbiór treningowy	Preprocessing	Zastosowanie
Transformacja Hougha [25]	Tradycyjna (analiza krawędzi)	Wykrywanie linii w przestrzeni ρ, θ	Brak (klasyczna metoda)	Gauss + detekcja krawędzi	Pasy ruchu (proste linie)
DeepLane [26]	Klasyfikacja	Wektor 317 wyjść, softmax predykcja	Zbiór prywatny (kamery boczne)	-	Pasy ruchu (jedna linia, prosta)
EELane [27]	Detekcja obiektów (OverFeat)	Wieloskalarnie <i>sliding window</i> + 3 gałęzie (klas./lok./det.)	Zbiór prywatny	-	Pasy ruchu, pojazdy
VPGNet [29]	Detekcja obiektów (multi-task)	Detekcja punktu zbiegu pasów	Zbiór prywatny	-	Pasy ruchu, pojazdy, oznaczenia drogowe
STLNet [30]	Detekcja (spatial-temporal) kontekst czasowy	BEV (<i>bird's eye view</i>)	Zbiór prywatny, sekwencje wideo	Transformacja BEV	Pasy ruchu (filmy, sekwencje)
CNN-LSTM [33]	Segmentacja (UNet/SegNet + LSTM)	Warstwa LSTM między enkoderem i dekoderem	TuSimple + dane własne	-	Pasy ruchu (filmy, sekwencje)
EL-GAN [36]	Segmentacja (GAN)	<i>Embedding loss</i> w generatorze	TuSimple	-	Pasy ruchu
LaneNet [38]	Segmentacja instancji (ENet)	H-Net (adaptacyjna perspektywa)	TuSimple	Transformacja H-Net	Pasy ruchu (każdy jako osobna instancja)

3. Projekt systemu

Przy projektowaniu systemu, oprócz głównego celu, jakim była implementacja sterowania autonomicznego pojazdu na podstawie algorytmu detekcji pasa ruchu, należało pamiętać o ważnych elementach, które zagwarantują działanie w czasie rzeczywistym, odporność na okluzje pasa, skalowalność oraz zapewnienie środowiska symulacyjnego do testowania rozwiązania.

W niniejszym rozdziale zostały przedstawione decyzje projektowe oraz architektura, umożliwiające realizację założonego zadania. System został zaprojektowany w sposób modułowy, co pozwala każdemu z komponentów pełnić określona funkcję. Taki układ umożliwił ich niezależne testowanie, łatwą modyfikację oraz integrację nowych elementów, co znaczco ułatwia dalszy rozwój systemu i adaptację do nowych potrzeb. Ostatecznie powstało pięć komponentów: moduł główny, planowania trasy, sterowania, detekcji pasa ruchu, wizualizacji oraz klienta simulatora, których diagram z kanałami komunikacyjnymi został zaprezentowany na rysunku 3.1. Ich funkcje oraz konkretne decyzje implementacyjne zostaną przedstawione w kolejnych sekcjach.



Rysunek 3.1. Schemat architektury zaprojektowanego systemu. Każda komunikacja między modułami odbywa się za pośrednictwem modułu głównego. Zdecydowano się jednak nie przedstawać tego na rysunku, by zachować czytelność ilustracji.

3.1. Wybór języka programowania

Wybór języka programowania dla wszystkich modułów był podyktowany przede wszystkim dostępnością biblioteki do zarządzania symulatorem Carla oraz narzędzi do sztucznej inteligencji, które są wspierane w Pythonie.

Oczywiście istniała możliwość implementacji serwera API, co pozwoliłoby na realizację modułów w różnych językach programowania i zapewnienie większej elastyczności w integracji z innymi systemami. Jednak zdecydowano się na użycie Pythona w całym

projektie, aby uniknąć komplikacji związanych z dodatkową warstwą komunikacji oraz ewentualnymi opóźnieniami.

Oprócz wyboru języka należało również dobrać inne narzędzia w postaci odpowiednich bibliotek wspierających realizację założonych zadań. W projektie zdecydowano się na użycie NumPy [40] oraz OpenCV [41], które dostarczają niezbędnych funkcji do przetwarzania danych i obrazu. NumPy to biblioteka do obsługi macierzy i wykonywania operacji na nich, szeroko wykorzystywana w dziedzinach analizy danych i sztucznej inteligencji. Z kolei OpenCV służy do przetwarzania obrazów i analizy wizyjnej.

3.2. Klient Carla

CARLA [5] to symulator stworzony w celu rozwoju systemów autonomicznej jazdy. Jest to otwarte oprogramowanie działające na silniku Unreal Engine [42], które oferuje realistyczne wirtualne środowisko, narzędzia do symulacji warunków drogowych, modelowania map oraz implementacji i testowania systemów autonomicznych w pojazdach.

CARLA umożliwia symulację różnorodnych warunków drogowych, takich jak ruch miejski, autostrady, skrzyżowania, pogoda oraz różne pory dnia. Kluczowym elementem symulatora jest szeroka gama czujników, takich jak kamery, radary, lidary, GPS oraz wiele innych. Oprócz tego umożliwia użytkownikom tworzenie niestandardowych czujników, które można zaprogramować zgodnie z potrzebami.

Zarządzanie symulacją oraz sterowanie pojazdami odbywa się w architekturze klient-serwer przy użyciu dedykowanej biblioteki w języku Python lub C++. W celu integracji funkcjonalności, takich jak inicjalizacja symulacji i konfiguracja pojazdu, a także sterowanie nim, które są niezbędne do implementacji i testowania algorytmu detekcji pasa ruchu, zdecydowano się na stworzenie klasy odpowiadającej za to.

3.3. Moduł detekcji pasa ruchu

Detekcja pasa ruchu stanowiła kluczowy element opisanego systemu, zależne są od niej moduły sterowania i planowania, co bezpośrednio przekłada się na jakość opracowanego systemu. Wymaganiem, jakie miał spełniać ten moduł, było generowanie linii opisujących pasy ruchu przy wykorzystaniu obrazów pochodzących z kamery umieszczonej na przodzie pojazdu. Na podstawie przeprowadzonego przeglądu opisanego w drugim rozdziale został wybrany LaneNet [38], oparty o metodę segmentacji semantycznej.

Podczas wyboru odpowiedniego algorytmu można było odrzucić metodę z Transformacją Hougha [25], ponieważ miała problemy z wykrywaniem zakrętów i zasłoniętych pasów, przez co widocznie była mniej zaawansowana niż pozostałe. Deeplane [26] również został odrzucony przez jego specyficzne ułożenie kamer, zaimplementowany algorytm musiał korzystać z kamery skierowanej do przodu, by mógł wykrywać przyszłe pasy i odpowiednio planować trasę. W przypadku pozostałych algorytmów najlepszy wybór stanowiłaby sieć będąca optymalnym punktem jakości predykcji i złożoności obliczeniowej,

3. Projekt systemu

która przekłada się na szybkość działania. Niestety, ze względu na brak kompleksowego porównania w dostępnych materiałach obejmujących artykuły naukowe i zasoby internetowe, w których sieci byłyby wytrenowane i przetestowane na jednym zbiorze danych, a ich szybkości predykcji na tych samych zasobach, nie można jednoznacznie stwierdzić, który z algorytmów sprawdziłby się najlepiej do tego zadania. Z tego powodu został wybrany LaneNet, który spośród opisanych algorytmów wydawał się optymalnym rozwiązaniem, zapewniającym kompromis między jakością a szybkością.

Implementacja sieci pochodzi z [43] i jest oparta o bibliotekę PyTorch [44], służącą do budowania i trenowania modeli sztucznej inteligencji. Wybrany zbiór danych do treningu i testowania to TuSimple [35], który został opracowany na potrzeby rozwoju detekcji pasów ruchu. Składa się z nagrani i odpowiadających im etykiet reprezentujących położenie pasa ruchu. Obejmuje 6408 zdjęć o rozdzielcości 1280x720. Jest bogaty w różnorodne scenariusze, obejmujące różne warunki drogowe, takie jak autostrady, proste odcinki, zakręty oraz sceny z częściowo zasłoniętymi pasami, na przykład przez inne pojazdy.



Rysunek 3.2. Przykłady obrazów ze zbioru danych TuSimple [35].

3.4. Moduł planowania trasy



Głównym zadaniem modułu planowania trasy było wygenerowanie wielomianu odpowiadającego trajektorii, jaką miał poruszać się pojazd. Dane wejściowe stanowiły pasy ruchu wykryte przez algorytm detekcji w postaci wielomianów.

