Politechnika Śląska
Wydział Automatyki, Elektroniki i Informatyki
Kierunek Informatyka

Rozprawa doktorska
Doktorat wdrożeniowy

Wykorzystanie symulacji komputerowej w systemach autonomii pojazdów lądowych

Autor: mgr inż. Łukasz Sobczak
Promotor: dr hab. inż. Adam Domański, Prof. PŚ

Gliwice, grudzień 2022

Streszczenie

Celem pracy było przygotowanie szeregu metod oraz środowiska symulacyjnego, które pozwoliłoby na ewaluację wybranych algorytmów autonomii jazdy przeznaczonych dla Bezzałogowych Platform Lądowych (BPL) oraz innych pojazdów robotycznych. Opracowane narzędzia i rozwiązania dotyczą zagadnień z zakresu implementacji systemów symulacji zarówno pojazdów autonomicznych, jak i sensorów stosowanych w systemach autonomii jazdy oraz pozwoliły na stworzenie kompleksowego symulatora pojazdów autonomicznych WST (Wirtualny System Testujący).
Realizacja pracy wymagała od autora szerokiego przeglądu literatury dotyczącej algorytmów wykorzystywanych przez wiele rodzajów samo-kierujących pojazdów. Lokalizacja i mapowanie otoczenia, percepcja otoczenia czy wyznaczanie trasy przejazdu i nawigacja do celu to główne zagadnienia autonomii jazdy, których opracowanie wymaga dostępu do platformy jezdnej, wielu rodzajów czujników i czasochłonnych testów wykonywanych w rzeczywistych warunkach. Autor w ramach rozprawy doktorskiej postawił tezę, że możliwe jest przygotowanie systemu symulacji pojazdu i jego czujników w sposób pozwalający dokonać weryfikacji i ewaluacji algorytmów autonomii bez dostępu do fizycznego pojazdu. Tym samym opracowany symulator może posłużyć do przygotowywania, testowania i oceny metod automatyzacji jazdy oraz zapewni środowisko symulacyjne dla pojazdów typu BPL o niemal dowolnych rozmiarach, przeznaczeniu czy konfiguracji napędu. System oferuje również dużą wydajność działania zapewniając wysoką jakość generowanych danych sensorycznych przy zachowaniu działania w czasie rzeczywistym. Pozwala ponadto na wymianę danych pomiędzy rzeczywistymi, a symulowanymi czujnikami dzięki zastosowaniu wspólnej reprezentacji danych i ich formatowi zgodnymi z rzeczywistymi urządzeniami. Dodatkowo przygotowany symulator umożliwia zebranie danych pozwalających na ewaluację zasadności użycia poszczególnych sensorów i ich dobór wraz z fizyczną lokalizacją na danym pojeździe. Opracowane rozwiązanie umożliwia bezpośrednią komunikację między systemem robota ROS (Robot Operating System), a środowiskiem symulacji z wysoką przepustowością danych co potwierdziły wykonane badania i porównanie z istniejącym rozwiązaniem.
Dużą część badań autor poświęcił na wykorzystanie opracowanego symulatora do ewaluacji działania algorytmu lokalizacji i mapowania SLAM, stanowiącego kluczową rolę niemal każdego systemu autonomicznej jazdy. Aby poprawnie ocenić i porównać skuteczność wyznaczania pozycji i jakość generowanej mapy otoczenia z wykorzystaniem danych symulacyjnych niezbędna jest weryfikacja dokładności syntetycznych danych względem danych, dostarczanych przez fizyczne urządzenie. Przeprowadzone badania wykazały, że generowane z użyciem symulatora dane w znacznym stopniu odpowiadają rzeczywistości, przede wszystkim dzięki symulacji charakterystycznych błędów i szumów danych sensorów. Następnie porównano wartości błędu lokalizacji wybranego algorytmu SLAM korzystającego z rzeczywistych i syntetycznych danych. Przygotowany tor testowy oraz jego odpowiednik odwzorowany w środowisku symulacyjnym umożliwiły

zbadanie wpływu zastosowania danych generowanych w symulatorze na jakość lokalizacji pojazdu. Uzyskane wyniki potwierdziły, że wartość błędu pozycjonowania pojazdu przy wykorzystaniu danych symulacyjnych jest zbliżona do tej, uzyskanej w warunkach rzeczywistych potwierdzając tym samym zasadność stosowania symulatora.
Wykonane badania i wykazanie skuteczności symulacji w kontekście lokalizacji i mapowania, umożliwiły ocenę możliwości zastosowania symulatora w procesie doboru czujników LiDAR, IMU i odometrii dla wybranego algorytmu SLAM. Wykorzystując opracowane środowisko zaproponowano procedurę testową pozwalającą na znalezienie odpowiedniej konfiguracji sensorów, przedstawiono również wady i zalety takiego rozwiązania.
Ponadto przygotowany symulator i jego możliwości zostały przeanalizowane pod kątem zastosowania w procesie badań i rozwoju pozostałych algorytmów wykorzystywanych podczas autonomicznej jazdy. Omówiono kwestię zastosowania zaprojektowanego środowiska do testów metod wstępnego przetwarzania danych sensorycznych. Przedstawiono także opracowany mechanizm tworzenia zbioru danych, które mogą posłużyć jako dane testowe i treningowe dla nauki modeli głębokich sieci neuronowych służących np. do segmentacji semantycznej obrazu pochodzącego z kamer umieszczonych na autonomicznym pojeździe. Zaprezentowano również praktyczne zastosowanie symulatora do opracowania i testowania algorytmów autonomicznej nawigacji, w tym tworzenia siatki zajętości przeszkód na podstawie danych lidarowych, wyznaczania ścieżki przejazdu oraz generowania komend sterowania pozwalających na podążania zgodnie z wyznaczonym planem. Ostatecznie przedstawiono również architekturę umożliwiającą testy systemu autonomii, w którym dane z rzeczywistych sensorów są zastąpione danymi symulacyjnymi, a funkcjonalności pozostają bez zmian z perspektywy operatora systemu. Przedstawione rezultaty prac zakończyły się wdrożeniem środowiska symulacyjnego do prac rozwojowych nad pojazdami wyposażonymi w system autonomicznej jazdy, którego komponenty zostały zaprojektowane, ewaluowane i dostosowane w zaprezentowanym symulatorze WST. Pozwoliło to ostatecznie na przygotowanie kilku Bezzałogowych Platform Lądowych spełniających założenia w kontekście autonomicznego poruszania się. Udowadnia to, że uzyskane w pracy wyniki przyczyniły się do opracowania symulatora, który stanowi istotną pomoc w procesie opracowywania systemów autonomii dla BPL.
Uzyskane przez autora wyniki badań i prac prowadzonych w związku z rozprawą doktorską umożliwiły poszerzenie stanu wiedzy poza tą dostępną w literaturze i zaprezentowały w kilku aspektach nowatorskie podejście do zastosowania metod symulacyjnych na potrzeby prac nad pojazdami autonomicznymi, które stanowią istotną gałąź w nowoczesnej informatyce. Ponadto otrzymane rezultaty oraz zaproponowane metody i potencjalne możliwości zastosowania w praktyce zostały opublikowane przez autora w czasopismach naukowych.

Abstract

The aim of the thesis was to prepare a series of methods and a simulation environment that would allow the evaluation of selected autonomous driving algorithms designed for Unmanned Land Platforms (BPLs) and other robotic vehicles. The developed tools and solutions refer to issues in the implementation of simulation systems for both autonomous vehicles and sensors used in autonomous driving systems, and made it possible to create a comprehensive simulator for autonomous vehicles WST (Virtual Testing System).

Realization of the thesis required the author to conduct an extensive literature review of the algorithms used by many types of self-driving vehicles. Localization and mapping of the environment, perception of the surroundings or routing and navigation to a destination are the main issues of autonomous driving, which require access to a driving platform, many types of sensors and time-consuming tests performed in real conditions. In contrast, the author's thesis presented as part of his PhD thesis is that it is possible to prepare a simulation system for a vehicle and its sensors in a way that allows verification and evaluation of autonomy algorithms without access to a physical vehicle. Thus, the developed simulator can be used to prepare, test and evaluate driving automation methods, and provides a simulation environment for BPL vehicles of almost any size, purpose or drive configuration. The system also offers high performance by providing high quality of generated sensors data while maintaining real-time operation. Moreover, it allows data exchange between real and simulated sensors thanks to the use of a common data representation and its format compatible with real devices. In addition, the prepared simulator makes it possible to collect data allowing the evaluation of the reasonableness of the use of specific sensors and their selection with their physical location on a given vehicle. The developed solution allows direct communication between the robot operating system (ROS) and the simulation environment with high data bandwidth, which was confirmed by the performed tests and comparison with the existing solution.

The author spent a large part of the research using the developed simulator to evaluate the performance of the SLAM localization and mapping algorithm, which is a crucial component of almost every autonomous driving system. In order to properly evaluate and compare the efficiency of position determination and the quality of the generated map of the environment using simulation data, it is necessary to verify the accuracy of the synthetic data against those provided by the physical device. The conducted tests showed that the data generated using the simulator significantly corresponds to reality, especially thanks to the simulation of characteristic errors and sensor data noise. The localization error values of the selected SLAM algorithm using real and synthetic data have been compared next. The prepared test track and its equivalent modeled in the simulation environment made it possible to study the impact of using data generated in the simulator on the quality of vehicle localization. The obtained results confirmed

that the value of the vehicle positioning error using simulation data is similar to that in real conditions thus confirming the validity of using the simulator.
Thanks to the performed research and demonstration of the effectiveness of the simulation in the context of localization and mapping, the author also decided to evaluate the possibility of using the simulator in the process of selecting LiDAR, IMU and odometry sensors for the selected SLAM algorithm. Using the developed environment, a test procedure was proposed to find a suitable sensor configuration, and the advantages and disadvantages of such a solution were also presented.
In addition, the prepared simulator and its capabilities were analyzed for application in the process of research and development of other algorithms used during autonomous driving. Thus, the issue of using the designed environment to test methods for preprocessing sensory data was discussed. Also presented is the developed mechanism for creating a dataset that can provide test and training data for learning deep neural network models used, for example, for semantic segmentation of images coming from cameras placed on an autonomous vehicle. A practical application of the simulator for developing and testing algorithms for autonomous navigation, including creating an obstacle occupancy grid based on lidar data, determining a driving path, and generating control commands to follow a defined plan, was also presented. Finally, an architecture was also presented to enable testing of the autonomy system, in which real sensor data is replaced by simulation data, while functionalities remain unchanged from the system operator's perspective. The presented results of the work concluded with the implementation of a simulation environment for the development of vehicles equipped with an autonomous driving system, the components of which were designed, evaluated and adapted in the presented WST simulator. It allowed to prepare several Unmanned Land Platforms that meet the objectives in the context of autonomous driving.
The results obtained by the author from the research and work performed in connection with the thesis made it possible to expand the state of knowledge beyond that available in the literature and presented, in several aspects, a novel approach to the application of simulation methods for the work on autonomous vehicles, which is an important discipline in modern computer science. In addition, the results obtained, as well as the proposed methods and potential applications in practice, have been published by the author in a number of articles in international journals.

Spis treści

Spis rysunków

Spis tabel

Rozdział 1

Wstęp

Wizja zastosowania systemów autonomii jazdy w codziennym świecie coraz śmielej pojawia się w świadomości nie tylko inżynierów i badaczy tych zagadnień, ale również wśród entuzjastów nowych technologii, a nawet zwykłych obserwatorów. Autonomiczne samochody poruszające się po publicznych drogach, mobilne roboty dostarczające przesyłki czy bezzałogowe pojazdy wojskowe realizujące misje obserwacyjne bądź transportowe to tylko niektóre z rozwiązań szeroko opisywanych w literaturze i mediach. Są to także projekty, które angażują badaczy z wielu dziedzin informatyki czy robotyki przyczyniając się tym samym do rozwoju m.in. głębokiego uczenia maszynowego wykorzystywanego do percepcji otoczenia, układów sterowania i regulatorów umożliwiających bezpieczne przemieszczanie się pojazdów, jak również zagadnień związanych z lokalizacją w przestrzeni czy problemem SLAM (z ang. *Simultaneous Localization And Mapping*).

Bezzałogowe platformy lądowe (BPL) to grupa pojazdów zdalnie sterowanych lub autonomicznych o wielu klasach masy i wielkości, a także różnych układów sterowania kołowego lub gąsienicowego. Można wyróżnić wiele rodzajów BPL od miniplatform do 30 kg, aż do platform bardzo ciężkich o masie powyżej 10 ton [1]. Zazwyczaj pojazdy te przeznaczone są do zastosowań militarnych, realizując misje obserwacyjne na danym obszarze lub transportowe, polegające na przewozie ładunku z punktu A do punktu B. W większości przypadków operują na terenie otwartym, ale miniplatformy pozwalają także na eksplorację pomieszczeń zamknię-

tych. Często wyposażone są one również w manipulator zapewniający operatorowi wysoką precyzję działania podczas zdalnego sterowania. Obecnie wielu producentów BPL opracowuje systemy autonomicznej jazdy dla swoich produktów, aby umożliwić wykonywanie misji bez udziału operatora lub przy jego minimalnym zaangażowaniu [2, 3].

W zależności od pożądanego poziomu autonomii zmienia się złożoność całego systemu. Zapewnienie kolejnych funkcjonalności wymaga stosowania coraz bardziej złożonych algorytmów i rozwiązań sprzętowych. Pociąga to za sobą konieczność przeprowadzania długotrwałych i skomplikowanych testów, aby zapewnić odpowiednio wysoki poziom bezpieczeństwa i zweryfikować poprawność działania systemu w różnych warunkach i okolicznościach. W zależności od pojazdu, dla którego przygotowywany jest system autonomicznej jazdy, testy takie wymagają często powtarzalności czy konkretnych warunków, co np. w przypadku samochodów może wymagać określonej sytuacji na drodze, co w praktyce okazuje się nie do powtórzenia w realnym świecie. Ponadto, aby zweryfikować poprawność autonomicznych samochodów niezbędne jest często pokonywanie milionów kilometrów w krótkim czasie, co jest sporym wyzwaniem logistycznym i kosztowym, natomiast w przypadku specjalistycznych pojazdów bezzałogowych czy robotów założenia mogą dotyczyć operowania w specyficznych warunkach trudnych do odwzorowania w rzeczywistości. Właśnie w tym celu powszechnie wykorzystywane są symulacje komputerowe. Zastosowanie wirtualnego środowiska pozwala na właściwie nieograniczone wykonywanie testów w powtarzalnych warunkach, symulowanie specyficznych sytuacji drogowych, czy odwzorowanie z wysoką dokładnością warunków środowiskowych, w których dany pojazd ma operować w sposób autonomiczny. W zależności od rodzaju platformy, której sterowanie ma zostać zautomatyzowane, stosowane są różne typy symulacji, które kładą nacisk na inne aspekty autonomicznej jazdy umożliwiając tym samym testy algorytmów realizujących różne zadania. Praca [4] opisuje zastosowanie symulacji do nauki głębokiej sieci neuronowej przeznaczonej do sterowania samochodem po drogach publicznych bezpiecznie i zgodnie z przepisami. Dzięki użyciu symulatora możliwa była walidacja działania wytrenowanego modelu w wielu złożonych sytuacjach drogowych, których wystąpienie wymaga w rzeczywi-

stości zbiegu określonych okoliczności występujących niezwykle rzadko w specyficznych warunkach. Praca [5] skupia się na symulacji danych dostarczanych przez jeden określony typ sensora co sprawia, że utworzone w ten sposób syntetyczne dane mogą posłużyć do treningu i testów algorytmów detekcji i klasyfikacji obiektów korzystających z głębokiego uczenia maszynowego. Autorzy [6] prezentują kompleksowy symulator przeznaczony do badań systemów autonomicznej jazdy w środowisku miejskim. Oferuje on symulację ruchu drogowego oraz generowanie danych z wielu rodzajów sensorów. Ponadto istnieje również szereg innych rozwiązań skupionych wokół wybranych aspektów tematu symulacji systemów jazdy autonomicznej, które opisano w dalszej części pracy.