3.5. Moduł sterowania

Zadaniem modułu sterowania było generowanie wartości przyspieszenia i kątu skrętu kół, korzystając z zaplanowanej trasy oraz aktualnej pozycji pojazdu. Komponent musiał zarówno zapewniać odpowiednią jakość sterowania, odpowiadającą za podążanie pojazdu za wyznaczoną trajektorią, jak i jego stabilność oraz uwzględniać ograniczenia, takie jak maksymalny kąt skrętu kół i maksymalne przyspieszenie. Wybrany został algorytm MPC, czyli sterowanie predykcyjne na podstawie modelu, który zapewnia wysoką jakość sterowania oraz możliwość uwzględnienia ograniczeń. MPC opiera się na sterowaniu predykcyjnym, czyli przewidywaniu przyszłych stanów systemu na podstawie modelu matematycznego oraz optymalizacji sterowania w zadanym horyzoncie predykcji. Do implementacji modułu sterowania wybrano model kinematyczny, który jest uproszczonym, ale efektywnym rozwiążaniem. Stwierdzono, że model dynamiczny uwzględniający wszystkie siły działające na pojazd jest zbyt skomplikowany i czasochłonny w implementacji dla tego problemu. Model kinematyczny opisuje ruch pojazdu przy pomocy zależności kinematycznych, uwzględniając zmienne takie jak kąt skrętu kół, prędkość, pozycję oraz orientację pojazdu.

3.5.1. Model kinematyczny pojazdu

Zmienne stanu pojazdu w chwili dyskretnej t potrzebne do opisania modelu kinematycznego oznaczamy jako:

$$x_t, \quad y_t, \quad \psi_t, \quad v_t, \quad \text{cte}_t, \quad \text{epsi}_t,$$

gdzie:

- x_t, y_t – współrzędne środka pojazdu w układzie 2D,
- ψ_t – kąt orientacji (yaw),
- v_t – prędkość pojazdu (skalar),
- cte_t (ang. *cross-track error*) – błąd boczny względem trajektorii,
- epsi_t (ang. *orientation error*) – błąd orientacji względem trajektorii.

Zmienne sterowania to:

$$\delta_t \quad (\text{kąt skrętu}), \quad a_t \quad (\text{przyspieszenie/hamowanie}).$$

Przyjęto, że model pojazdu ma parametr L_f oznaczający odległość środka ciężkości od przedniej osi. Horyzont predykcji oznaczony przez N , a okres próbkowania Δt . Postać równania kinematycznego wygląda następująco:

$$\begin{aligned}
 x_{t+1} &= x_t + v_t \cos(\psi_t) \Delta t, \\
 y_{t+1} &= y_t + v_t \sin(\psi_t) \Delta t, \\
 \psi_{t+1} &= \psi_t + \frac{v_t}{L_f} \delta_t \Delta t, \\
 v_{t+1} &= v_t + a_t \Delta t.
 \end{aligned} \tag{1}$$

W celu uwzględnienia błędu bocznego i błędu orientacji względem funkcji opisującej zaplanowaną trasę, przyjęto:

$$\begin{aligned}
 \text{cte}_{t+1} &= (f(x_t) - y_t) + v_t \sin(\text{epsi}_t) \Delta t, \\
 \text{epsi}_{t+1} &= (\psi_t - \psi_{\text{des}}(x_t)) + \frac{v_t}{L_f} \delta_t \Delta t,
 \end{aligned} \tag{2}$$

gdzie $f(x_t)$ jest wartością funkcji opisującej położenie trajektorii w punkcie x_t , a $\psi_{\text{des}}(x_t) = \arctan(f'(x_t))$ oznacza orientację tej trajektorii.

3.5.2. Algorytm MPC

Funkcja kosztu:

$$\begin{aligned}
 J = & \sum_{k=0}^{N-1} \left(w_{\text{cte}} [\text{cte}_k]^2 + w_{\text{epsi}} [\text{epsi}_k]^2 + w_v [v_k - v_{\text{des}}]^2 \right) \\
 & + \sum_{k=0}^{N-1} \left(w_\delta [\delta_k]^2 + w_a [a_k]^2 \right) \\
 & + \sum_{k=0}^{N-2} \left(w_{\Delta\delta} [\delta_{k+1} - \delta_k]^2 + w_{\Delta a} [a_{k+1} - a_k]^2 \right).
 \end{aligned} \tag{3}$$

gdzie:

- $w_{\text{cte}}, w_{\text{epsi}}, w_v, w_\delta, w_a, w_{\Delta\delta}, w_{\Delta a}$ – współczynniki wag,
- v_{des} – prędkość zadana,
- δ_k, a_k – sterowania w chwili k .

Zadanie optymalizacji podlega następującym ograniczeniom:

1. *Warunki początkowe:*

$$x_0 = x_{\text{pocz}}, \quad y_0 = y_{\text{pocz}}, \quad \psi_0 = \psi_{\text{pocz}}, \quad v_0 = v_{\text{pocz}}, \quad \text{cte}_0 = \text{cte}_{\text{pocz}}, \quad \text{epsi}_0 = \text{epsi}_{\text{pocz}}. \tag{4}$$

2. *Ograniczenia sterowań i prędkości:*

$$-\delta_{\max} \leq \delta_k \leq \delta_{\max}, \quad -a_{\max} \leq a_k \leq a_{\max}, \quad 0 \leq v_k \leq v_{\max}, \tag{5}$$

przy czym mogą występować dodatkowe ograniczenia na zmianę sterowania w czasie:

$$-\Delta\delta_{\max} \leq \delta_{k+1} - \delta_k \leq \Delta\delta_{\max}. \tag{6}$$

Pełne zadanie optymalizacji w podejściu MPC można zapisać jako:

$$\min_{\substack{x_k, y_k, \psi_k, v_k, cte_k, \epsilon_k, \\ \delta_k, a_k}} J,$$

z zastrzeżeniem, że:

- (1) Równania (1) – (2) są spełnione dla $k = 0, \dots, N - 1$,
 - (2) Warunki początkowe (4) są spełnione,
 - (3) Ograniczenia (5) oraz (6) są spełnione.
- (7)

Rozwiążanie (7) daje sekwencję $\{\delta_0, a_0, \delta_1, a_1, \dots\}$, spośród której do sterowania pojazdem wybierany jest pierwszy krok (δ_0, a_0) , a następnie zadanie optymalizacji rozwiązywane jest ponownie w kolejnej chwili.

Do implementacji algorytmu sterowania wybrano bibliotekę CasADi [45], która jest otwartoźródłowym narzędziem do optymalizacji równań nieliniowych, dostarcza metody do pisania równań i tworzenia ograniczeń występujących w regulowanym modelu w języku Python.

3.6. Moduł wizualizacji

Moduł wizualizacji miał służyć do obrazowania wyników predykcji pasów ruchu oraz zaplanowanej trasy. Został stworzony w celach monitorowania poprawności działania wspomnianych modułów. Ostateczny produkt wizualizacji stanowił obraz pochodzący z kamery pojazdu z nałożonymi wielomianami odpowiadającymi zaplanowanej trasie i pasom ruchu. Moduł miał móc działać w czasie rzeczywistym, jak i posiadać opcję zapisywania wizualizacji do pliku, aby umożliwić analizowanie wyników po zakończeniu symulacji i porównywanie.

3.7. Moduł główny i plik konfiguracyjny

Zadanie modułu głównego było kluczowe dla poprawnego działania systemu, jego funkcja to przede wszystkim integracja pozostałych komponentów, sprawdzanie poprawności i przesyłanie danych między nimi oraz wywoływanie odpowiednich elementów systemu. Oprócz tego powinien zajmować się inicjalizacją wszystkich modułów z odpowiednią konfiguracją. Przy projektowaniu przyjęto założenie, że plik konfiguracyjny w formacie YAML będzie zawierał wszystkie elementy, których wartości będzie można modyfikować, czyli zmienne takie jak aktualna mapa, uruchomienie wizualizacji, czy wielkość okna kamery i inne, tak żeby zapewnić prosty sposób na kontrolę i możliwość edycji.

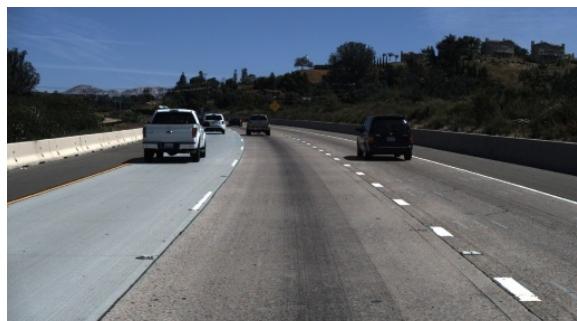
4. Implementacja

W niniejszym rozdziale zostanie opisany proces implementacji konkretnych funkcjonalności, jakie posiada gotowy system. W kolejnych sekcjach przedstawione zostaną szczegóły tworzenia kluczowych komponentów rozwiązania i problemy, jakie należało rozwiązać, żeby osiągnąć zamierzony efekt.