Symulator przygotowany w ramach niniejszej rozprawy, w porównaniu do wspomnianych wyżej rozwiązań, zorientowany jest na dostarczaniu takich danych symulacyjnych, które pozwalają na opracowywanie i dostosowywanie, ale także porównanie i ocenę algorytmów autonomii jazdy dla pojazdów BPL. Ponadto umożliwia symulację działania sensorów oraz platformy jezdnej zarówno w pomieszczeniu zamkniętym, jak i w terenie otwartym, ograniczając potrzebę ciągłego wykorzystania rzeczywistego pojazdu.

Głównym celem pracy było stworzenie mechanizmów symulacyjnych, które pozwoliłyby na ewaluację wybranych algorytmów autonomicznej jazdy. Postawiono tezę, że **w celu opracowania systemu autonomicznej jazdy dla bezzałogowej platformy lądowej możliwe jest przygotowanie systemu symulacji pojazdu i jego czujników w sposób pozwalający dokonać weryfikacji i ewaluacji algorytmów autonomii bez dostępu do fizycznego pojazdu**. W ramach realizacji celu pracy opracowano symulator dla pojazdów typu BPL dowolnej klasy, który może posłużyć do przygotowania, testowania i oceny skuteczności algorytmów autonomicznej jazdy, charakteryzujący się m.in. następującymi cechami:

- zapewnienie środowiska symulacyjnego dla pojazdów typu BPL o dowolnych rozmiarach, przeznaczeniu i konfiguracji napędu, operujących zarówno w terenie, jak i pomieszczeniach zamkniętych;

- zachowanie wysokiej jakości grafiki wirtualnych kamer oraz wydajności pozwalającej na symulację pozostałych czujników;
- zamiana źródła danych sensorycznych z symulatora na fizyczne czujniki nie powinna wymagać zmian w oprogramowaniu i samych algorytmach systemu autonomii, a co za tym idzie, sposób reprezentacji danych i ich format powinien być zgodny z rzeczywistymi urządzeniami;
- możliwość wykorzystania symulatora do doboru sensorów i ich docelowej lokalizacji na danym robocie.

Praca składa się z siedmiu rozdziałów. Rozdział 2 prezentuje szczegóły wybranych zagadnień autonomii jazdy, omawia historię rozwoju pojazdów autonomicznych na przestrzeni lat oraz przedstawia stosowane algorytmy i sensory, których przykłady zastosowań można znaleźć w literaturze. Rozdział 3 zawiera przegląd dostępnych narzędzi symulacji systemów autonomicznych oraz wprowadza zagadnienia będące kluczowymi komponentami opracowywanego systemu: system komunikacji między symulatorem, a autonomią oraz środowisko 3D oparte na silniku graficznym Unity. Rozdział 4 poświęcono na opis utworzonego w ramach pracy Wirtualnego Systemu Testującego, jego architektury, zagadnień symulacji pojazdu, otoczenia oraz poszczególnych implementacji modeli symulacyjnych czujników. Rozdział 5 dotyczy przeprowadzonych badań z wykorzystaniem środowiska symulacyjnego i wykonanych na ich podstawie analiz i ewaluacji algorytmów autonomii jazdy. Rozdział 6 opisuje wdrożenie systemu autonomicznego sterowania na platformę docelową przy zastosowaniu autorskiego symulatora. W rozdziale 7 zaprezentowano końcowe wnioski oraz podsumowano osiągnięte rezultaty, przedstawiono również pomysły na dalsze prace i rozbudowanie opracowanego systemu.

Rozdział 2

Zagadnienia autonomii jazdy

W niniejszym rozdziale został opisany stan wiedzy odnoszący się do systemów autonomicznej jazdy, ich historii, stosowanych rozwiązań sensorycznych i wykorzystywanych algorytmów. Omówienie danych zagadnień ma na celu przedstawienie w jaki sposób działania opisane w dalszej części rozprawy wpisują się w daną dziedzinę wiedzy celem wykazania zasadności wykorzystania symulacji na wielu etapach opracowywania rozwiązań problemów autonomii jazdy.

2.1 Rozwój pojazdów autonomicznych

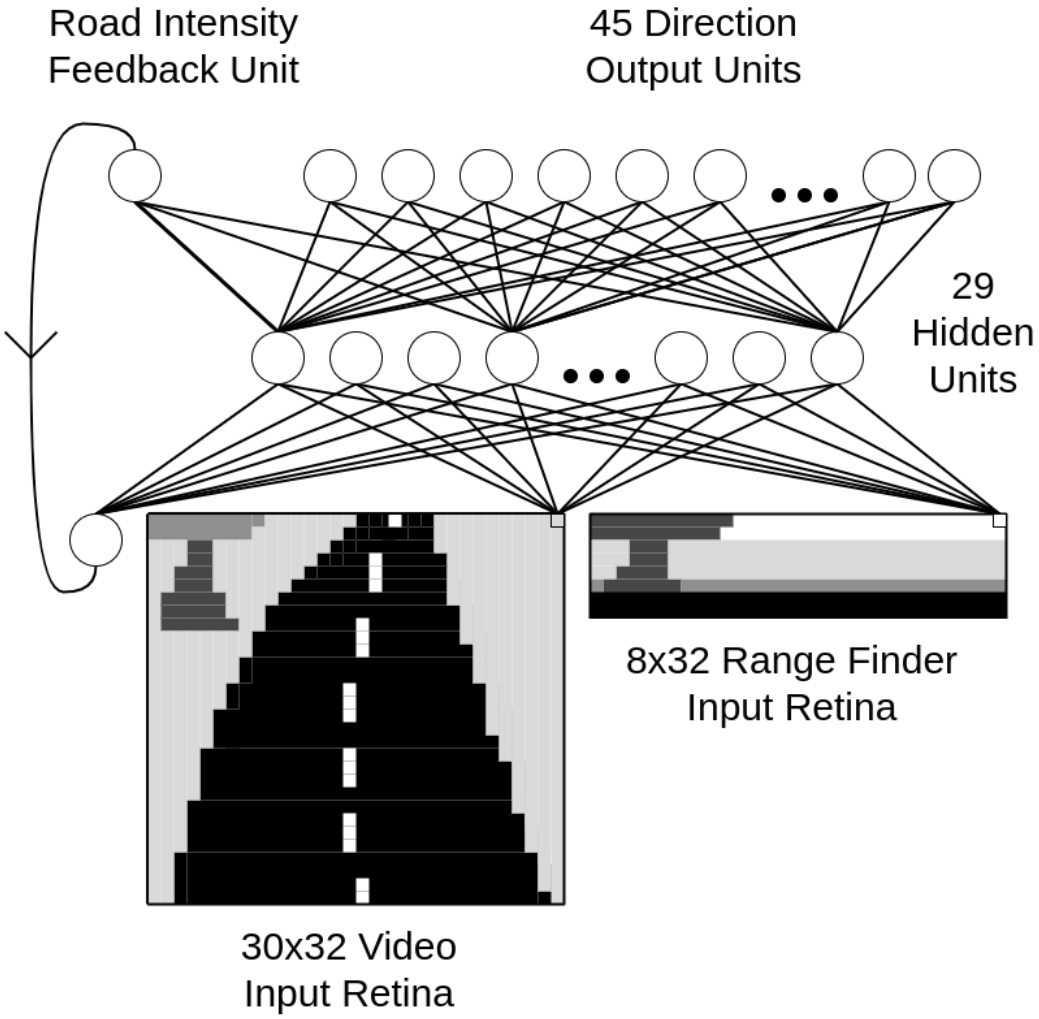
Prace nad pojazdami bez kierowców trwają od wielu lat, najstarsze wzmianki historyczne dotyczą projektów z pierwszej połowy XX wieku [7]. Były to jednak pojazdy zdalnie sterowane, dla których obecność człowieka i jego zaangażowanie było niezbędne do działania. Znacznie bardziej zaawansowanym pojazdem okazał się Stanford Cart z 1961 roku – wózek zdalnie sterowany wyposażony w kamerę [8]. Początkowo został zbudowany w celu weryfikacji możliwości sterowania pojazdem na powierzchni Księżyca przez człowieka na Ziemi, opierając się na przekazywanym sygnale wideo. Projekt rozwijał się na Uniwersytecie Stanforda aż do 1979 roku, kiedy to pojazd potrafił poruszać się autonomicznie z wykorzystaniem wizji po pomieszczeniu zawierającym przeszkody przez kilka godzin bez żadnej interwencji człowieka. Warto jednak odnotować, że przetwarzanie obrazu i planowanie dalszej

ścieżki przejazdu po każdym kolejnym ruchu zajmowało ówczesnym komputerom aż 10 do 15 minut.

W kolejnych latach bardzo ważne w kontekście rozwoju autonomicznych pojazdów okazały się dwa niezależne projekty dotyczące autonomicznych pojazdów: projekt Autonomous Land Vehicle (ALV) [9] ufundowany przez amerykańską agencję DARPA rozpoczęty w 1983 roku oraz 8-letni projekt PROMETHEUS rozpoczęty w 1987 roku zrzeszający ośrodki badawcze i producentów kilku krajów europejskich, sfinansowany przez europejską organizację EUREKA o wartości aż 749 milionów euro [10].

Projekt ALV definiował kilka wymagań funkcjonalnych, które miały spełniać poszczególne podsystemy. Głównym wymaganiem było zapewnienie autonomicznej mobilności w dynamicznym, naturalnym środowisku (z ang. *autonomous mobility in a dynamic unconstrained environment*) oraz pojmowanie otoczenia, określenie odpowiedniej ścieżki przejazdu i utrzymywanie pojazdu na tej ścieżce [9]. Głównymi sensorami odpowiedzialnym za percepcję otoczenia były kamera oraz skaner laserowy. Niestety opracowywane rozwiązanie było niedoskonałe, system wizyjny był podatny na zakłócenia, a oprogramowanie nie potrafiło wykrywać nawet prostych przeszkód poprawnie. Ostatecznie w 1988 roku projekt ALV oficjalnie zamknięto [11].

W tym samym czasie powstawał pojazd ALVINN (z ang. *Autonomous Land Vehicle In a Neural Network*) [12] na uniwersytecie Carnegie Mellon (CMU). Był to jeden z pierwszych projektów, który wykorzystywał sieci neuronowe w praktyce. Pojazd sterowany przez to oprogramowanie potrafił utrzymywać się na pasie ruchu wyznaczonym przez krawędzie drogi. Architektura sieci była mocno ograniczona (rys. 2.1), składała się z jednej warstwy ukrytej z 29 neuronami. Na wejściu był przekazywany obraz z kamery o rozdzielczości 30 pikseli na 32 piksele, odpowiedź z dalmierza laserowego o rozdzielczości 8 pikseli na 32 piksele oraz pojedyncza wartość określająca intensywność drogi względem poprzedniej klatki co łącznie dawało 1217 wejść. Wyjściem sieci było 45 wartości reprezentujących pożądany skręt kół oraz wspomniana intensywność drogi. Trenowanie takiej sieci neuronowej zostało przeprowadzone z użyciem symulatora, który generował wirtualne obrazy dróg. Po-



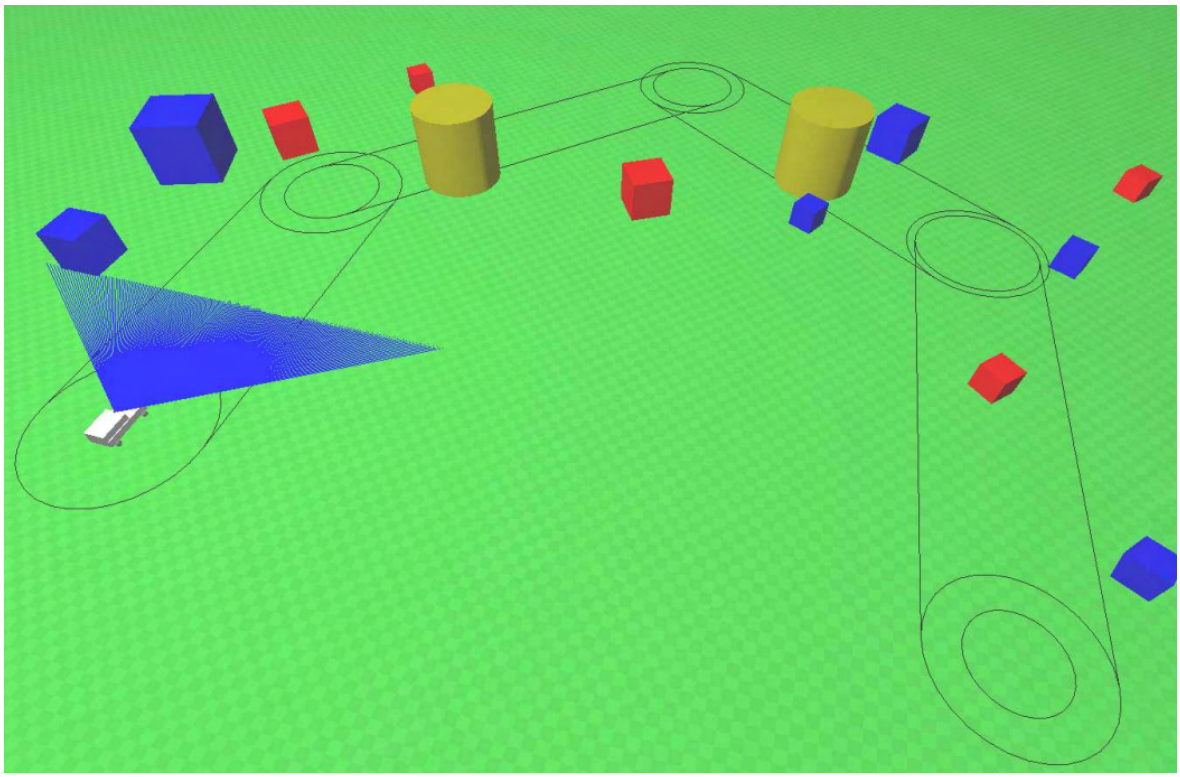
Rysunek 2.1: Architektura sieci neuronowej w projekcie ALVINN [12].

jazd podczas testów w 1989 roku pokonał autonomicznie odcinek 400 metrów utrzymując się na swoim pasie ruchu. Zaprezentowane przez autora podejście było innowacyjne, jednak z perspektywy czasu okazuje się mało optymalne. Przede wszystkim bezpośrednie przekazanie obrazka do sieci neuronowej powoduje, że sieć musi decydować, które informacje na nim są istotne do poprawnego kierowania. Współcześnie sieci konwolucyjne potrafią wyekstrahować kluczowe cechy z klatki filmu, które dopiero przekazywane są do w pełni połączonych warstw sieci neuronowej. Ponadto w przypadku np. wymijania przeszkód potrzebne jest szersze spojrzenie na sytuację na drodze oraz wykonanie szeregu działań, co sprawia, że trenowanie sieci neuronowej powinno przebiegać na znacznie większej ilości danych treningowych, a cały

system wymagałby dodatkowych sensorów wokół pojazdu dla lepszego zorientowania się w przestrzeni. Niemniej jednak praktyczne wykorzystanie sieci neuronowej oraz kamery i czujnika laserowego do sterowania pojazdem pozwala uznać tę pracę jako jedną z pionierowych podejść do rozwoju autonomicznych samochodów. Co istotne, jest to również jeden z pierwszych udokumentowanych przypadków wykorzystujących symulację komputerową w procesie tworzenia systemu autonomii jazdy.