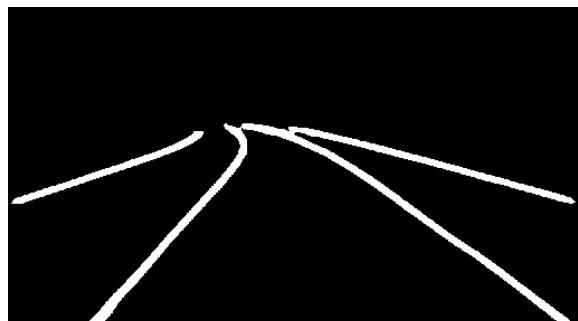
4.1. Detekcja pasa ruchu

Implementacja sieci LaneNet pochodzi z [43], gotowe rozwiązanie posiadało metody do wczytywania zbioru danych, nauczenia i przetestowania sieci oraz narzędzi do monitorowania, nie zawierało jednak opcji predykcji pojedynczego obrazu. Z tego względu, bazując na istniejącej części implementacji, stworzono dodatkowy moduł, który pozwalał na wykonanie predykcji dla pojedynczych klatek obrazu, zgodnie z potrzebami opisanego systemu.

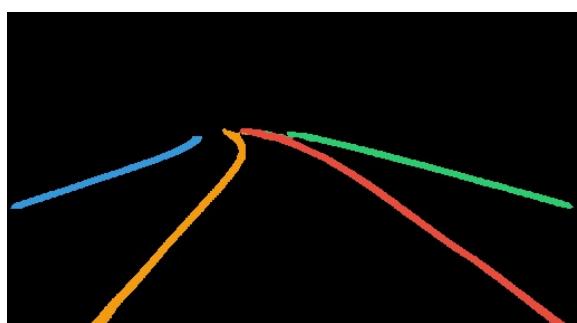
Głównym zadaniem związanym z detekcją pasa ruchu był trening modelu LaneNet na zbiorze danych TuSimple. Po procesie uczenia algorytm osiągnął zadowalające wyniki na zbiorze testowym pochodzący z tego samego źródła - przykładowy wynik predykcji i działanie zostały zaprezentowane na rysunku.



(a) Obraz wejściowy



(b) Predykcja maski segmentacji binarnej

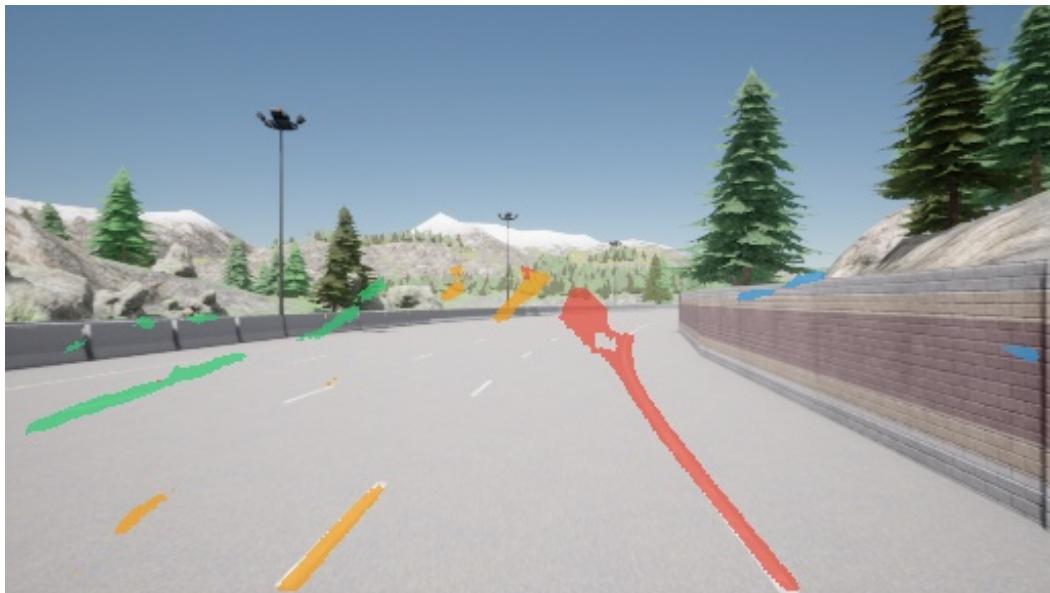


(c) Predykcja instancji



(d) Wynik segmentacji semantycznej nałożony na obraz wejściowy

Rysunek 4.1. Wyniki działania algorytmu LaneNet na podstawie obrazu wejściowego.

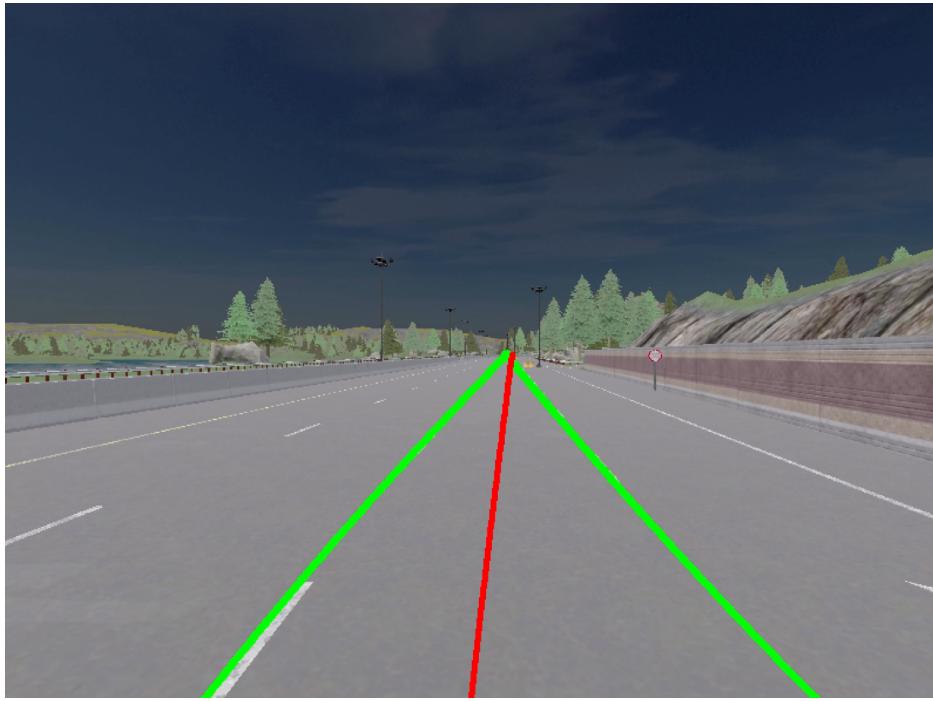


Rysunek 4.2. Segmentacja semantyczna obrazu z symulatora po wytrenowaniu na zbiorze TuSimple.

Niestety, w przypadku danych pochodzących z symulatora, sieć wytrenowana wyłącznie na zbiorze TuSimple nie radziła sobie dostatecznie dobrze, generując liczne błędy w detekcji, przykładowa błędna detekcja została przedstawiona na rysunku 4.2. Z uwagi na problem z generalizacją sieci i rozbieżnością między rzeczywistymi obrazami a tymi pochodzącymi z symulatora, zdecydowano się na zebranie dodatkowych danych treningowych ze środowiska testowego, w tym celu wykorzystano rozwiązanie [46] znalezione w internecie, pozwalające na zautomatyzowanie stworzenia zbioru danych z adnotacjami pochodzącymi z CARLI. Następnie przeprowadzono proces dotrenowania, w którym sieć, już nauczoną do zadowalających efektów na TuSimple, trenowano ponownie na nowo zebranych danych. Dzięki temu osiągnięto zdecydowanie lepsze wyniki, które pozwoliły na implementację dalszej części algorytmu.

4.2. Planowanie trasy

Przewidziane zadanie planowania trasy polegało na dopasowaniu wielomianu, oznaczającego trajektorię, po której poruszać miał się pojazd, na podstawie dwóch linii ograniczających pas ruchu, pochodzących z modułu detekcji. Żeby to osiągnąć, należało wyznaczyć punkty, które znajdowały się na środku pasa (każdy punkt był średnią położenia w poziomie pomiędzy lewą i prawą krawędzią pasa w danej wysokości obrazu), a następnie przeprowadzić aproksymację tych punktów za pomocą wielomianu. Proces ten ilustruje rysunek 4.3, w którym czerwony wielomian reprezentuje wynik dopasowania do środkowej linii pasa ruchu, natomiast linie zielone oznaczają granice pasa.



Rysunek 4.3. Ilustracja z oznaczonymi pasami ruchu i zaplanowaną trasą.

W efekcie dopasowania otrzymujemy równanie wielomianu w układzie współrzędnych obrazu:

$$x_{\text{img}}(y_{\text{img}}) = a_2 y_{\text{img}}^2 + a_1 y_{\text{img}} + a_0, \quad (8)$$

gdzie:

- a_0, a_1, a_2 – współczynniki dopasowanego wielomianu,
- x_{img} – współrzędna pozioma (oś rosnąca w prawo),
- y_{img} – współrzędna pionowa (oś rosnąca w dół),
- początek układu $(0,0)$ znajduje się w lewym górnym rogu obrazu.

Kolejny etap to przekształcenie otrzymanego wielomianu do układu współrzędnych pojazdu, tak by moduł sterowania mógł bezpośrednio z niego korzystać. Dlatego:

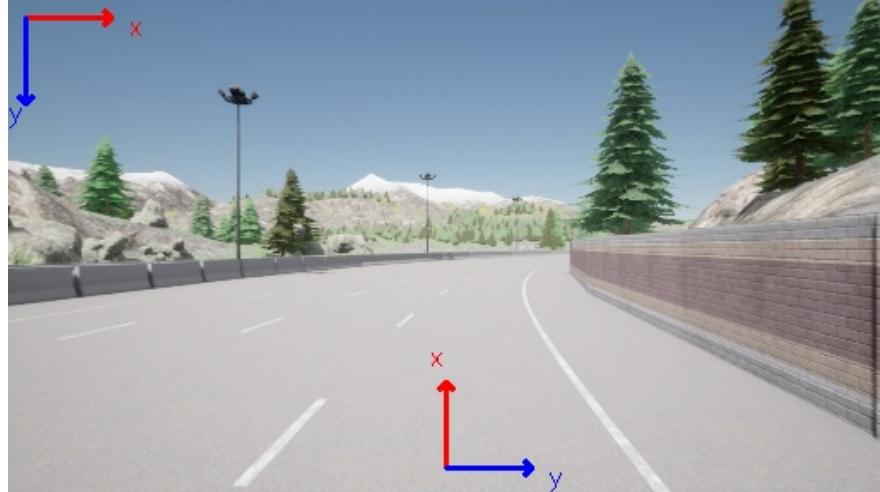
$$W = \text{szerokość obrazu}, \quad H = \text{wysokość obrazu}.$$

Układ współrzędnych pojazdu definiujemy w ten sposób, że punkt $(x_v = 0, y_v = 0)$ odpowiada środkowi dolnej części obrazu, a osiom przypisujemy następujące kierunki:

$$\begin{aligned} x_v &= (H - 1) - y_{\text{img}}, \\ y_v &= x_{\text{img}} - \frac{W}{2}. \end{aligned} \quad (9)$$

Oznacza to, że oś x_v rośnie ku górze (w przeciwnym kierunku do y_{img}), a oś y_v w prawo

(zgodnie z x_{img}). Rysunek 4.4 przedstawia poglądowy schemat obu układów współrzędnych i sposób, w jaki są względem siebie przesunięte.



Rysunek 4.4. Ilustracja układów współrzędnych obrazu (lewy górny róg) i pojazdu (po środku na dole).

Definiujemy równanie w przestrzeni pojazdu:

$$y_v(x_v) = b_2 x_v^2 + b_1 x_v + b_0$$

co po podstawieniu równań z (8) i (9) oraz uproszczeniu daje:

$$\begin{aligned} b_2 &= a_2, \\ b_1 &= -\left(a_1 + 2(H-1)a_2\right), \\ b_0 &= a_0 + a_1(H-1) + a_2(H-1)^2 - \frac{W}{2}. \end{aligned} \quad (10)$$

Dzięki powyższym operacjom moduł planowania może przekazać zaplanowaną trajektorię w układzie współrzędnych pojazdu do komponentu odpowiedzialnego za sterowanie.

4.3. Sterowanie

Implementacja sterowania została zrealizowana w oparciu o zadanie optymalizacji opisane w rozdziale dotyczącym algorytmu MPC, przy użyciu biblioteki CasADi oraz dobraniu wag poszczególnych elementów funkcji kosztu J zdefiniowanej w (3). Wagi te zostały ustalone eksperymentalnie w drodze testów.

4.4. Dodatkowe informacje

W niniejszym podrozdziale zostaną opisane narzędzia użyte do stworzenia systemu, które nie były niezbędne do osiągnięcia celu, natomiast w znacznym stopniu przyczyniły się do polepszenia jego jakości, przyspieszenia procesu implementacji, uproszczenia

4. Implementacja

przyszłego rozwoju oraz poprawienia utrzymywalności kodu. Chodzi o elementy pełniące funkcje pomocnicze w inżynierii oprogramowania, takie jak system kontroli wersji, dokumentacja, narzędzia CI/CD czy testy.

4.4.1. System kontroli wersji

System kontroli wersji jest nieodzownym elementem każdego projektu programistycznego. Pozwala na zarządzanie zmianami w kodzie, śledzenie postępów, zapisywanie historii oraz pracę w zespołach. W projekcie zdecydowano się na wykorzystanie systemu Git [47], jednego z najpopularniejszych i najbardziej wszechstronnych z dostępnych rozwiązań. Według [48], Git jest używany aż w 79% wszystkich repozytoriów, co świadczy o jego dominującej pozycji w branży.

Jako zdalne narzędzie do przechowywania kodu użyto GitHub [49], który oferuje narzędzia do zarządzania zadaniami, przeglądania kodu czy automatyzacji procesów CI/CD.

Przy rozwoju projektu zazwyczaj korzystano z podejścia opartego o gałąź (ang. *branches*), gdzie przy implementacji nowej funkcjonalności najpierw tworzoną osobną gałąź, wdrażano ją, testowano, a następnie integrowano z główną gałęzią za pomocą operacji ‘merge’. Takie podejście pozwala na łatwe przełączanie się między różnymi wersjami kodu, eksperymentowanie z alternatywnymi rozwiązaniami oraz utrzymanie stabilności głównej gałęzi. Dodatkowo korzystano z narzędzi umożliwiających poruszanie się po historii repozytorium, co pozwalało na szybkie przywrócenie ostatniej stabilnej wersji w przypadku wykrycia błędu.

4.4.2. Dokumentacja

Dokumentacja jest ważnym elementem każdego projektu programistycznego, stanowi podstawowe źródło informacji dotyczących implementacji, a także działania systemu. W opisywanym rozwiązaniu skorzystano z narzędzia do automatycznego generowania dokumentacji Sphinx [50], które umożliwia tworzenie jej na podstawie docstringów zawartych w kodzie źródłowym. Następnym krokiem jest komplikacja tak udokumentowanych plików, której wynikiem jest kod w formacie HTML. Powstałą dokumentację można przeglądać lokalnie jako stronę internetową lub wdrożyć na serwerze, by udostępnić ją online.

Funkcjonalność każdej z klas została przedstawiona w komentarzach docstring, które opisują działanie metod oraz parametry wejściowe i wyjściowe. Dodatkowo zadbano o to, by nazwy klas i funkcji jednoznacznie wskazywały ich przeznaczenie, ułatwiając innym osobom zrozumienie kodu i zwiększać jego utrzymywalność. Przykładowa dokumentacja została przedstawiona na rysunku 4.5.

4.4.3. Testy

Kolejnym kluczowym aspektem wytwarzania oprogramowania jest odpowiednie testowanie kodu źródłowego. Są różne rodzaje testów - jednostkowe, testujące pojedyncze

Detection Autonomous Driving

Navigation

Contents:

- lane_detection module
- mpc_controller module
- path_planning module
- visualization module
- carla_client module
- autonomous_driving_system module

Quick search

Go

Autonomous Driving's documentation!

Contents:

- [lane_detection module](#)
 - [LaneDetectionModule](#)
- [mpc_controller module](#)
 - [MPCController](#)
- [path_planning module](#)
 - [PathPlanningModule](#)
- [visualization module](#)
 - [VisualizationModule](#)
- [carla_client module](#)
 - [CarlaClient](#)
- [autonomous_driving_system module](#)
 - [AutonomousDrivingSystem](#)

Indices and tables

- [Index](#)
- [Module Index](#)
- [Search Page](#)

(a) Strona główna dokumentacji z nawigacją

(b) Fragment dokumentacji dotyczący planowania trasy

Rysunek 4.5. Dokumentacja w formacie HTML ze sphinx [50].