Natomiast wspomniany wcześniej Prometheus był projektem międzynarodowym dotyczącym rozwiązania problemów współczesnego transportu, w którym brali udział europejscy producenci, uniwersytety, ośrodki badawcze oraz instytucje rządowe. Głównymi celami tego przedsięwzięcia było: zwiększenie bezpieczeństwa, zwiększenie wydajności oraz zmniejszenie zanieczyszczenia [13]. W ramach projektu Prometheus udział brał m.in. Ernst Dickmanns uważany za pioniera pojazdów autonomicznych [14] wraz ze swoim zespołem z uniwersytetu Monachium. Zaprezentowali oni 3 pojazdy, które w 1994 roku samodzielnie pokonały ponad 1000 kilometrów po autostradzie wokół Paryża. Rok później postanowiono pokonać autonomicznie 1600 km z Monachium do Odense w Danii. Droga przebiegała autostradą, a pojazd jechał z prędkością pomiędzy 60, a 180 km/h. Co więcej, samochód wykonał autonomicznie ponad 400 wyprzedzeń i zmian pasów, a najdłuższy odcinek drogi pokonany bez interwencji kierowcy wyniósł ok. 160 km [15].

Po roku 2000 w Stanach Zjednoczonych nastąpiła intensyfikacja działań prowadzących do opracowania autonomicznych samochodów. 13 marca 2004 odbyła się pierwsza edycja konkursu Grand Challenge organizowanego przez agencję DARPA, gdzie nagrodą był milion dolarów. Celem dla biorących w nim udział pojazdów autonomicznych było przejechanie trasy o długości 142 mil wyznaczonej na pustyni Mojave w czasie mniejszym niż 10 godzin bez wypadków i interwencji człowieka. Niestety okazało się to wyjątkowo trudnym zadaniem, gdyż żadna z ekip biorących udział w wyzwaniu nie zdołała pokonać nawet 5% dystansu wyścigu. W związku z tym, ponad 1,5 roku później, powtórzono konkurs, zwiększono nagrodę dwukrotnie, a trasę wydłużono do 175 mil. 8 października 2005 wystartowały 23 samochody z czego jedynie 5 zdołało dojechać do mety. Zwycięski okazał się pojazd Stanley



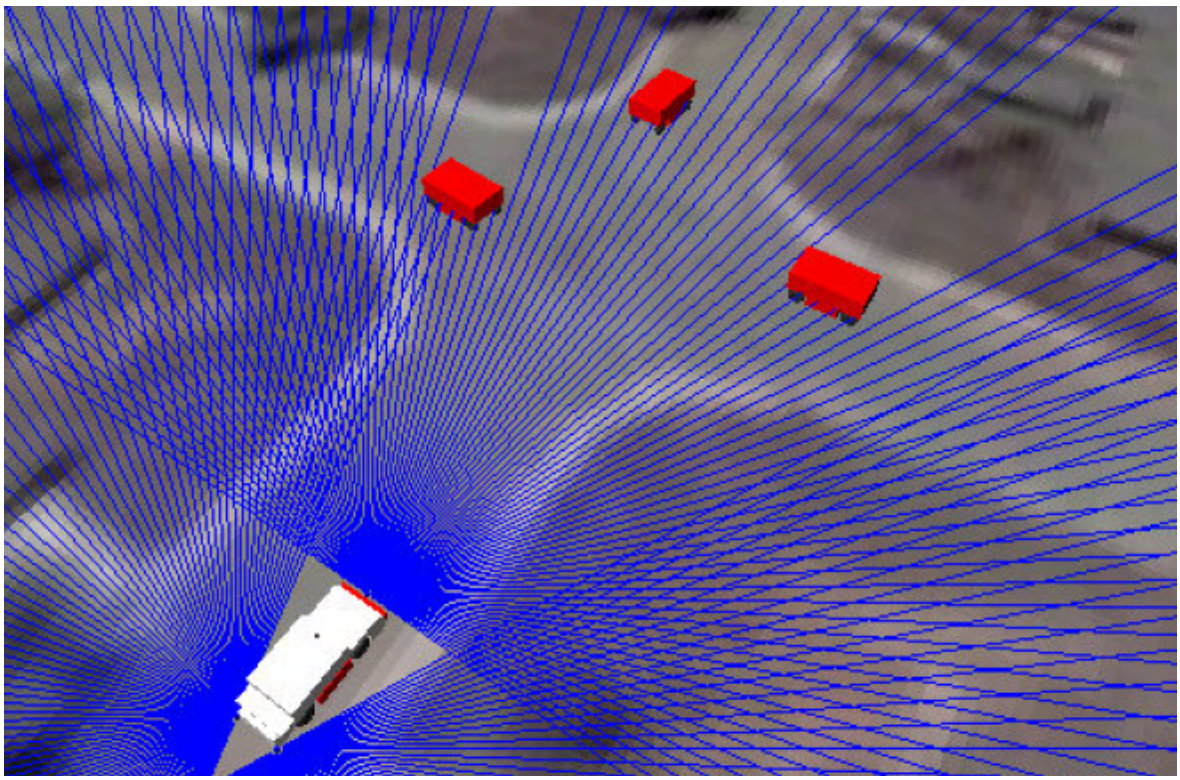
Rysunek 2.2: Uproszczone środowisko symulacyjne 3D zespołu Caltech biorącego udział w DARPA Grand Challenge [16].

naukowców z Uniwersytetu Stanforda z Sebastianem Thrunem na czele. Wykorzystali szereg wzajemnie uzupełniających się czujników (m.in. lidary i kamery) oraz przygotowali rozbudowane oprogramowanie wielowątkowe, które zawierało ok. 30 równolegle pracujących modułów odpowiedzialnych za poszczególne zadania takie jak percepcja otoczenia czy kontrola pojazdu [17]. Wśród wybranych prac naukowych dotyczących poszczególnych rozwiązań biorących udział w konkursie, omawiane są również zagadnienia związane z zastosowaniem technik symulacji komputerowej podczas rozwoju konkretnych funkcjonalności pojazdów autonomicznych. Przykładowo w [16] zostało przedstawione wykorzystanie ówcześnie nowo powstałego środowiska symulacyjnego Gazebo opartego na projektach Player i Stage [18], który to symulator rozwijany jest do dziś w celu badań nad systemami robotycznymi i sensorycznymi. Zastosowano go w celu rozwoju i testowania oprogramowania odpowiedzialnego za planowanie trasy przejazdu jednego z zespołów biorących udział w konkursie. Było to możliwe do zrealizowania dzięki zastosowaniu uproszczonych

modeli symulacyjnych lidaru, czujnika położenia i GPS oraz modelu samego pojazdu, które uruchamiane były w bardzo ograniczonym środowisku 3D (rys. 2.2). Niemniej jednak w tym przypadku użycie symulatora było ograniczone ze względu na chęć twórców projektu do wykonywania testów w realnym świecie w warunkach pustynnych odpowiadających rzeczywistym, które występują na trasie konkursu.

Dwa lata później, na początku listopada 2007 roku odbył się kolejny konkurs z serii Grand Challenge nazywany Urban Challenge. Tym razem zasady były bardziej złożone. Celem było autonomiczne poruszanie się po mieście i wykonywanie misji transportowych pomiędzy określonymi punktami, co – ze względu na charakter agencji ogłaszającej konkurs – miało symulować dostawy wojskowe w obszarach miejskich. Oprócz przestrzegania zasad ruchu drogowego wymagane były również następujące manewry: wymijanie zaparkowanych lub wolniejszych pojazdów, ustępowanie pierwszeństwa, włączanie się do ruchu na drodze szybkiego ruchu, skręt w lewo na skrzyżowaniu, parkowanie w miejscu parkingowym, wykonanie zawracania kiedy droga jest zablokowana. W ramach przygotowania do konkursu DARPA zapewniła szczegółowy plan dostępnego obszaru wraz z informacjami o liniach, znakach poziomych i pionowych, miejscach parkingowych i specjalnych punktach kontrolnych. Dodatkowo dostarczono zdjęcie lotnicze wysokiej rozdzielczości z całym obszarem. Prędkość pojazdów była ograniczona do maksymalnie 30 mph. Finał odbywający się 3 listopada został poprzedzony eliminacjami, z których wyłoniono 11 samochodów. Podczas konkursu pojazdy wykonywały zadania w tym samym czasie wraz z 50 innymi samochodami kierowanymi przez ludzi. Zwycięzcą okazał się robot Boss zaprojektowany przez uczelnię CMU i firmę General Motors, który w najkrótszym czasie (4h 10m 20s) poradził sobie ze wszystkimi zadaniami. 19 minut później wyścig na drugim miejscu zakończył Junior opracowany przez ekipę z Uniwersytetu Stanforda, którą tworzyli w większości ci sami ludzie co Stanleya – zwycięzcę poprzedniego Grand Challenge. Oba samochody wykorzystywały wiele czujników dla jak najlepszej percepcji otoczenia. Bardzo ciekawe okazało się, że w obu przypadkach postanowiono nie korzystać z kamer ze względu na bardziej złożony proces interpretacji danych, stwierdzono bowiem, że w tym konkursie wystarczające będą czujniki aktywne takie jak radary czy lidary [19]. Analizując podejścia poszcze-

gólnych zespołów naukowców do problemu można zauważyć wyraźnie, że istotnym elementem prac zaczęło być wykorzystanie symulacji komputerowej do przeprowadzania testów i weryfikacji działania opracowywanych algorytmów [20–22]. Przykładowo rys. 2.3 prezentuje środowisko 3D z wirtualnymi pojazdami oraz symulacją trzech czujników lidar. Pozorowanie hipotetycznych sytuacji drogowych oraz analiza odpowiedzi algorytmów na syntetyczne dane generowane przez uproszczone modele sensorów pozwoliła, przy ograniczonych ilościach czasu i zasobach, opracować system zdolny z powodzeniem wykonywać złożone zadania konkursowe.



Rysunek 2.3: Środowisko symulacyjne obszaru Urban Challenge wraz z danymi lidarowymi stosowane przez Ohio State University [20].

Konkursy zorganizowane przez agencję DARPA wpłynęły na dynamiczny rozwój systemów autonomicznej jazdy. W 2009 roku Google rozpoczął prace nad swoim pojazdem samokierującym, w których początkowo udział wzięło 15 inżynierów. Byli to głównie dawni uczestnicy Grand Challenge i Urban Challenge z Uniwersytetu Stanforda pod kierownictwem Sebastiana Thruna. Tajny projekt potentata internetowego ujrzał światło dzienne w maju 2012, kiedy to po raz pierwszy w historii,

w obecności władz stanu Kalifornia w USA, przeprowadzono test pojazdu na drogach publicznych [23]. Toyota Prius zmodyfikowana przez Google została pierwszym samochodem dopuszczonym do ruchu jako pojazd autonomiczny. W kolejnych latach projekt dynamicznie rozwijał się, aż w 2016 roku została wydzielona osobna firma Waymo, która obecnie może pochwalić się flotą ponad 600 pojazdów samokierujących i ponad 20 milionami przejechanych mil po drogach publicznych [24]. Dodatkowo inżynierowie i naukowcy Waymo wykorzystują środowisko symulacyjne SimulationCity [25], w którym wirtualne pojazdy pokonały do tej pory dystans ponad 15 miliardów mil. Stosowany przez firmę symulator jest bardzo zaawansowanym narzędziem. Pozwala generować syntetyczne dane sensoryczne w wirtualnym środowisku, które odpowiada rzeczywistym obszarom działania floty pojazdów i symuluje realne zachowania innych uczestników ruchu. W celu urzeczywistnienia danych uzyskiwanych z wirtualnych sensorów stosowana jest sieć neuronowa GAN [26], która opierając się na danych sensorycznych zebranych podczas rzeczywistych przejazdów potrafi zmodyfikować dane z wirtualnej kamery i lidaru tak, aby odpowiadała jak najbardziej rzeczywistemu obrazowi. Ponadto zastosowanie symulatora pozwoliło naukowcom potwierdzić istotną rolę pojazdów autonomicznych w zapobieganiu wypadków [27]. Przeanalizowali oni bowiem 10-letni zbiór wypadków w symulacji gdzie jednym z zaangażowanych w niebezpieczną sytuację był pojazd Waymo. Okazało się, że we wszystkich przypadkach system potrafi uniknąć zderzenia z innym uczestnikiem ruchu.