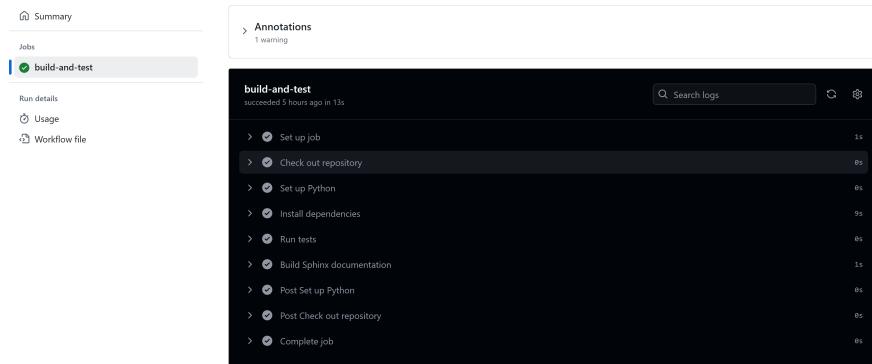
funkcje lub klasy, integracyjne sprawdzające część funkcjonalności systemu, na który składa się kilka komponentów, testy systemowe, weryfikujące działanie całego programu, kończąc na testach akceptacyjnych, zbliżonych do testów systemowych, tylko przeprowadzanych przez klienta. Najlepszym możliwym podejściem jest implementacja każdego z tych rodzajów i pełne pokrycie kodu testami; nie gwarantuje to wyeliminowania wszystkich błędów, ponieważ nigdy nie ma takiej gwarancji, jednak znaczco zmniejsza ryzyko ich wystąpienia.

W niniejszej pracy zostały zaimplementowane częściowe testy jednostkowe, które obejmowały zarówno standardowe przypadki, jak i sytuacje brzegowe, a także obsługę wyjątków. Do tworzenia testów wykorzystano bibliotekę unittest w języku Python. Testy zostały zorganizowane w jednym folderze o nazwie `tests/`, a ich nazwy wskazują, której z klas dotyczą.

Przygotowane testy stanowią jedynie podstawę do dalszego testowania. Aby osiągnąć pełniejszą weryfikację działania systemu, należałoby dążyć do pokrycia 100% kodu testami jednostkowymi. Ponadto, kolejnymi krokami byłoby opracowanie testów integracyjnych, w celu sprawdzenia współdziałania poszczególnych modułów oraz testów systemowych, umożliwiających weryfikację działania całego systemu.

Głównym źródłem informacji o nieprawidłowościach w implementacji był moduł wizualizacji, który pozwalał w szybki sposób ocenić, czy zadania detekcji, planowania i sterowania wykonują się poprawnie. Nie jest to metoda doskonała — w dłuższej perspektywie korzyści z możliwości natychmiastowego sprawdzenia działania systemu bez

4. Implementacja



Rysunek 4.6. Podgląd automatyzacji procesu CI/CD projektu w GitHub Actions.

dogłębnych testów mogą się zatrzeć, a proces wytwarzania oprogramowania wymaga staranniejszej weryfikacji. W przypadku omawianego, relatywnie niewielkiego projektu, takie podejście skróciło jednak czas tworzenia systemu.

4.4.4. Narzędzia CI/CD

Współcześnie nieodłącznym elementem procesu wytwarzania oprogramowania są narzędzia CI/CD (ang. *Continuous Integration / Continuous Delivery*). Celem jest automatyzacja kluczowych etapów tworzenia systemu, takich jak komplikacja, testowanie, generowanie dokumentacji czy wdrożenie kodu; zapewnia to szybką informację zwrotną na temat wprowadzanych zmian i usprawnia pracę. Na rynku dostępnych jest kilka rozwiązań, takich jak Jenkins [51], GitLab CI/CD [52], czy GitHub Actions [49]. W opisywanym projekcie zdecydowano się na GitHub Actions ze względu na bezpośrednią integrację z platformą GitHub oraz możliwość darmowego korzystania w przypadku publicznych repozytoriów.

Proces, którego przebieg pokazano na rysunku 4.6, skonfigurowano za pomocą pliku YAML umieszczonego w katalogu `.github/workflows/`. Główne etapy to:

1. **Pobranie kodu (checkout)** — pobranie aktualnego stanu repozytorium.
2. **Instalacja zależności** — przygotowanie środowiska Python z wymaganymi bibliotekami (NumPy, Sphinx, unittest itp.).
3. **Uruchomienie testów** — sprawdzenie poprawności kluczowych funkcji modułów.
4. **Kompilacja dokumentacji** — przy użyciu biblioteki sphinx wygenerowanie dokumentacji w formacie HTML; w razie błędów proces kończy się statusem niepowodzenia. Ten krok w praktyce służy wyłącznie do sprawdzenia czy dokumentacja skompiluje się poprawnie, jednak kolejnym etapem mogłoby być wdrożenie zaktualizowanej wersji dokumentacji na stronie.

Wynikiem przejścia wszystkich kroków (jeśli nie wystąpią błędy) jest komunikat o powodzeniu, widoczny zarówno w repozytorium GitHub, jak i na odnośniku do ostatniego wykonania procesu.

Dzięki tak skonfigurowanemu procesowi uzyskano spójną i zautomatyzowaną ścieżkę

wytwarzania oprogramowania, gdzie każda aktualizacja kodu w zdalnym repozytorium jest weryfikowana, a ewentualne błędy zostają wychwycone. Pozwala to na przyspieszenie całego cyklu rozwoju projektu i pomaga w jego utrzymaniu. W przyszłości rozwiązanie CI/CD można rozszerzyć o integrację z narzędziami do analizy jakości kodu typu linter w celu ujednolicenia stylu programowania w całym repozytorium. Oprócz tego można wprowadzić automatyczne wdrażanie na środowisko produkcyjne.

5. Eksperymenty i wyniki

W niniejszym rozdziale zostaną przedstawione rezultaty przeprowadzonych eksperymentów zarówno w odniesieniu do całego systemu zawierającego detekcję, planowanie i sterowanie, jak i w odniesieniu do zastosowanej sieci neuronowej LaneNet.



5.1. Model detekcji pasów ruchu

W trakcie trenowania i walidacji modelu sieci neuronowej stosowano szereg metryk, które pomogły w obserwowaniu postępu uczenia oraz doborze hiperparametrów. W celu wyjaśnienia sposobu obliczania należy zdefiniować podstawowe pojęcia oznaczające rodzaje możliwych predykcji:

- **TP (True Positive)** - poprawne wykrycie obecności pasa,
- **FP (False Positive)** - fałszywe wykrycie pasa, tam gdzie go nie ma,
- **FN (False Negative)** - brak wykrycia pasa, tam gdzie powinien być wykryty,
- **TN (True Negative)** - poprawne wykrycie braku pasa.

Dzięki nim można zdefiniować następujące miary jakości:



- **Precision (precyzja):**

$$\text{precision} = \frac{TP}{TP + FP},$$

opisuje, jaki odsetek elementów oznaczonych przez model jako pozytywne faktycznie jest pozytywny.

- **Recall (czułość):**

$$\text{recall} = \frac{TP}{TP + FN},$$

wskazuje, jaki procent elementów rzeczywiście pozytywnych został wychwycony przez model.

- **F1-score:**

$$F1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}},$$

to miara będąca średnią harmoniczną precyzji i czułości.

- **loss (strata)** stanowi wartość funkcji straty

Model LaneNet był trenowany na zbiorze danych TuSimple uprzednio podzielonym na zbiór treningowy oraz walidacyjny. Obserwacja wyników opisanych metryk pozwoliła na wybór odpowiednio wytrenowanego modelu. Wybrane wykresy zostały przedstawione na rysunkach 5.!

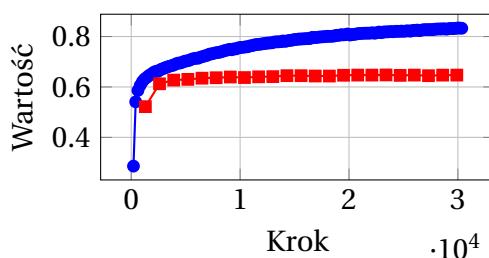


gdzie oś X oznaczona jest jako krok i oznacza numer pakietu danych wejściowych (ang. *batch*), który przy treningu sieci wynosił 16, a oś Y to wartość danej metryki.