Obecnie większość producentów samochodów bądź dostawców rozwiązań z branży automotive opracowuje własne systemy zautomatyzowanej i autonomicznej jazdy. Temat ten angażuje poza Google również innych gigantów branży IT jak Uber bądź Apple. Istotną rolę odgrywają również tzw. startupy, które dostarczają rozwiązań skupiających się wokół pojazdów autonomicznych prężnie rozwijając technologie z pogranicza informatyki i robotyki, która wpływa na przyszłość transportu zarówno osobowego, jak i zdefiniowanego szerzej transportu drogowego. Także w kontekście zastosowania symulatorów następuje stały rozwój technologii. Rozwijane jest kilka wirtualnych środowisk symulacyjnych jak Carla [6] czy AirSim [28]. Istnieją również inne narzędzia inżynierskie jak Matlab/Simulink z pakietem Automated Driving To-

olbox [29] czy Ansys Autonomous Vehicle Simulation [30], które mogą dostarczyć modeli zachowań poszczególnych komponentów systemu i posłużyć jako elementy w procesie testowania i walidacji algorytmów autonomicznej jazdy lub systemów ADAS (z ang. *Advanced Driver Assistance Systems*).

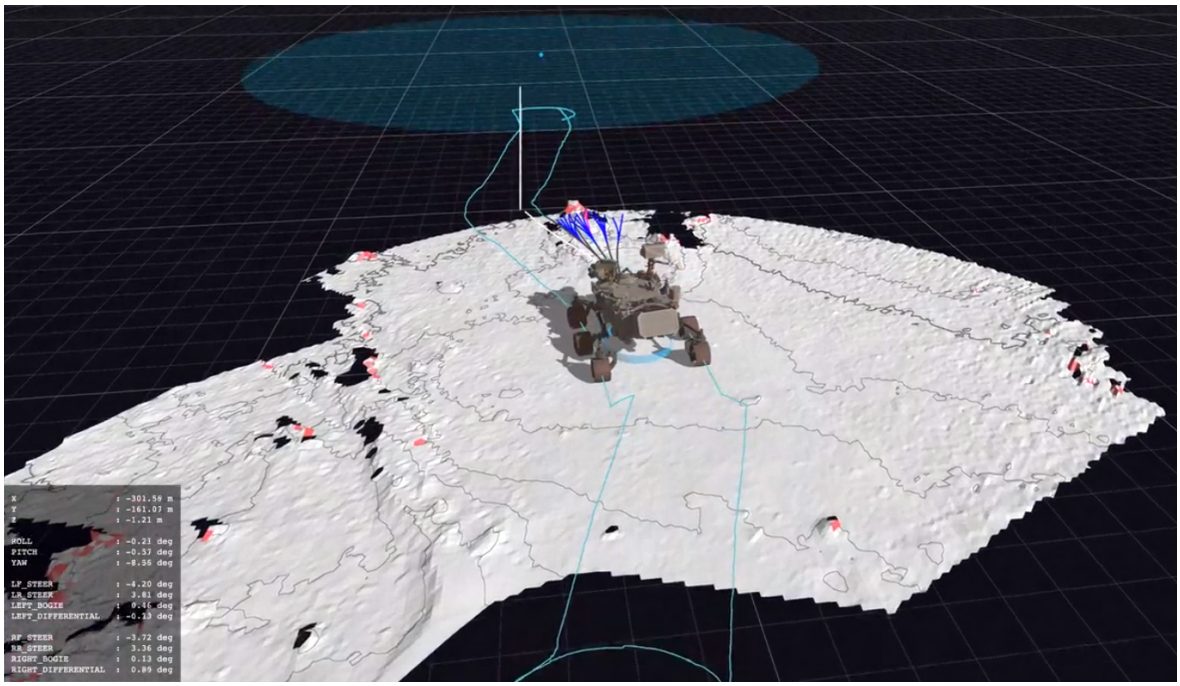
Na przestrzeni ostatnich lat powstało wiele klasyfikacji określających stopień autonomii pojazdów. Jedną z najpopularniejszych jest przedstawiony przez organizację SAE International (z ang. *Society of Automotive Engineers*) opis sześciu poziomów autonomii jazdy [31]. Szczegółowo definiuje on stopień zaangażowania systemów w proces przemieszczania się samochodu po drogach publicznych od całkowitego braku aktywnych asyst (poziom 0), aż do pełnej autonomii w każdych warunkach (poziom 5). W tabeli 2.1 przedstawiono podsumowanie kolejnych poziomów autonomii jazdy według klasyfikacji SAE. W podobny sposób opisywane są stopnie automatyzacji zadań w innych obszarach, np. w pracy [32] dokonano przeglądu poziomów autonomii w kontekście interakcji z człowiekiem robotów i innych pojazdów bezzałogowych. Autorzy zaprezentowali 10-stopniową klasyfikację ze względu na automatyzację procesów decyzyjnych, ale wspominają również o podziale ze względu na poziom interakcji z operatorem, stopień skomplikowania misji czy złożoność środowiska, w którym operuje system.

Warto również wspomnieć, że w konkursach Grand Challenge brała także udział firma Oshkosh, która zaczęła opracowywać technologię TerraMax. Nastawiona jest ona na zastosowania militarne w misjach rozpoznawczych czy transportowych na obszarach o podwyższonym ryzyku ataku [33]. Niestety ze względu na charakter firmy dostępna jest tylko szczątkowa ilość informacji dotyczących zastosowanej technologii. Wspomniany projekt pokazuje jednak, że technologia autonomii jazdy opracowywana jest nie tylko w kontekście samochodów osobowych i cywilnych zastosowań. Zagadnienie to ważne jest również w wielu innych obszarach nauki i przemysłu. W tym miejscu warto wspomnieć chociażby o istocie stosowania autonomicznych pojazdów do celów eksploracji przestrzeni kosmicznej. Wszelkiego rodzaju łaziki obecne np. na powierzchni Marsa działają głównie w trybie półautonomicznym, w którym to operator na Ziemi nadzoruje rzeczywisty przejazd robota po wcześniejszym jego zaprogramowaniu i przetestowaniu w symulatorze

<table>
<tr><th>poziom SAE</th><th>stopień zaangażonwania kierowcy</th><th>cechy systemu</th><th>przykładowe funkcjonalności</th></tr>
<tr><td>0</td><td rowspan="3">niezbędne ciągłe nadzorowanie systemu</td><td>zapewnia ostrzeżenia lub chwilową asystę kierowcy</td><td>automatyczne hamowanie awaryjne, ostrzeżenie o pojeździe w martwym punkcie, ostrzeżenie o przekroczeniu pasa jezdni</td></tr>
<tr><td>1</td><td>zapewnia pomoc przy sterowaniu lub hamowaniu/ przyspieszaniu</td><td>utrzymywanie na pasie ruchu lub adaptacyjny tempomat</td></tr>
<tr><td>2</td><td>zapewnia pomoc przy sterowaniu i hamowaniu/ przyspieszaniu</td><td>utrzymywanie na pasie ruchu i adaptacyjny tempomat</td></tr>
<tr><td>3</td><td>system może zażądać od kierowcy przejęcia sterowania</td><td rowspan="2">potrafi prowadzić pojazd jedynie podczas określonych warunków</td><td>asystent jazdy w korku</td></tr>
<tr><td>4</td><td rowspan="2">system nie będzie wymagał przejęcia sterowania</td><td>autonomiczna taksówka bez kierowcy, niewymagane pedały i kierownica</td></tr>
<tr><td>5</td><td>potrafi prowadzić pojazd we wszystkich warunkach</td><td>takie jak przy poziomie 4, ale w dowolnych warunkach</td></tr>
</table>

Tablica 2.1: Poziomy autonomii jazdy SAE [31].

na Ziemi. Niemniej jednak stopniowe zwiększanie zdolności autonomicznej jazdy zwiększa m.in. zasięg działania pojazdu, jego prędkość poruszania, a co za tym idzie również w znaczący sposób poszerza obszar badań i eksploracji powierzchni Czerwonej Planety [34]. Dobrym przykładem jest tutaj łazik Perseverance, który potrafi przemieszczać się z 6-krotnie większą prędkością względem swojego poprzednika Curiosity (120 metrów na godzinę w porównaniu do 20 metrów na godzinę). Udało się to osiągnąć dzięki nowszej wersji oprogramowania AutoNav [35] stosowanego w łazikach NASA. Weryfikacja zdolności pokonywania określonej trasy odbywa się przy wykorzystaniu środowiska symulacji SSim opracowanego przez Amerykańską



Rysunek 2.4: Symulacja działania systemu AutoNav łazika Perseverance [37].

Agencję Kosmiczną [36]. Oprogramowanie to pozwala na symulowanie działania modułu planowania (rys. 2.4), jak i innych kluczowych podsystemów łazika, w tym instrumentów pokładowych, zasilania czy komunikacji.

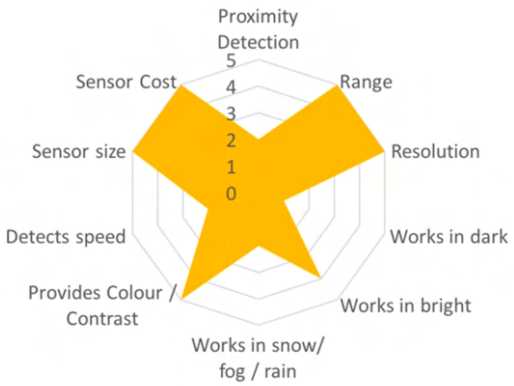
Zagadnienia autonomii jazdy pojawiają się obecnie niemal we wszystkich dziedzinach technologii i życia codziennego, która wymaga mobilności i może podlegać robotyzacji. Oprócz omówionych wcześniej motoryzacji i transportu osób istotnym obszarem wydaje się także rolnictwo. Zastosowanie AI (z ang. *Artificial Intelligence*), w tym metod uczenia maszynowego czy algorytmów wizji komputerowej opracowywanych w ostatnich latach [38–41], pozwala dzisiaj na implementację rozwiązań w produktach końcowych, będących krok przed masową produkcją. Przykładowo firma Carbon Robotics zaprezentowała autonomiczny opielacz do usuwania chwastów [42]. Wytrenowane modele sieci neuronowych pomagają osiągać milimetrową dokładność przy eliminacji niepożądanych roślin w uprawach. Ponadto dane z kamer 3D przetwarzane przez algorytmy percepcji otoczenia czy nawigacji zapewniają precyzyjny przejazd w obszarze rolnym bez asysty operatora. Z kolei przedsiębiorstwo John Deere będące światowym liderem w technologii rolnej zaprezento-

wało na początku 2022 roku pierwszy ciągnik wyposażony w funkcje autonomicznej jazdy, który umożliwia kierowcy opuszczenie pojazdu podczas pracy systemu [43]. Algorytmy oparte na sieciach neuronowych pozwalają na zrozumienie świata otaczającego ciągnik, co pozwala na bezpieczne pokonywanie trasy przy wykonywaniu prac na polach uprawnych. Innym obszarem technologii, w którym rozwijane i testowane są samokierujące pojazdy to tzw. transport ostatniej mili (z ang. *last-mile delivery*) [44]. Pomijając transport drogowy, czyli przewóz osób i towarów po drogach publicznych, można wskazać np. roboty Starship [45] czy Delivers.ai [46], które, w założeniach, poruszając się po chodnikach wśród ruchu pieszych mają dostarczać jedzenie, zakupy bądź inne drobne przesyłki w okolicy w sposób autonomiczny pokonując trasę od dostawcy do klienta. Również i w takich przypadkach naturalnym wydaje się zastosowanie symulatorów takich jak wspomniane wcześniej Gazebo bądź Carla na potrzeby walidacji opracowywanych algorytmów czy modeli.

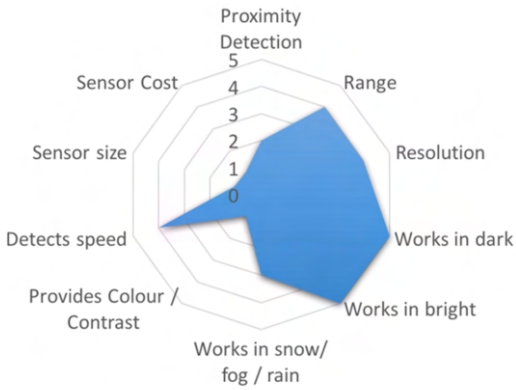
2.2 Sensory

Pojazdy autonomiczne, aby poruszać się samodzielnie wymagają sensorów, które pozwalają na zbieranie informacji o otoczeniu, które następnie trafiają do właściwych algorytmów danego systemu. Uzyskane w ten sposób dane są następnie przetwarzanie przez algorytmy, które umożliwiają lokalizację pojazdu, detekcję i klasyfikację obiektów wokół niego czy wyznaczanie bezpiecznej drogi i nawigowanie do celu. Dostępnych jest wiele rodzajów czujników wyróżniających się parametrami pracy czy innymi właściwościami zapewniającymi bezproblemowe działanie w różnych warunkach oświetleniowych czy pogodowych. Na rys. 2.5 przedstawiono wpływ także innych atrybutów poszczególnych rodzajów urządzeń, który został oszacowany na podstawie wieloletnich prac realizowanych przez Lexa Fridmana, badacza AI pracującego w MIT (z ang. *Massachusetts Institute of Technology*). Różnice skuteczności działania danych sensorów w konkretnych aspektach pokazują, że kompletny system, którego działanie powinno być jak najbardziej niezawodne, powinien być wyposażony w kilka rodzajów sensorów uzupełniających się między sobą.

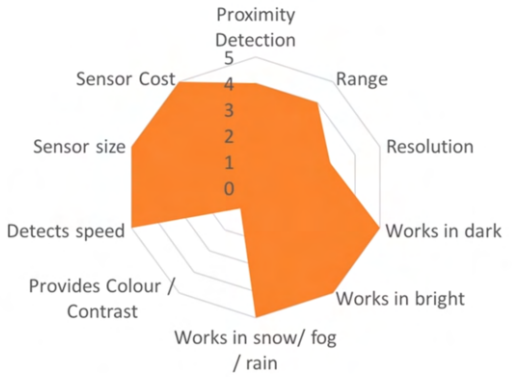
Analizując rodzaje czujników, które można zastosować dla potrzeb systemu au-



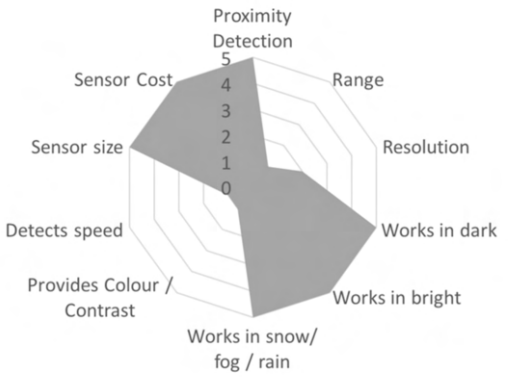
(a) Kamera.



(b) LiDAR.



(c) RADAR.



(d) Ultradźwięki.

Rysunek 2.5: Skuteczność działania sensorów w konkretnych obszarach [47].

tonomicznej jazdy można dokonać podziału ze względu na sposób odbioru sygnałów z otoczenia na dwie główne grupy:

- sensory pasywne,
- sensory aktywne.