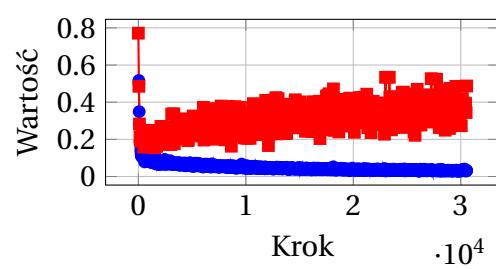
Przy wyborze odpowiedniego modelu starano się dobrać moment treningu, w którym wartości precyzji, F1, czułości na zbiorach treningowym i walidacyjnym byłyby maksymalne, a wartość funkcji straty minimalna. Niestety, optymalne wartości tych funkcji nie

znajdują się wszystkie w jednym miejscu, co jest widoczne na rysunkach. F1-score i precyzja rosną stabilnie zarówno na zbiorze testowym, jak i walidacyjnym. Wartość czułości na zbiorze walidacyjnym spada w miarę postępu treningu, co wskazuje na potencjalne problemy z generalizacją. Z kolei funkcja straty jest wyraźnie minimalizowana na zbiorze treningowym, a na zbiorze walidacyjnym jest niestabilna, jednak można zauważać trend wzrostowy, co może wskazywać na przeuczanie.

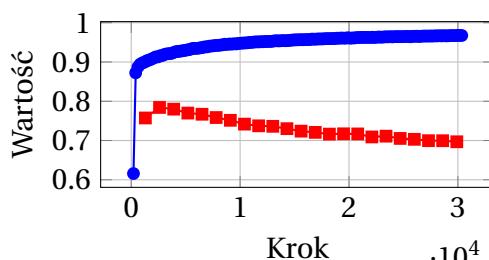
Ze względu na powyższe oraz kluczową potrzebę, aby model dobrze radził sobie z generalizacją, co jest istotne z uwagi na jego docelowe zastosowanie w środowisku innym niż dane treningowe, zdecydowano się wybrać moment treningu, w którym wartość czułości jeszcze nie zmalała znacząco, a miary F1-score i precyzji zaczynały osiągać stabilizację. Optymalnym punktem okazał się około 9000. krok, co przy rozmiarze batcha równym 16 i zbiorze treningowym liczącym 3600 przykładów odpowiada około 40. epoce.



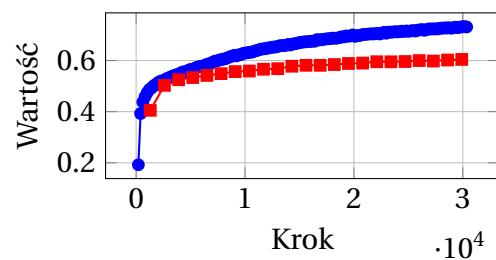
Rysunek 5.1. F1-score.



Rysunek 5.2. Loss.



Rysunek 5.3. Recall.



Rysunek 5.4. Precision.

—●— Zbiór testowy —■— Zbiór walidacyjny

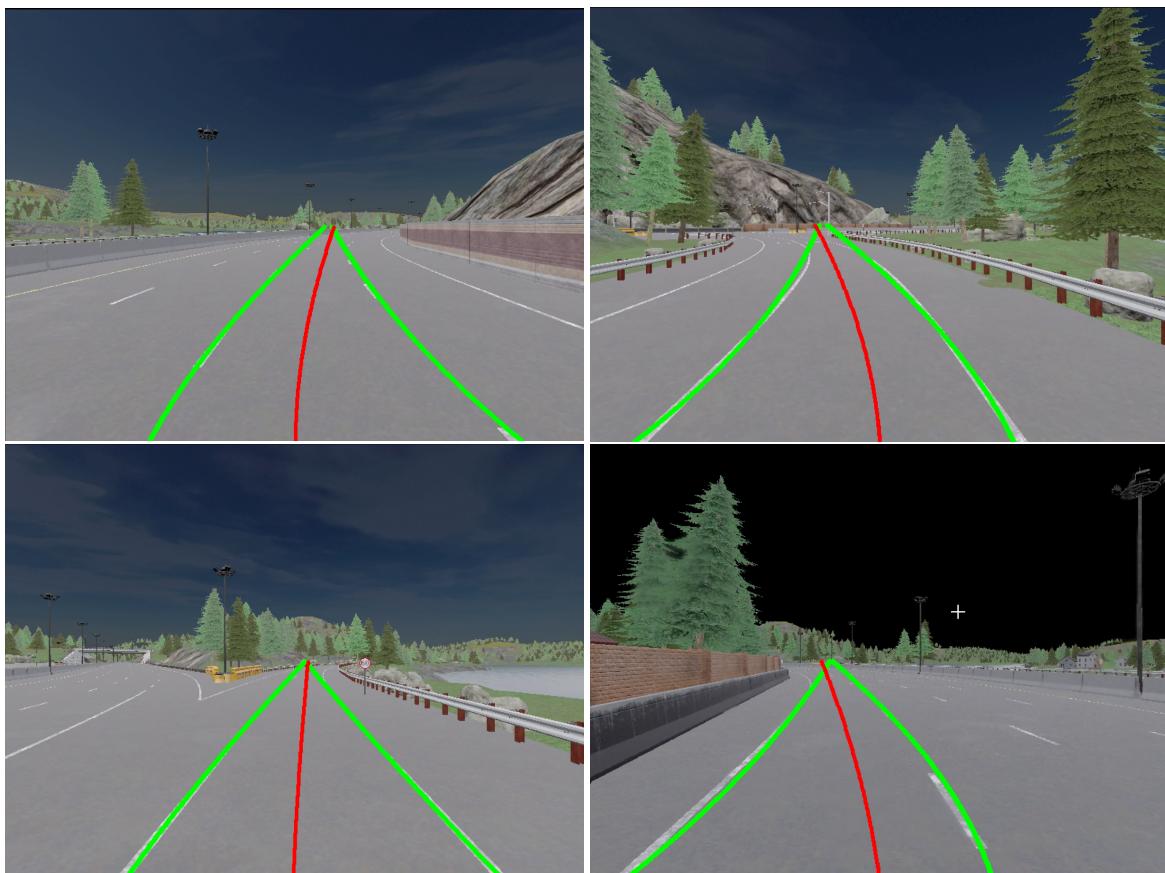
Rysunek 5.5. Porównanie różnych metryk (F1-score, Loss, Recall, Precision) uzyskanych podczas trenowania modelu.

5.2. System autonomicznej jazdy

Testowanie całego systemu odbywało się na podstawie danych pochodzących z modułu wizualizacji. Przygotowano scenariusze testowe obejmujące proste odcinki drogi oraz zakręty w obie strony z różnymi kątami, sprawdzano zachowanie z ciągłymi i przerwanymi pasami ruchu. Oprócz tego testy były przeprowadzane w różnych warunkach

widoczności, aby sprawdzić odporność rozwiązania. Przypadek uznawano za sukces w momencie, gdy pojazd, nie opuszczając danego pasa ruchu, przejechał testowy odcinek drogi i prowadził stabilne sterowanie, czyli nie doprowadzał do oscylacji w obrębie pasa.

Ostatecznie większość scenariuszy testowych zakończyła się powodzeniem: system poradził sobie dobrze z prostymi odcinkami drogi oraz lekkimi i średnimi zakrętami, a rodzaj linii (ciągłe lub przerywane) nie miał istotnego wpływu na skuteczność detekcji. Problemem okazały się natomiast ostre zakręty o kącie $\geq 80^\circ$, przy których pojazd chwilami zjeżdżał poza granicę pasa ruchu. Źródłem takiego zachowania był opisany wcześniej problem generalizacji sieci neuronowej. Mimo że została ona dodatkowo dotrenowana na zbiorze danych z Carr^F nie dysponowano wystarczająco zróżnicowaną bazą obrazów — skrypt generujący zdjęcia tworzył bowiem wiele kopii tych samych ujęć, przez co ostatecznie nowych danych nie było wiele i ograniczyło to możliwości do skutecznego sterowania w bardziej wymagających warunkach. 



Rysunek 5.6. Przykładowe widoki z testów systemowych.



6. Podsumowanie

Celem niniejszej pracy była implementacja kompleksowego systemu sterowania autonomicznym pojazdem na podstawie predykcji pochodzących z algorytmu detekcji pasa ruchu. Realizacja tego celu wymagała przeprowadzenia szeregu zadań, takich jak zaprojektowanie architektury systemu, wybór i integracja algorytmu detekcji pasa ruchu, stworzenie rozwiązania odpowiedzialnego za planowanie trasy oraz implementacja sterowania.