Główna różnica między nimi polega na tym, że sensory pasywne jedynie odbierają sygnały ze środowiska, natomiast sensory aktywne dokonują pomiarów najpierw emitując sygnał, a następnie analizując otrzymaną odpowiedź. Do pierwszej grupy zalicza się przede wszystkim wszelkiego rodzaju kamery takie jak kamery stereowizyjne bądź termowizyjne, ale także czujniki IMU wyposażone zwykle w ak-

celerometr, żyroskop oraz magnetometr. Z kolei do sensorów aktywnych należą lidary, radary czy ultradźwiękowe lub inne czujniki odległości. W kolejnych sekcjach omówione zostaną te najszerzej używane w kontekście systemów autonomii jazdy dla bezzałogowych platform lądowych lub innych samokierujących pojazdów.

2.2.1 Kamery

Występuje kilka rodzajów kamer, które mogą znaleźć zastosowanie w systemach autonomii wszelkich rodzajów pojazdów. Można wyróżnić przede wszystkim standardowe kamery jednoobiektywowe wyposażone zazwyczaj w matryce o dużej rozdzielczości typu CMOS (z ang. *Complementary Metal-Oxide-Semiconductor*), które rejestrując kolorowy obraz potrafią dostarczać duże ilości danych do dalszego przetwarzania i analizy. Ponadto stosowane są kamery stereowizyjne wyposażone w dwie matryce i obiektywy, które dzięki kalibracji zapewniają dodatkowe informacje o głębi obrazu wykorzystywane przy lokalizacji czy nawigacji. Co więcej, można wyróżnić grupę kamer specjalistycznych takich jak np. kamery termowizyjne. Dostarczają one dodatkowych danych, głównie w kontekście obserwacji otoczenia, oferując większą pewność i niezawodność przy wykrywaniu obiektów, szczególnie stanowiąc uzupełnienie kamer działających w standardowym paśmie światła widzialnego.

Na podstawie obrazu z dowolnych kamer, z pomocą wyuczonych sieci neuronowych, można przeprowadzać segmentację semantyczną widzianej sceny, rozpoznawać obiekty w środowisku otaczającym pojazd [48] oraz jednocześnie tworzyć mapę 3D i lokalizować swoja pozycję (SLAM) [49]. Ponadto przeprowadzając fuzję danych z LiDAR-ami można uzyskać dokładniejsze wyniki [50] lub zapewnić redundancję systemów dla zwiększenia bezpieczeństwa.

Skuteczność działania kamer może być ograniczona przez brak widoczności spowodowany przez słabe warunki oświetleniowe, gęstą mgłę czy niekorzystne warunki pogodowe. Jednak koszty, niewielki rozmiar, zasięg działania, duża rozdzielczość i ilość dostarczanych danych są niewątpliwym zaletą i głównym powodem popularności stosowania kamer w systemach autonomicznej jazdy.

2.2.2 LiDAR

LiDAR (z ang. *Light Detection And Ranging*) jest urządzeniem wyposażonym w jeden lub wiele laserów, które emitują impulsy o długości fali w zakresie bezpiecznym dla ludzkiego oka (zazwyczaj w okolicach wartości 905 nm) oraz odpowiadające poszczególnym laserom fotodetektory. Dzięki precyzyjnemu pomiarowi czasu pomiędzy emisją, a detekcją wiązki lasera, obliczana jest odległość do obiektu odbijającego promień i tym samym współrzędna punktu w przestrzeni. Istnieje wiele rodzajów tych urządzeń, z czego najpopularniejszym rozwiązaniami w kontekście zastosowań robotycznych wydają się lidary 2D i 3D.

LiDAR 2D wyposażony jest w pojedynczy laser, który obracając się wokół osi pionowej skanera dokonuje pomiarów zwracając skan laserowy składający się z dwuwymiarowych punktów XY. Z kolei lidary 3D są zwykle wyposażone w 16 do nawet 128 laserów ustawionych pionowo i pochylonych pod różnym kątem obracając się nawet z prędkością 1200 obrotów na minutę. Dzięki temu możliwe jest zebranie szczegółowej chmury punktów otoczenia składającej się z trójwymiarowych współrzędnych XYZ. Chmura taka może zostać wykorzystana np. do wygenerowania mapy wysokiej rozdzielczości i dokonania pozycjonowania w tej mapie [51]. Ponadto dane z takich urządzeń mogą posłużyć do wykrywania i klasyfikacji obiektów [52,53] lub segmentacji semantycznej punktów [54].

Do korzyści płynących z wykorzystywania LiDAR-ów w systemach autonomii można zaliczyć duży zasięg (maksymalnie nawet 300 metrów w zależności od modelu), sporą rozdzielczość danych oraz działanie niezależne od warunków oświetleniowych. Natomiast niewątpliwą wadą tego rozwiązania jest koszt zważywszy na precyzję wykonania tych urządzeń, ale także rozmiar urządzenia ze względu na ruchome elementy. W ostatnich latach poczyniono jednak sporo wysiłku i badań, aby zniwelować te wady, co zaowocowało pojawieniem się tzw. lidarów *solid state* wykonanych w technologii MEMS (a ang. *Microelectromechanical System*) charakteryzujących się stałą matrycą nadajników i odbiorników bez ruchomych części. Pozwoliło to znacząco zmniejszyć urządzenie oraz ograniczyć koszty, a przy tym dodatkowo zwiększyć rozdzielczość uzyskiwanych danych.

2.2.3 RADAR

RADAR (z ang. *Radio Detection And Ranging*) za pomocą fal radiowych potrafi wykrywać i rozpoznawać obiekty, określać dystans i ich prędkość. Ponadto jest skuteczny w warunkach, w których praca innych sensorów jest utrudniona: przy intensywnych opadach, w ciemności (noc) i w gęstej mgle. Koszty, rozmiar urządzenia, dokładność czy zasięg działania także są niewątpliwymi zaletami stosowania tych urządzeń. Jedyną wadą w porównaniu do pozostałych wykorzystywanych sensorów jest rozdzielczość i mniejsza ilość otrzymywanych danych. Radary są popularne w systemach autonomicznych i systemach ADAS dla motoryzacji. Spowodowane jest to lepszą zdolnością wykrywania obiektów przy wyższych prędkościach autostradowych. Pojazdy bezzałogowe z kolei poruszają się z mniejszymi prędkościami, co sprawia, że zazwyczaj zestaw LiDAR + system wizyjny oparty na kamerach jest rozwiązaniem skuteczniejszym od radarów, szczególnie gdy koszty nie odgrywają najważniejszej roli.

Niemniej jednak radary mogą pełnić istotną rolę w dostarczaniu danych o otoczeniu na potrzeby detekcji obiektów. Zastosowanie urządzeń wykorzystujących nadajniki i odbiorniki fali o odpowiedniej długości pozwala skutecznie wykrywać i śledzić obiekty [55]. Dodatkowo, wspomagając się danymi wizyjnymi z kamer mono [56] lub stereo [57] możliwe jest zwiększenie dokładności detekcji przeszkód, również wykorzystując techniki głębokiego uczenia maszynowego [58].

2.2.4 Ultradźwiękowy czujnik odległości

Czujniki odległości, opierające swoje działanie na ultradźwiękach, wykorzystują emiter i detektor fali dźwiękowej o wysokiej częstotliwości, na podstawie której mierzona jest odległość do przeszkody. Największą zaletą tych czujników jest ich niewielki koszt zakupu oraz dokładność pomiarów w bliskiej odległości. Skuteczność działania czujników ultradźwiękowych jest również niezależna od warunków pogodowych oraz oświetleniowych. Wadą tego typu sensorów jest jednak niewielki zasięg i rozdzielczość oraz ilość otrzymywanych danych.

Głównym zastosowaniem ultradźwiękowych czujników odległości jest wykrywanie przeszkód na niewielkim dystansie przy niskiej prędkości. Ich zastosowanie ogranicza się zazwyczaj do czujników parkowania w samochodach osobowych oraz do prostego wykrywania przeszkód przez małe pojazdy, których celem jest poruszanie się głównie wewnątrz pomieszczeń. Wśród przykładów z literatury można znaleźć m.in. niskobudżetowe rozwiązania skupiające się wokół niewielkich robotów zdolnych do autonomicznej nawigacji do wybranego celu wraz z omijaniem przeszkód bazując na wykrywaniu obiektów na podstawie czujników ultradźwiękowych [59,60]. Potencjalnie jednak możliwe jest również zastosowanie sieci neuronowych do bardziej efektywnego wykrywania przeszkód dysponując jedynie zbiorem odczytów z tego typu sensorów [61]. Ponadto uzyskiwane odczyty można także zastosować jako źródło danych dla algorytmów lokalizacji i mapowania przestrzeni, również w nieznanym otoczeniu [62].

2.2.5 IMU

IMU (z ang. Inertial Measurement Unit) jest urządzeniem wyposażonym w akcelerometr, żyroskop, ewentualnie w magnetometr czy czujnik temperatury. Udostępnia dane takie jak prędkości kątowe czy przyspieszenia działające na obiekt. Pozwala na określenie orientacji w przestrzeni, a nawet pozycji dzięki metodom nawigacji zliczeniowej. Umożliwia również wspomaganie pracy odbiorników GNSS podczas braku sygnału, np. w tunelach czy wewnątrz budynków. Ponadto dane pochodzące z IMU są często wykorzystywane przy fuzji innych sensorów takich jak kamera czy lidar. Pozwalają bowiem na wyznaczanie lokalizacji przy jednoczesnym mapowaniu danego obszaru. Zapewniają również źródło kompensacji błędów przy algorytmach wyznaczających odometrię pojazdu.

Przykładem zastosowania IMU jest system lokalizacji pojazdu i mapowania otoczenia z wykorzystaniem LiDAR-u [63]. Dzięki pomiarom przyspieszenia i prędkości kątowych oraz korzystając dodatkowo z prędkości i przebytego dystansu obliczonego na podstawie danych z enkoderów kół stosowana jest fuzja informacji oraz aktualizacja i korekta wyznaczonej pozycji robota w przestrzeni. Połączenie odczytów

uzyskiwanych przez różne typy czujników wraz z danymi uzyskiwanymi z IMU może służyć nie tylko do lokalizacji, ale też do mapowania otoczenia [64]. W podobny sposób dokonywana jest również integracja pomiarów globalnej pozycji dostarczanej przez system GPS z dowolnym sensorem IMU z myślą o zapewnienie nawigacji w środowisku zewnętrznym [65]. Tego typu fuzja danych może być przeprowadzona również z użyciem odpowiedniego filtru Kalmana, a nawet ich kombinacji [66]. Ponadto czujnik IMU dostarcza informacji o siłach działających na pojazd co może zostać wykorzystane do detekcji niespodziewanych zdarzeń takich jak poślizg kół [67] czy błędy kalibracji kamery spowodowane nadmiernymi wibracjami [68].

2.2.6 GNSS

GNSS (z ang. Global Navigation Satellite System) to dowolny system nawigacji satelitarnej, który umożliwia określenie bezwzględnej pozycji na Ziemi podając parametry długości i szerokości geograficznej, a także wysokości nad poziomem morza. Dzięki satelitom umieszczonym na orbicie ziemskiej, które przesyłają sygnały radiowe do odbiorników, zapewnia aktualną pozycję z dużą dokładnością sięgającą, w zależności od urządzenia i użytej technologii, od ok. 1 metra do nawet kilku centymetrów.

Ograniczenie pracy systemu nawigacji satelitarnej może wystąpić, gdy przestrzeń nad odbiornikiem zostanie przysłonięta, np. w tunelu czy w budynku. W zastosowaniach militarnych poleganie w pełni na systemach GNSS może okazać się zgubne w przypadku działań wojennych, gdyż systemy te są rozwiązaniami państwowymi i należą do takich potęg militarnych jak USA, Rosja czy Chiny, co w razie wyłączenia lub zaszyfrowania sygnału stwarza niebezpieczeństwo niepowodzenia misji.

Jako, że systemy nawigacji satelitarnej takie jak np. GPS służą do wyznaczania globalnej pozycji na Ziemi dysponują ograniczoną dokładnością. Dlatego też popularnym rozwiązaniem jest stosowanie fuzji danych z sensorem IMU w celu zapewnienia najwyższej dokładności pomiarów co jest krytyczne w kontekście autonomii jazdy, szczególnie w przypadku samochodów czy innych pojazdów poruszających się

po drogach publicznych. W literaturze można znaleźć szereg prac traktujących o integracji danych IMU z GNSS [69–71], znacząco zwiększając dokładność lokalizacji i zapewniając większą niezawodność systemu. Aby zachować najwyższą jakość pomiarów, ale i bezpieczeństwo, niezbędne jest monitorowanie jakości uzyskiwanych odczytów [72], jak również detekcja ataków typu *spoofing* [73], polegających na przekłamywaniu odczytów GNSS. Ciekawym rozwiązaniem jest także opracowanie algorytmu planowania drogi opierającego się na ukształtowaniu terenu i wykrywaniu potencjalnych obszarów o ograniczonej widoczności satelitów [74].

2.3 Algorytmy autonomicznej jazdy

Na przestrzeni ostatnich lat opracowano wiele różnego rodzaju podejść do architektury systemów autonomicznej jazdy. Projekty badawcze lub konkursy typu Grand Challenge pozwoliły na opracowanie wzorców, które z każdym kolejnym projektem stają się powoli standardem. W pracy [75] autorzy wyróżniają dwa rodzaje architektury pojazdów autonomicznych różniące się między sobą sposobem komunikacji z pojazdem i jego sterowaniem. W pierwszym sposobie zakłada się, że system inteligencji jazdy wysyła komunikaty bezpośrednio kontrolujące pojazd (przyspieszenie, hamowanie, skręt kół). Z kolei drugie podejście polega na rozdzieleniu systemów bezpieczeństwa i elementów wykonawczych od modułów autonomicznej jazdy. System autonomii generuje trajektorię, po której powinien poruszać się pojazd, a następnie jest ona realizowana przez odseparowany moduł kontroli pojazdu. Dzięki temu krytyczne systemy odpowiedzialne za bezpieczeństwo, zarządzanie energią czy stabilizację toru jazdy działają niezależnie i gwarantują wyższy poziom bezpieczeństwa poprzez uproszczenie zależności systemu. Analizując wiele prac można zauważyć, że wykształciło się kilka paradygmatów dotyczących oprogramowania i budowy pojazdów autonomicznych. Przede wszystkim można zauważyć, że istnieje określony zbiór najczęściej wykorzystywanych typów czujników, do których możemy zaliczyć LiDAR-y, kamery RGB czy radary. Jest to spowodowane określonymi funkcjonalnościami, które spełniają poszczególne komponenty oprogramowania wykorzystujące informacje z wielu czujników. Można bowiem wyróżnić 4

główne zadania stojące przed systemem autonomicznej jazdy: lokalizacja i mapowanie, zrozumienie otaczającego świata, planowanie ścieżki i sterowanie pojazdem (jego kontrola). W kolejnych sekcjach zostaną omówione możliwości realizacji tych zadań.

2.3.1 Lokalizacja i mapowanie

Ustalenie lokalizacji pojazdu jest kluczowe w kontekście autonomicznej jazdy dowolnej platformy lądowej. Można wyróżnić dwa typy lokalizowania w przestrzeni: globalne oraz lokalne. Do globalnej nawigacji używa się zazwyczaj systemów GNSS takich jak GPS, GLONASS czy Galileo. Dzięki wykorzystaniu odbiorników wysokiej jakości możliwe jest uzyskanie pozycji względem świata z dokładnością od metra do nawet kilku centymetrów. Pomocna w tym zadaniu często okazuje się fuzja danych z czujnikami inercyjnymi IMU i czujnikami odometrycznymi. Dzięki niej można osiągnąć jeszcze większą dokładność lub zapewnić pracę systemu w przypadku braku dostępności sygnału GPS. Dobrym przykładem w tej materii jest robot Boss, który wygrał Urban Challenge. Zastosowany układ zapewniał dokładność na poziomie 30 cm, a w przypadku braku sygnału GPS na odcinku 1 km błąd pomiaru nie przekraczał 1 metra [76].

System globalnej lokalizacji jest niezbędny żeby określić przybliżoną pozycję, ale może być niewystarczający, aby uzyskać dokładność pozwalającą na wykonanie bardziej złożonych manewrów, np. utrzymywanie się na pasie ruchu czy omijanie przeszkód. Podobnie, gdy pojazd czy też robot operuje w pomieszczeniu zamkniętym lub na obszarze bez dostępu do nawigacji satelitarnej, np. w przypadku łazików marsjańskich działających na innej planecie. Wówczas niezbędny jest lokalny system pozycjonowania oraz nieodzowny element jakim jest proces tworzenia mapy otoczenia czyli mapowanie. Mapowanie odbywa się najczęściej z wykorzystaniem LiDAR-u lub kamery. Dzięki algorytmom SLAM (z ang. *Simultaneous Localization and Mapping*) możliwe jest zlokalizowanie się w chmurze punktów otoczenia uzyskanej z lidaru i dalsze mapowanie przestrzeni. Z kolei w przypadku kamer do utworzenia chmury punktów używane są punkty charakterystyczne na obrazie, które

wyszukiwane są z pomocą różnych algorytmów, a następnie śledzone na kolejnych klatkach obrazu. Znając parametry kamery i wykorzystując triangulację można uzyskać informację o głębi poszczególnych punktów. W przypadku standardowych kamer jest to zadanie bardziej złożone i podatne na błędy, jednak zastosowanie kamer 3D pozwala osiągnąć większą niezawodność dzięki stałej i znanej odległości pomiędzy dwoma klatkami obrazu obserwującymi dane punkty na obrazie. Dodatkowo skutecznym sposobem na poprawę działania SLAM-u jest wykorzystanie czujnika IMU. Fuzja danych z akcelerometru, żyroskopu, magnetometru i wizji bądź lidaru pozwala na uzyskanie dokładniejszych wyników [77, 78].

2.3.2 Zrozumienie otaczającego świata

Mając mapę otoczenia kolejnym krokiem dla pojazdu autonomicznego jest zrozumienie otaczającego świata. W tym celu wykorzystuje się najczęściej aż 3 typy czujników: LiDAR, kamery i RADAR. Dzięki symultanicznej pracy mogą skutecznie wykonywać trzy główne zadania pozwalające na percepcję otoczenia: segmentacja semantyczna, detekcja obiektów i ich klasyfikacja. Dodatkowo informacje o obiektach poruszających się wokół platformy mogą być uzupełnione o dodatkowe informacje jak ich prędkość czy nawet prawdopodobieństwo wykonania konkretnego ruchu dzięki predykcji zachowań tychże obiektów.

Segmentacja semantyczna

Zastosowanie segmentacji semantycznej stanowi dodatkowe źródło danych o otoczeniu, zapewniając lepsze zrozumienie sytuacji przez pojazd autonomiczny. Pozwala m.in. na wyznaczenie obszarów przejezdnych dla danego pojazdu czy wydzielenie stref, w których znajdują się określone obiekty. Segmentacja odbywa się zwykle z wykorzystaniem obrazu z kamery, choć używane mogą być również dane ze skanera laserowego. Realizacja segmentacji przebiega zazwyczaj przy zastosowaniu metod głębokiego nauczania. Szeroko wykorzystywane są konwolucyjne sieci neuronowe CNN (z ang. *Convolutional Neural Network*) [79–81]. Rezultatem segmentacji semantycznej jest zdolność do wyznaczenia kategorii każdego piksela na

obrazie [79]. Wśród wielu modeli segmentacji obrazu opartych na sieciach typu CNN popularnymi rozwiązaniami są np. SegNet [82] czy FCN (z ang. *Fully Convolutional Networks*) [83].

Wykrywanie i klasyfikacja obiektów

Równie istotna w kontekście systemów autonomicznej jazdy jest detekcja obiektów statycznych oraz ruchomych, która umożliwia skuteczne omijanie przeszkód czy planowanie bezpiecznego przejazdu. Wykrywanie obiektów można zrealizować wykorzystując różne typy sensorów, m.in. LiDAR, RADAR, ultradźwiękowe czujniki odległości czy kamery. Zważywszy na niewielki zasięg ultradźwięków, są one najczęściej wykorzystywane podczas poruszania się z małą prędkością. Pozwalają na wykrycie pojedynczego obiektu w polu detekcji sensora i uniknięcie z nim kontaktu. Pozostałe czujniki jak kamera, LiDAR czy RADAR oferują zazwyczaj znacznie większe zasięgi działania oraz rozdzielczość dostarczanych danych. Pozwala to na uzyskanie lepszej skuteczności i dokładności detekcji obiektów. Ponadto, więcej przetwarzanych danych umożliwia nie tylko wykrywanie obiektów, ale też ich klasyfikację, czyli określanie typu obiektu. W przypadku zastosowania danych wizyjnych najpopularniejsze i najskuteczniejsze rozwiązania do wykrywania i klasyfikacji obiektów oparte są na głębokim nauczaniu. Wśród najpopularniejszych rozwiązań są m.in. AlexNet [84], GoogLeNet [85] oraz YOLO [48]. Dodatkowo dzięki użyciu RADAR-u możliwe jest również uzyskanie precyzyjnej informacji o prędkości poruszającego się obiektu, co w zastosowaniach ściśle związanych z motoryzacją jest cenną informacją, szczególnie w kontekście systemów bezpieczeństwa.

Zastosowanie kilku typów czujników umożliwia przeprowadzenie fuzji danych z różnych sensorów. Pozwala to na zwiększenie niezawodności oraz skuteczności działania detekcji i klasyfikacji obiektów. Autorzy pracy [86] wykorzystali kamerę oraz LiDAR 2D do wykrywania, śledzenia i klasyfikowania obiektów, stosując fuzję danych wizyjnych oraz skan laserowy.

Odpowiednie wykrycie i zlokalizowanie przeszkód w przestrzeni pozwala na stworzenie tzw. siatki zajętości (z ang. *occupancy grid*). Każdy element siatki może za-

wierać szereg różnych informacji o otoczeniu, np. może określać czy dany obszar jest przejezdny na podstawie progu dopuszczalnej wysokości [17]. Te dane z kolei mogą zostać wykorzystane na dalsze etapie przy planowaniu trasy autonomicznego przejazdu.

Predykcja zachowań

Określanie rodzaju poszczególnych obiektów pozwala przewidywać ich zachowania i zrozumieć ich intencje. Oprogramowaniem dokonującym predykcji przyszłych działań wykrytych wokół pojazdu obiektów jest np. moduł Behaviour Prediction zaimplementowany przez firmę Waymo [24]. Podczas tworzenia tego oprogramowania naukowcy wykorzystali dane zebrane podczas testów na drogach publicznych. Miliony pokonanych kilometrów umożliwiły stworzenie szczegółowych modeli zachowań poszczególnych uczestników ruchu i określenie puli ich najbardziej prawdopodobnych zachowań.

2.3.3 Planowanie ścieżki

Skuteczne i przede wszystkim bezpieczne zaplanowanie przejazdu możliwe jest jedynie w przypadku, gdy znana jest lokalizacja pojazdu, określono pozycję punktu docelowego oraz przetworzono informacje o otoczeniu uzyskując niezbędne dane o obiektach wokół pojazdu. Podobnie jak w przypadku lokalizacji, można wyróżnić dwa typy planowania ścieżki: globalny i lokalny. Jeśli pojazd porusza się po zdefiniowanym obszarze to globalne planowanie polega na wyznaczeniu najlepszej trasy przejazdu. Przykładowo może się to odbywać z wykorzystaniem grafu [76] i zdefiniowanej funkcji kosztu, która promuje bezpieczniejsze rozwiązania i odrzuca te, które mogą okazać się groźne w skutkach. W przypadku autonomicznej jazdy po drogach publicznych takim niebezpiecznym manewrem określa się np. skręt w lewo na skrzyżowaniu [19]. Moduł globalnego planowania przejazdu uruchamiany jest przed rozpoczęciem podróży oraz za każdym razem, gdy niezbędna jest zmiana trasy, np. w przypadku zablokowanej drogi.

Mając wyznaczoną globalną ścieżkę do celu niezbędne jest przełożenie danych uzyskanych podczas skanowania otoczenia na lokalny przebieg drogi, który obejmuje wszelkie manewry zgodnie z percepcją obszaru wokół pojazdu. Dlatego też niezbędne jest również lokalne planowanie ścieżki, które obejmuje omijanie przeszkód czy też interakcję z innymi uczestnikami drogi. Opisana dwuetapowa metoda planowania drogi do celu została zaimplementowana przez pojazd Junior podczas Urban Challenge [19]. Pokonał on skomplikowaną trasę przejazdu oraz dowiódł, że takie podejście zapewnia bezpieczne i wydajne podążanie do wyznaczonego punktu. W przypadku pojazdu, który porusza się w nieznanym obszarze (z ang. *unstructured environment*) globalne wyznaczanie trasy ogranicza się do wyznaczania drogi w kierunku celu z uwzględnieniem obszarów już przebytych. W tym przypadku podczas wyznaczania drogi zazwyczaj wykorzystuje się algorytmy znajdowania najkrótszej ścieżki w grafie takie jak np. algorytm A* lub – będący jego rozszerzeniem – algorytm D*. Jako przykład można ponownie odwołać się do konkursu agencji DARPA, gdzie zwycięzca – samochód Boss – miał zaimplementowany algorytm o nazwie Anytime D* [76, 87], a druga ekipa z pojazdem Junior używała Hybrid A* [19]. W obu przypadkach algorytmy te były uruchamiane na obszarach nieznanych i nieokreślonych mapą takich jak np. parkingi. W przypadku jazdy poza drogami istnieją również inne rozwiązania, np. algorytm [88], który polega na wybraniu odpowiedniej trajektorii uwzględniającej model dynamiczny pojazdu, jego prędkość czy bezpieczeństwo danego przejazdu. Lokalne planowanie ścieżki powinno działać w trybie ciągłym, z każdym ruchem pojazdu optymalizując przejazd zgodnie z mapą otoczenia.

2.3.4 Sterowanie

Ostatnim zagadnieniem związanym z systemami autonomii jazdy jest kontrolowanie pojazdu. Polega na wysyłaniu takich komend sterowania do pojazdu, aby ten poruszał się zgodnie z zaplanowaną ścieżką. Możliwości prowadzenia pojazdu ograniczają się przede wszystkim do regulacji przyspieszenia, hamowania oraz kąta skrętu kół. Moduł sterowania jest zazwyczaj niezależny od systemu autonomii,

oczekuje na trajektorię przejazdu wyznaczoną podczas planowania drogi do celu. W podejściu polegającym na wyborze najlepszej trajektorii wyznaczonej podczas planowania [89] niezbędne jest właściwe wykonanie przejazdu wykorzystując opisywany moduł sterowania. Można zrealizować to w sposób bezpośredni, próbując za wszelką cenę utrzymywać zadaną ścieżkę, uwzględniając model fizyczny pojazdu. Jednak bardziej słusznym podejściem wydaje się fragmentaryczne traktowanie zadanej trasy przejazdu, jej podział oraz dalsze łączenie trajektorii (z ang. *trajectory blending*) pozwalające na wygładzenie trasy i płynne zachowanie pojazdu. Bezpieczne wykonanie przejazdu zapewnia zastosowanie podczas planowania ścieżki odpowiedniego modelu dynamicznego danego pojazdu, który zawiera funkcję maksymalnej prędkości przy danym skręcie kół i określa maksymalne przyspieszenie, zwalnianie czy opóźnienie w sterowaniu [76].

Nieco odmiennym podejściem do sterowania pojazdem jest zastosowanie tzw. *end-to-end deep learning*. W pracy przedstawionej przez naukowców *Nvidia* [90] została przedstawiona wielowarstwowa sieć konwolucyjna CNN, która jedynie na podstawie obrazu z kamery potrafi dostosować kąt obrotu kierownicy tak, aby utrzymać się na pasie ruchu lub odcinku drogi. Sam proces nauczania sieci neuronowej również odbywa się jedynie poprzez przekazywanie obrazu, więc w pewnym sensie nauka i działanie tego rozwiązania odbywa się w sposób podobny do człowieka. Jednak metoda ta ze względu na swoje ograniczenia (w większości przypadków nie jest w stanie bezpiecznie ominąć przeszkód, a nawet ich wykryć), może być stosowana jedynie w ograniczonym zakresie.