Prace rozpoczęto od przeglądu istniejących rozwiązań pełniących rolę autopilotów w autonomicznych pojazdach. Wspólnym mianownikiem większości z nich był podział zadań na budowanie obrazu przestrzeni przy użyciu sensorów, planowanie trasy i sterowanie. Dzięki temu przy tworzeniu systemu zdecydowano się na podejście modułowe, zawierające wspomniane komponenty, które ułatwiło przyszły rozwój oprogramowania oraz umożliwiło osobną weryfikację poprawności działania każdego z modułów.

Wybór algorytmu detekcji pasa ruchu również opierał się na dogłębnej analizie różnych metod, zaczynając od tradycyjnych metod opierających się na klasycznym przetwarzaniu obrazu, po sieci neuronowe, w których zdefiniowane było zadanie klasyfikacji, detekcji lub segmentacji. Analiza doprowadziła do decyzji o wykorzystaniu modelu LaneNet, który wydawał się najbardziej efektywny w kontekście realizowanego zadania.

Dla modułu sterowania zastosowano algorytm MPC oparty na uproszczonym modelu kinematycznym pojazdu. Dzięki temu podejściu udało się uzyskać wysoką jakość sterowania, zdolną utrzymywać pojazd na wyznaczonej trajektorii w obliczu wymagających warunków drogowych.

System został zaimplementowany zgodnie ze sztuką wytwarzania oprogramowania, co objawia się m.in. powstaniem szczegółowej dokumentacji opisującej kluczowe elementy systemu i ich działanie. Sam kod źródłowy jest dobrze udokumentowany, co ułatwia zrozumienie oraz przyszły rozwój projektu. Do zapewnienia jakości oprogramowania stworzono również testy jednostkowe sprawdzające podstawowe funkcjonalności modułów. Proces testowania oraz komplikacji dokumentacji został zautomatyzowany przy użyciu narzędzia CI/CD GitHub Actions, co przyspieszyło cały cykl wytwarzania i pomogło w eliminacji błędów.

Algorytm detekcji pasa ruchu LaneNet został nauczony i przetestowany na zbiorze danych TuSimple, gdzie osiągnął satysfakcyjujące wyniki. Całość systemu sprawdzono w różnych scenariuszach testowych, uwzględniających różne typy pasów oraz zakręty. Testy przebiegły pomyślnie, a algorytm sprawdził się w większości przypadków. System posiada wiele kierunków rozwoju, w tym m.in. rozszerzenie detekcji obiektów o znaki drogowe, inne pojazdy, sygnały świetlne i inne elementy otoczenia, a także integracja tych danych z modułem planowania. Dodatkowo możliwe jest wdrożenie modułu nawigacji GPS, który umożliwia poruszanie się pojazdu do określonego miejsca na mapie, uwzględniając zarówno dane z detekcji otoczenia, jak i informacje GPS.

Bibliografia

- [1] D. Parekh, N. Poddar, A. Rajpurkar i in., "A Review on Autonomous Vehicles: Progress, Methods and Challenges", *Electronics*, t. 11, nr. 14, 2022, ISSN: 2079-9292. DOI: 10.3390/electronics11142162. adr.: <https://www.mdpi.com/2079-9292/11/14/2162>.
- [2] J. Nidamanuri, C. Nibhanupudi, R. Assfalg i H. Venkataraman, "A Progressive Review: Emerging Technologies for ADAS Driven Solutions", *IEEE Transactions on Intelligent Vehicles*, t. 7, nr. 2, s. 326–341, 2022. DOI: 10.1109/TIV.2021.3122898.
- [3] P. E. i Rada Unii Europejskiej, *Rozporządzenie Parlamentu Europejskiego i Rady (UE) 2019/2144 z dnia 27 listopada 2019 r. w sprawie wymogów dotyczących homologacji typu pojazdów silnikowych i ich przyczep oraz układów, komponentów i oddzielnych zespołów technicznych przeznaczonych do tych pojazdów, w odniesieniu do ich ogólnego bezpieczeństwa oraz ochrony osób znajdujących się w pojeździe i niechronionych uczestników ruchu drogowego*, Dostęp: 14 grudnia 2024, 2019. adr.: <https://eur-lex.europa.eu/legal-content/PL/TXT/?uri=CELEX%3A32019R2144>.
- [4] GoodCar Team, *Lane Keep Assist System – Car Safety*, <https://goodcar.com/car-safety/lane-keep-assist-system>, 2024.
- [5] CARLA Team, *CARLA: Open-source simulator for autonomous driving research*, <https://carla.org>, 2017.
- [6] A. Vaswani, N. Shazeer, N. Parmar i in., *Attention Is All You Need*, arXiv preprint, arXiv:1706.03762, 2017. adr.: <https://arxiv.org/abs/1706.03762>.
- [7] T. Autonomous, *Breakdown: How Tesla will transition from Modular to End-To-End Deep Learning*, 2023. adr.: <https://www.thinkautonomous.ai/blog/tesla-end-to-end-deep-learning/>.
- [8] I. Tesla, *Tesla Privacy Notice*, Informacje dotyczące zbierania i przetwarzania danych przez pojazdy Tesli, 2023. adr.: https://www.tesla.com/pl_pl/legal/privacy?utm_source=chatgpt.com (term. wiz. 27.12.2023).
- [9] I. Tesla, *Tesla AI Day Full Self-Driving Presentation*, <https://www.youtube.com/watch?v=j0z4FweCy4M&t=3082s>, Omówienie autopilota Tesli od 48:00 do 1:13:00., 2021.
- [10] T. AI, *Tesla's Self-Driving Algorithm Explained*, <https://towardsai.net/p/1/teslas-self-driving-algorithm-explained>, 2024.
- [11] AutoExpert.pl, *Grupa Volkswagen rozszerza współpracę z Mobileye*, <https://autoexpert.pl/artykuly/grupa-volkswagen-rozszerza-wspolprace-z-mobileye>, 2024.
- [12] Mobileye, *Mobileye EyeQ Technology Overview*, <https://www.mobileye.com/technology/eyeq-chip/>, 2024.
- [13] Mobileye, *EyeQ Kit: Building Unique Applications on Mobileye's Platform*, <https://www.mobileye.com/blog/eyeq-kit-sdk/>, 2024.
- [14] Wikipedia, *Waymo – Wikipedia, wolna encyklopedia*, <https://pl.wikipedia.org/wiki/Waymo>, 2024.

6. Bibliografia

- [15] Waymo, *Simulation City: Introducing Waymo's Most Advanced Simulation System*, <https://waymo.com/blog/2021/07/simulation-city/>, 2021.
- [16] Restackio, *Waymo Lidar Technology in AI*, <https://www.restack.io/p/waymo-lidar-answer-ai-autonomous-vehicles-cat-ai>, 2024.
- [17] Waymo, *Waymo Open Dataset*, <https://github.com/waymo-research/waymo-open-dataset>, 2024.
- [18] I. Biznes, *Autonomiczne taksówki Waymo już wkrótce pojawią się na ulicach Miami*, <https://itbiznes.pl/transport/waymo-taksowki-autonomiczne-miami-moove/>, 2024.
- [19] Waymo, *Introducing Waymo's Research on an End-to-End Multimodal Model for Autonomous Driving*, <https://waymo.com/blog/2024/10/introducing-emma/>, 2024.
- [20] Waymo, *Introducing the 5th-generation Waymo Driver: Informed by experience, designed for scale, engineered to tackle more environments*, <https://waymo.com/blog/2020/03/introducing-5th-generation-waymo-driver/>, 2020.
- [21] A. Foundation, *Autoware Overview*, <https://autoware.org/autoware-overview/>, 2024.
- [22] A. Foundation, *Benchmark Tool Nodes Documentation*, <https://autowarefoundation.gitlab.io/autoware.auto/AutowareAuto/benchmark-tool-nodes-readme.html>, Informacje o wykorzystaniu KITTI w Autoware., 2024.
- [23] Klemenko, *KITTI Dataset on Kaggle*, <https://www.kaggle.com/datasets/klemenko/kitti-dataset>, Zbiór danych KITTI, dostępny na platformie Kaggle., 2024.
- [24] A. Foundation, *Autoware Architecture Design*, <https://autowarefoundation.github.io/autoware-documentation/main/design/autoware-architecture/>, 2024.
- [25] Z. Zhang i X. Ma, “Lane Recognition Algorithm Using the Hough Transform Based on Complicated Conditions”, *Journal of Computer and Communications*, t. 7, nr. 11, s. 65–75, 2019. DOI: 10.4236/jcc.2019.711005. adr.: <https://www.scirp.org/journal/paperinformation?paperid=96456>.
- [26] A. Gurghian, T. Koduri, S. V. Bailur, K. J. Carey i V. N. Murali, “DeepLanes: End-To-End Lane Position Estimation Using Deep Neural Networks”, w *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2016, s. 38–45. DOI: 10.1109/CVPRW.2016.12.
- [27] B. Huval, T. Wang, S. Tandon i in., “An Empirical Evaluation of Deep Learning on Highway Driving”, *CoRR*, t. abs/1504.01716, 2015. arXiv: 1504.01716. adr.: <http://arxiv.org/abs/1504.01716>.
- [28] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus i Y. LeCun, *OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks*, 2014. arXiv: 1312.6229 [cs.CV]. adr.: <https://arxiv.org/abs/1312.6229>.

- [29] S. Lee, J. Kim, J. S. Yoon i in., *VPGNet: Vanishing Point Guided Network for Lane and Road Marking Detection and Recognition*, 2017. arXiv: 1710.06288 [cs.CV]. adr.: <https://arxiv.org/abs/1710.06288>.
- [30] Y. Huang, S. Chen, Y. Chen, Z. Jian i N. Zheng, "Spatial-Temporal Based Lane Detection Using Deep Learning", w *Artificial Intelligence Applications and Innovations*, L. Iliadis, I. Maglogiannis i V. Plagianakos, red., Cham: Springer International Publishing, 2018, s. 143–154, ISBN: 978-3-319-92007-8.
- [31] V. Badrinarayanan, A. Kendall i R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, t. 39, nr. 12, s. 2481–2495, 2017.
- [32] B. Emek Soylu, M. S. Guzel, G. E. Bostancı, F. Ekinci, T. Asuroglu i K. Acici, "Deep-Learning-Based Approaches for Semantic Segmentation of Natural Scene Images: A Review", *Electronics*, t. 12, nr. 12, 2023, ISSN: 2079-9292. DOI: 10.3390/electronics12122730. adr.: <https://www.mdpi.com/2079-9292/12/12/2730>.
- [33] Q. Zou, H. Jiang, Q. Dai, Y. Yue, L. Chen i Q. Wang, "Robust Lane Detection From Continuous Driving Scenes Using Deep Neural Networks", *IEEE Transactions on Vehicular Technology*, t. 69, nr. 1, s. 41–54, sty. 2020, ISSN: 1939-9359. DOI: 10.1109/tvt.2019.2949603. adr.: <http://dx.doi.org/10.1109/TVT.2019.2949603>.
- [34] O. Ronneberger, P. Fischer i T. Brox, "U-net: Convolutional networks for biomedical image segmentation", w *International Conference on Medical Image Computing and Computer-Assisted Intervention*, Springer, 2015, s. 234–241.
- [35] TuSimple, *TuSimple Lane Detection Benchmark*, 2017. adr.: <https://www.kaggle.com/datasets/manideep1108/tusimple>.
- [36] M. Ghafoorian, C. Nugteren, N. Baka, O. Booij i M. Hofmann, *EL-GAN: Embedding Loss Driven Generative Adversarial Networks for Lane Detection*, 2018. arXiv: 1806.05525 [cs.CV]. adr.: <https://arxiv.org/abs/1806.05525>.
- [37] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza i in., *Generative Adversarial Networks*, 2014. arXiv: 1406.2661 [stat.ML]. adr.: <https://arxiv.org/abs/1406.2661>.
- [38] D. Neven, B. D. Brabandere, S. Georgoulis, M. Proesmans i L. V. Gool, *Towards End-to-End Lane Detection: an Instance Segmentation Approach*, 2018. arXiv: 1802.05591 [cs.CV]. adr.: <https://arxiv.org/abs/1802.05591>.
- [39] A. Paszke, A. Chaurasia, S. Kim i E. Culurciello, *ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation*, 2016. arXiv: 1606.02147 [cs.CV]. adr.: <https://arxiv.org/abs/1606.02147>.
- [40] C. R. Harris, K. J. Millman, S. J. van der Walt i in., *Array programming with NumPy*, 2020. DOI: 10.1038/s41586-020-2649-2. adr.: <https://numpy.org/>.
- [41] *OpenCV Library*, Open Source Computer Vision Library, 2023. adr.: <https://opencv.org/>.
- [42] Epic Games, *Unreal Engine*, 2025. adr.: <https://www.unrealengine.com>.

6. Bibliografia

- [43] S. Qian, *Lane Detection: An Instance Segmentation Approach*, https://github.com/ShenhanQian/Lane_Detection-An_Instance_Segmentation_Approach, Accessed: January 2, 2025, 2018.
- [44] A. Paszke, S. Gross, F. Massa i in., *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox i R. Garnett, red., 2019. adr.: <https://pytorch.org/>.
- [45] J. Andersson, K. Gillis, G. Horn, J. B. Rawlings i M. Diehl, *CasADi – A symbolic framework for dynamic optimization*, <https://web.casadi.org/>.
- [46] Glutamat42, *Carla Lane Detection Dataset Generation*, <https://github.com/Glutamat42/Carla-Lane-Detection-Dataset-Generation>, 2021.
- [47] *Git - Distributed Version Control System*, <https://git-scm.com/>, 2025.
- [48] *OpenHub - Repository Comparison*, <https://openhub.net/repositories/compare>.
- [49] *GitHub - Where the World Builds Software*, <https://github.com/>.
- [50] S. D. Team, *Sphinx: Python Documentation Generator*, 2025. adr.: <https://www.sphinx-doc.org/>.
- [51] J. Community, *Jenkins: The Leading Open-Source Automation Server*, 2025. adr.: <https://www.jenkins.io/>.
- [52] G. Inc., *GitLab CI/CD: Continuous Integration and Delivery*, 2025. adr.: <https://docs.gitlab.com/ee/ci/>.

Wykaz symboli i skrótów

PW – Politechnika Warszawska
CNN – ang. *Convolutional Neural Network*
ADAS – ang. *Advanced Driver Assistance System*
EL-GAN – Embeding Loss - Generative Adversarial Neural Network
MPC – ang. *Model Predictive Control*
LSTM – ang. *Long short-term memory*
BEV – ang. *Bird's-Eye View*
CI/CD – ang. Continous Integration / Continous Delivery
HTML – ang. HyperText Markup Language

Spis rysunków

1.1	Lane Keep Assist System – system wspomagania utrzymania pasa ruchu, przykład ADAS. Źródło: [4]	10
2.1	Architektura percepcji autopilota Tesli. Źródło:[10]	13
2.2	Sensory autonomicznej taksówki firmy Waymo. Źródło:[20]	14
2.3	Architektura systemu Autoware.Źródło:[24]	15
2.4	Ułożenie kamer dla DeepLane. Źródło:[26]	19
2.5	Przebieg algorytmu STLNet. Źródło:[30]	21
2.6	Architektura sieci typu enkoder-dekoder używana do segmentacji na przykładzie SegNet [31]. Źródło:[32]	21
2.7	Architektura EL-GAN, ilustrująca trening generatora i dyskryminatora. Źródło:[36]	23
2.8	Schemat architektury i działania sieci LaneNet. Źródło:[38]	24
3.1	Schemat architektury zaprojektowanego systemu. Każda komunikacja między modułami odbywa się za pośrednictwem modułu głównego. Zdecydowano się jednak nie przedstawiać tego na rysunku, by zachować czytelność ilustracji.	26
3.2	Przykłady obrazów ze zbioru danych TuSimple [35].	28
4.1	Wyniki działania algorytmu LaneNet na podstawie obrazu wejściowego.	32
4.2	Segmentacja semantyczna obrazu z symulatora po wytrenowaniu na zbiorze TuSimple.	33
4.3	Ilustracja z oznaczonymi pasami ruchu i zaplanowaną trasą.	34
4.4	Ilustracja układów współrzędnych obrazu (lewy górny róg) i pojazdu (po środku na dole).	35
4.5	Dokumentacja w formacie HTML ze sphinx [50].	37
4.6	Podgląd automatyzacji procesu CI/CD projektu w GitHub Actions.	38
5.1	F1-score.	41
5.2	Loss.	41

5.3 Recall.	41
5.4 Precision.	41
5.5 Porównanie różnych metryk (F1-score, Loss, Recall, Precision) uzyskanych podczas trenowania modelu.	41
5.6 Przykładowe widoki z testów systemowych.	42

Spis tabel

2.1 Porównanie rozwiązań autonomicznej jazdy	16
2.2 Porównanie wybranych algorytmów detekcji pasa ruchu.	25