Rozdział 3

Narzędzia symulacji systemów autonomicznych

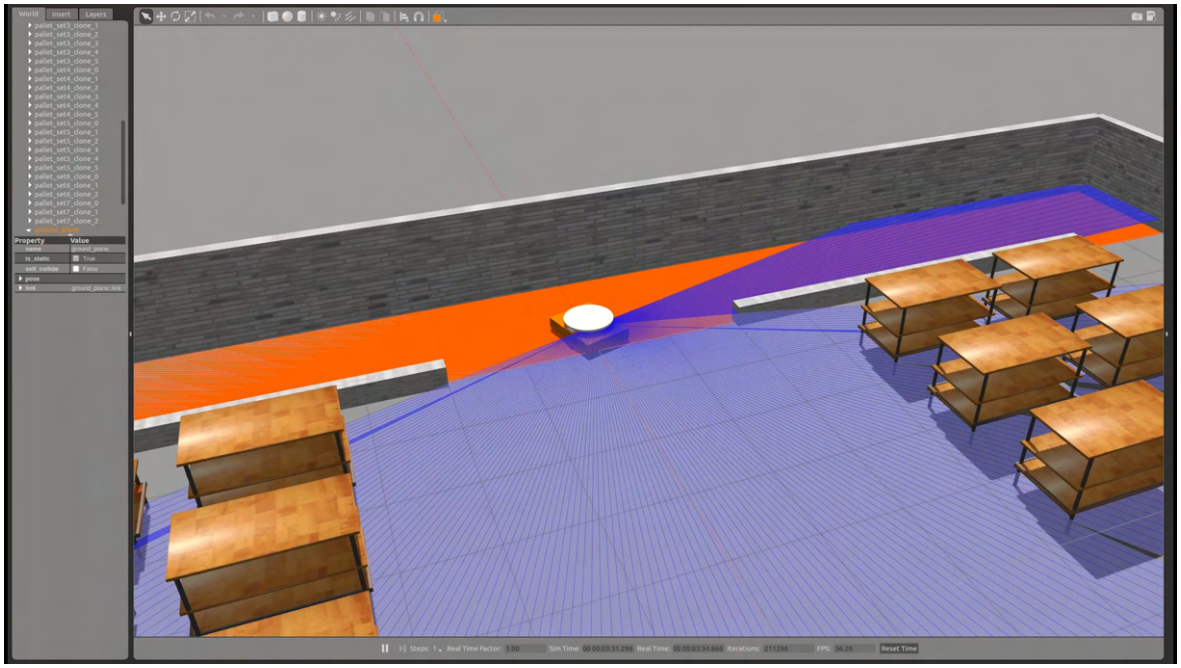
W rozdziale przedstawiono przegląd środowisk symulacji wspomagających rozwój systemów autonomicznych. Ponadto omówiono narzędzia, które wykorzystano do opracowania kompleksowego symulatora pozwalającego na wykonanie prac prowadzących do przygotowania bezzałogowego pojazdu lądowego operującego autonomicznie. W ramach narzędzi przedstawiono elementy umożliwiające zarówno stworzenie modelu pojazdu w symulatorze, jak i czujników, których zachowanie i oddziaływanie z otoczeniem zaimplementowano.

3.1 Przegląd systemów symulacji

Jak wspomniano w sekcji 2.1, środowiska symulacyjne jako narzędzia przy opracowywaniu systemów autonomicznej jazdy pojawiały się od dawna. Większość rozwiązań w głównej mierze była początkowo przeznaczona do przetestowania idei i konceptów podczas badań nad zagadnieniami z obszaru robotyki, ewentualnie, w późniejszym czasie, do wstępnej weryfikacji działania poszczególnych algorytmów przed wdrożeniem ich na finalny pojazd.

Początkowo stosowano narzędzia wizualizacyjne i symulacyjne działające w try-

bie 2D. Środowisko Player oraz Stage [18] zapewniało, odpowiednio, interfejs komunikacyjny oraz warstwę graficzną, w której możliwe było wykonywanie symulacji kilku modeli sensorów, w tym czujników ultradźwiękowych, lidarów 2d czy odometrii, co sprawiało, że była to ówcześnie szeroko stosowana platforma do badań i opracowywania kontrolerów sterowania robotów różnego typu, w tym współpracujących ze sobą w ramach jednego systemu. Rozszerzeniem tego projektu było Gazebo [91] działające w środowisku 3D oraz oferujące symulację parametrów fizyki takich jak tarcie czy kolizje między obiektami. Dzięki integracji z narzędziami wspierającymi programowanie robotów, prostej konfiguracji i niewielkiej ilości czasu potrzebnej do uruchomienia prostego środowiska symulacji, środowisko to stało się standardem wśród naukowców i osób związanych z robotyką i badaniem tego typu systemów. Rozwiązanie to rozwijane jest do dziś i stanowi świetne narzędzie testów algorytmów sterowania, planowania ścieżki czy wykrywania przeszkód dla robotów o prostej budowie z kilkoma podstawowymi sensorami, takimi jak ultradźwiękowe czujniki odległości czy lidary 2D. Na rys. 3.1 przedstawiono przykładową symulację magazynu wraz z robotem wyposażonym w skaner laserowy.



Rysunek 3.1: Symulacja magazynu w środowisku Gazebo [92].

Gazebo charakteryzuje się również kilkoma wadami. Przede wszystkim wykorzy-

stuje on przestarzały silnik renderujący Ogre [93], który obecnie odstaje od konkurencyjnych rozwiązań i nie oferuje zaawansowanych funkcjonalności, dzięki którym możliwe byłoby osiągnięcie większego realizmu odwzorowania rzeczywistości. Ponadto budowanie rzeczywistych i złożonych światów symulacji oraz naturalnych, rozbudowanych modeli robotów ogranicza sposób reprezentacji modeli w Gazebo, gdzie są one zazwyczaj opisywane za pośrednictwem plików XML, które ograniczają możliwości wizualne i fizyczne [94].

Na przestrzeni lat opracowano wiele innych środowisk służących do symulacji działania robotów i pojazdów autonomicznych. Wśród nich powstało kilka mniejszych projektów opracowanych z myślą o DARPA Urban Challenge [95]. Przygotowywano również bardziej uniwersalne rozwiązania, takie jak V-REP, Webots, Microsoft Robotics Developer Studio czy USARSim [96]. Są to jednak obecnie rozwiązania niszowe, o ograniczonej funkcjonalności, w wielu przypadkach niewspierane już przez twórców.

Jednak w ostatnim czasie dynamiczny rozwój autonomicznych samochodów sprawił, że pojawiają się nowe rozwiązania symulacyjne, wśród których najbardziej rozbudowany wydaje się symulator Carla [6]. Oferuje symulacje większości sensorów mających zastosowanie w systemach autonomicznych takich jak kamery, lidary, radary, IMU i GNSS. Dodatkowo zapewnia symulację działania algorytmów percepcji otoczenia dostarczając informacje o obiektach w otoczeniu symulowanego pojazdu, segmentacji semantycznej obrazu z wirtualnej kamery czy informacji o przekroczeniu linii na jezdni. Na rys. 3.2 został przedstawiony zrzut ekranu ukazujący podgląd środowiska w symulatorze, ruch drogowy oraz podgląd danych z syntetycznych sensorów: kamery RGB, kamery głębi oraz kamery symulującej działanie algorytmu segmentacji semantycznej obrazu.

Symulator Carla udostępnia ponadto przyjazny interfejs programistyczny pozwalający modyfikować otoczenie i panujące warunki w symulacji, a sieć dróg może zostać łatwo zaimportowana wraz z całym oznakowaniem i infrastrukturą drogową. Natomiast zorientowanie na środowiska miejskie oraz pojazdy poruszające się po drogach publicznych sprawiło, że implementacja tego symulatora na potrzeby jazdy

Rysunek 3.2: Symulator Carla wraz z podglądem wirtualnych kamer głębi i segmentacji [97].

w terenie jest mocno ograniczona. Ponadto sposób budowy modeli pojazdu wymusza stosowanie geometrii Ackermanna [98] co mocno zawęża możliwość symulacji pojazdów (tylko 4-kołowe pojazdy ze skrętną przednią osią). Jest to problematyczne w przypadku bezzałogowych platform lądowych, które z reguły wykorzystują tzw. sterowanie różnicowe (burtowe) (z ang. differential drive).

Konkurencyjnym rozwiązaniem względem Carli jest Microsoft AirSim [28]. Jako zaletę należy wymienić możliwości symulacji czterowirnikowego drona typu UAV co znacznie poszerza zakres prac, które można przeprowadzić w ramach symulatora. Niestety w przypadku wykorzystania tego symulatora, pojazd naziemny ograniczony jest do jednego modelu samochodu i nie oferuje możliwości użycia pojazdu o innym układzie sterowania. Co więcej symulacja sensorów jest zdecydowanie mniej zaawansowana, a w przypadku kamer i lidarów nie pozwala na symulowanie szumów lub zniekształceń charakterystycznych dla konkretnych czujników.

Trzecim środowiskiem skupionym wokół symulacji środowiska miejskiego i samochodów osobowych jest symulator LG SVL. Był to projekt oparty na licencji komercyjnej, nie jest jednak już wspierany przez producenta. Oferował większość funkcjonalności symulatora Carla, wyróżniając się przy tym bardzo realistyczną oprawą graficzną, dzięki zaawansowanym potokom renderowania przez silnik graficzny Unity. Zapewniał również rozbudowane efekty pogodowe oraz wierne odwzo-

rowanie cyklu dnia i nocy.

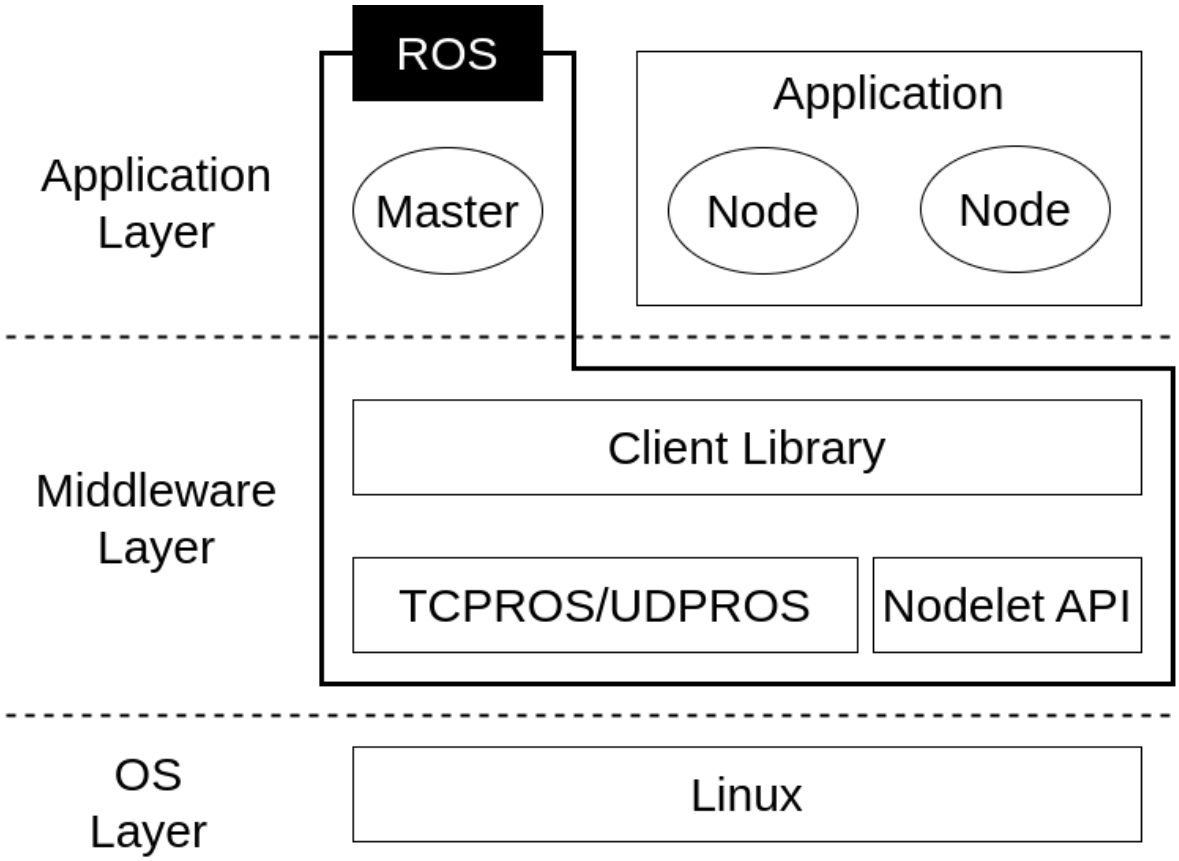
W literaturze można znaleźć także wzmianki o pracach skupiających się na pojedynczym zagadnieniu takim jak np. symulacja działania lidaru w LIDARsim [5] czy też Blensor działający w oparciu na programie do modelowania 3D Blender. Charakteryzują się one wysoką jakością symulowanych danych, jednak ich działanie jest ukierunkowane tylko w jednym konkretnym celu, np. do testów segmentacji danych, czy weryfikacji działania algorytmów sztucznej inteligencji w określonych, powtarzalnych warunkach.

Na przestrzeni lat rozwijano szereg innych rozwiązań tworzonych z myślą o testach oraz symulacji pojazdów i algorytmów autonomicznych [99]. Inżynierskie oprogramowanie Matlab wraz z pakietem Simulink oferuje zbiór narzędzi służących do projektowania, testowania i symulacji aktywnych systemów bezpieczeństwa ADAS dla samochodów osobowych. Automated Driving Toolbox umożliwia symulowanie działania sensorów i wykonywanie testów na zdefiniowanych scenariuszach drogowych. Oferuje również pomocnicze aplikacje m.in. do tworzenia etykiet dla zbiorów danych uczących dla sieci neuronowych lub wizualizację danych sensorycznych. Podobne funkcjonalności oferuje PreScan, który umożliwia przeprowadzenie testów typu Hardware In the Loop (HIL) symulując rzeczywiste sygnały sensoryczne, testując w ten sposób główne jednostki sterujące pojazdów autonomicznych lub wyposażonych w systemy ADAS. Innym rodzajem środowiska symulacyjnego jest CarSim, który oferuje zaawansowaną symulację właściwości dynamicznych pojazdu i doskonałe odwzorowanie jego parametrów fizycznych i wpływu czynników zewnętrznych. Zapewnia przy tym integrację z pozostałymi narzędziami rozbudowując ich funkcjonalności. Są to jednak przykłady komercyjnych narzędzi produkcyjnych wykorzystywanych głównie w branży motoryzacyjnej na etapie wdrażania i testowania finalnych rozwiązań. Nie oferują one przy tym rozwiązań otwartoźródłowych symulatorów wykorzystywanych przede wszystkim w pracach badawczych i rozwojowych nad systemami autonomii jazdy.

3.2 System ROS

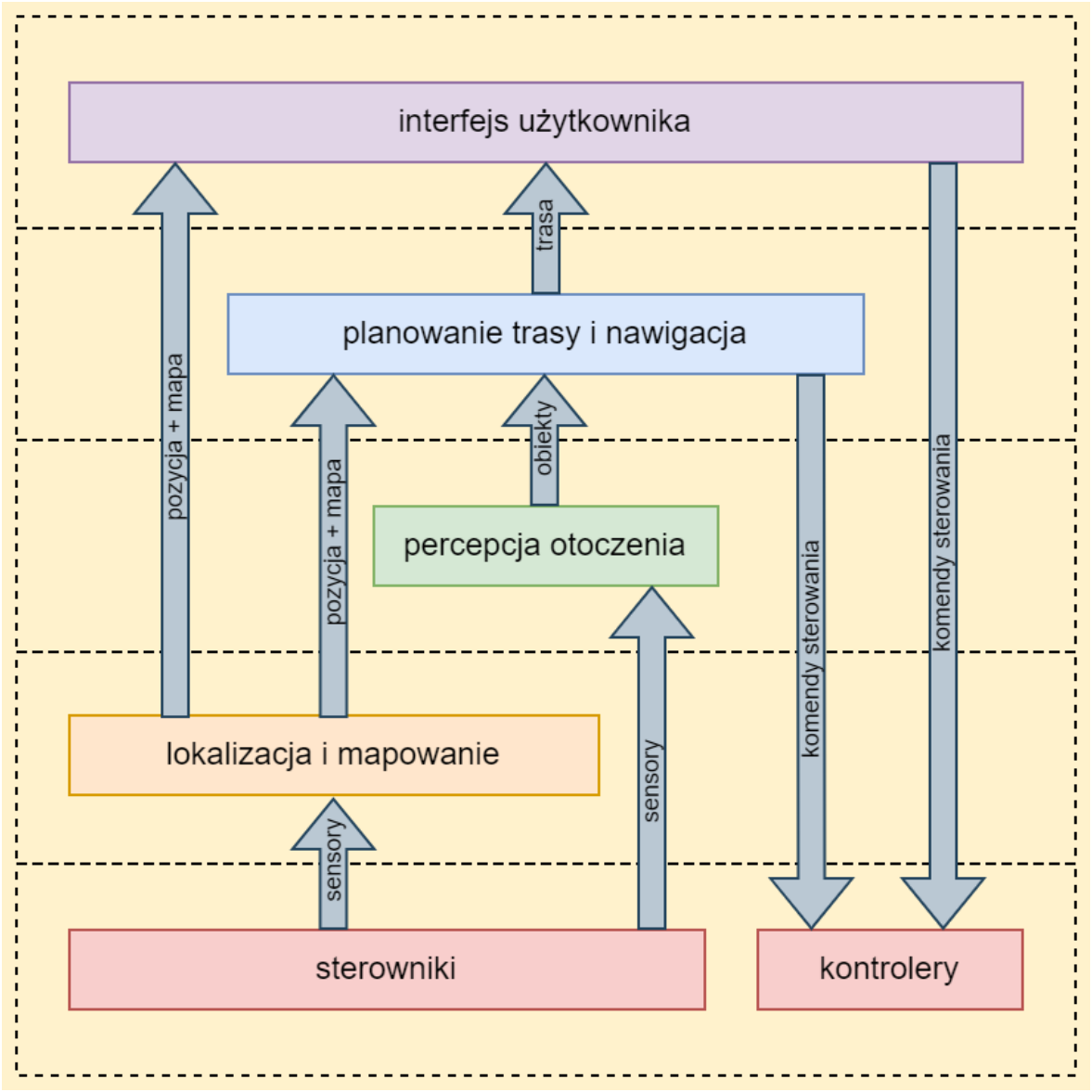
ROS (z ang. *Robot Operating System*) [100] to zestaw narzędzi i bibliotek programistycznych, których celem jest ułatwienie tworzenia oprogramowania dla robotów. Nie jest to system operacyjny sensu stricto, bardziej poprawnym określeniem zdaje się tzw. oprogramowanie pośredniczące (ang. *middleware*) [101], gdyż umożliwia komunikację poszczególnym podsystemom robota i zapewnia definicje podstawowych wiadomości wymienianych pomiędzy urządzeniami i komponentami systemu. Oprogramowanie to stanowi często szkielet architektury pojazdów autonomicznych. Opracowywane w ramach niniejszej rozprawy środowisko symulacyjne zapewnia kompatybilność z ROS oferując tym samym większą uniwersalność tego rozwiązania. Celem przybliżenia ROS należy wspomnieć o architekturze i jego głównych składowych. Jedną z najważniejszych idei stojących za ROS jest rozbicie funkcjonalności projektowanego systemu na wiele małych programów tzw. węzłów (z ang. *nodes*), komunikujących się między sobą poprzez wymianę wiadomości na dedykowanych kanałach zwanych tematami (z ang. *topics*). Jest to klasyczna implementacja wzorca publikuj/subskrybuj (z ang. *publish/subscribe*) z głównym węzłem *Master*, do którego każdy nowy węzeł zwraca się z informacją jakie tematy publikuje, a jakie subskrybuje. Dodatkowo ROS oferuje możliwość wywoływania tzw. serwisów na zasadzie bezpośrednich zapytań do danego węzła, w których dane zarówno pytania, jak i odpowiedzi są predefiniowane i umożliwiają natychmiastowe uzyskanie informacji np. o stanie danego węzła bez potrzeby subskrybowania określonego tematu i oczekiwania na otrzymanie pakietu. Ponadto w ramach dostępnego interfejsu programistycznego Nodelet API możliwe jest tworzenie programów w ramach tzw. nodeletów, które są dynamicznymi bibliotekami ładowanymi podczas działania systemu ROS i zapewniającą tzw. komunikację *zero-copy* pomiędzy sobą umożliwiając przekazywanie danych bez kopiowania. Rys. 3.3 przedstawia architekturę ROS wraz z wyróżnieniem 3 warstw, na których operuje oprogramowanie.

Ważnym elementem każdego systemu robotycznego są połączone ze sobą układy współrzędnych w strukturze drzewiastej. Każdy sensor, bądź aktuator działa w obrębie własnego układu w określonym czasie i np. w celu poruszenia manipulatorem



Rysunek 3.3: Architektura systemu ROS [102].

czy zmierzenia odległości przez czujnik laserowy, dany układ współrzędnych musi być przeliczany i synchronizowany w czasie z pozostałymi. W ROS dostępne jest natywne wsparcie dla tego typu operacji, zagwarantowane przez kilka elementów systemu. Pierwszym z nich jest sformalizowany sposób opisu modelu robota w postaci pliku URDF [103], w którym definiowane są m.in. połączenia (z ang. *links*) pomiędzy komponentami robota, ich położenie względem siebie czy dodatkowe parametry wizualizacyjne, co pozwala utworzyć wspomnianą wcześniej strukturę drzewiastą połączonych ze sobą elementów nazywaną drzewem transformacji (z ang. *transform tree*). Ponadto dane operujące w konkretnym układzie współrzędnych zawierają w nagłówku wiadomości pole *link*, w którym zawarta jest nazwa liścia w drzewie. W nagłówku wiadomości znajduje się również znacznik czasowy wskazujący czas pomiaru, który pozwala na poprawną interpretację danych jak np. wyznaczenie położenia robota względem mapy w konkretnym punkcie w czasie. Dodatkowo istnieje



Rysunek 3.4: Schemat ideowy architektury systemu autonomicznego opartego na ROS.

zbiór funkcji umożliwiających przeliczanie danych z jednego układu współrzędnych na drugi, co ułatwia w rezultacie prace nad systemem.

Standardowa konfiguracja ROS sterująca pracą platformy robotycznej, której celem jest jazda w nieznanym terenie z punktu A do punktu B z ominięciem przeszkód obejmuje szereg algorytmów i programów w postaci węzłów ROS. Na rys. 3.4 przedstawiono warstwową strukturę systemu opartego na ROS. Najniższą war-

stwę oprogramowania stanowią węzły bezpośrednio komunikujące się ze sprzętem, a więc wszelkie sterowniki pobierające dane z sensorów i publikujące je na odpowiednich tematach, a także kontrolery elementów wykonawczych takich jak silniki, które subskrybują komendy sterowania wyliczane w dalszych etapach przetwarzania danych. To właśnie na tym poziomie powinien operować symulator zapewniając niezmienioną architekturę systemu względem fizycznej platformy oraz umożliwiając przeprowadzanie testów algorytmów ze wszystkich wyższych poziomów. Kolejne warstwy korzystając z danych sensorycznych potrafią wyznaczyć mapę otoczenia, ustalić swoją pozycję czy zrozumieć otaczający świat i pokierować do celu wysyłając komendy sterowania do najniższych warstw oprogramowania.

3.3 Środowisko Unity

Opracowanie systemu symulacji wymaga zaprojektowania i implementacji wielu komponentów oprogramowania. Wyświetlanie grafiki zwykle przebiega z wykorzystaniem niskopoziomowego interfejsu programistycznego API, który umożliwia rysowanie i manipulację grafiką z wykorzystaniem funkcji operujących na warstwie sprzętowej takiej jak karta graficzna. Do najpopularniejszych interfejsów tego typu zalicza się OpenGL [104], DirectX [105] czy Vulkan [106].

Zastosowanie niskopoziomowego API pozwala na przygotowanie środowiska graficznego czy zrealizowanie dowolnej symulacji zarówno w technice 2D, jak i 3D. Jednak tworzenie bardziej złożonych systemów wymaga stworzenia specjalistycznego silnika graficznego realizującego nie tylko podstawowe funkcje służące do prezentacji grafiki na ekranie, ale też umożliwiającego wykonywanie bardziej złożonych operacji takich jak np. wczytywanie modeli 3D, wizualizacja wirtualnej sceny, modelowanie światła czy stosowanie programów cieniujących. Do tego typu silników znajdujących zastosowanie przede wszystkim w grach komputerowych należy zaliczyć głównie Unreal Engine [107] oraz Unity [108]. Te dwa popularne systemy stanowią bardzo dobre środowisko do implementacji symulatorów i symulacji komputerowych działających w czasie rzeczywistym. Dzięki wydajnym mechanizmom optymalizacyjnym umożliwiają wierne odwzorowanie świata rzeczywistego przy zachowaniu

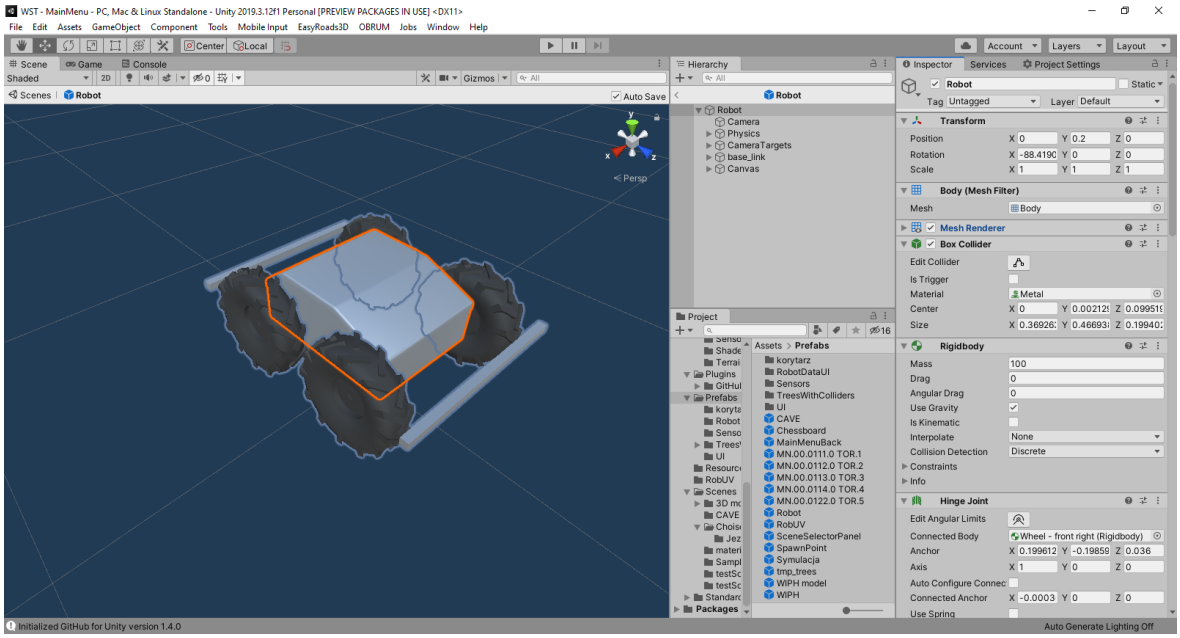
rozsądnych wymagań systemowych oraz czasu symulacji zbliżonego do czasu rzeczywistego. Pierwszy z nich znalazł zastosowanie m.in. w omawianym w sekcji 3.1 symulatorze Carla, który dzięki wysokiej jakości grafiki pozwala wiernie odwzorować rzeczywistość widzianą z perspektywy samochodu autonomicznego. Unity, nie ustępując wiele swojemu konkurentowi możliwościami graficznymi, zapewnia intuicyjny interfejs oraz łatwiejszy interfejs programistyczny oparty na języku programowania C#. Ponadto umożliwia łatwy wybór niskopoziomego interfejsu DirectX, OpenGL lub Vulkan. Istotną zaletą Unity jest również rozbudowany edytor graficzny umożliwiający wizualizację wirtualnej sceny symulacji, konfigurowanie parametrów w intuicyjny sposób oraz podgląd wprowadzanych zmian na bieżąco bez potrzeby budowania czy kompilowania całej aplikacji.

Poza graficzną warstwą i możliwościami poszczególnych silników istotną rolę odgrywa również symulacja fizyki. Zastosowanie jednego z popularnych silników gier 3D zapewnia łatwiejszy dostęp do nadawania fizycznych cech wirtualnym obiektom oraz ich interakcji z otoczeniem. Unity wykorzystuje opracowany przez firmę Nvidia silnik fizyki PhysX [109], który cechuje się wysokim poziomem odwzorowania świata rzeczywistego przy jednoczesnym zachowaniu wydajności pozwalającej bez trudu symulować oddziaływania fizyczne w czasie rzeczywistym.

W ramach doktoratu zdecydowano się na zastosowanie Unity do zbudowania Wirtualnego Systemu Testującego (WST) jako systemu, który zapewni środowisko do budowania wirtualnych map, modeli pojazdów i konfiguracji systemu. Dodatkowo umożliwi implementację modeli symulacyjnych sensorów wykorzystywanych przez pojazdy autonomiczne oraz pozwoli na zbudowanie systemu komunikacji ze środowiskiem ROS wykorzystywanym w ramach platform bezzałogowych, które powinny zostać zasymulowane. Zadanie zostało zrealizowane co opisano w rozdziale 4, w którym poruszono kwestie implementacji poszczególnych elementów WST, w szczególności wirtualnych modeli czujników czy symulacji otoczenia i poszczególnych pojazdów.

Na rys. 3.5 przedstawiono przykładowy zrzut ekranu edytora Unity w środowisku Windows z wczytanym projektem WST, wraz z otwartym modelem małej

platformy autonomicznej. Dzięki graficznemu interfejsowi możliwa jest wizualizacja utworzonego wcześniej modelu pojazdu, konfiguracja parametrów fizycznych platformy oraz ustawienie wirtualnych wiązań między poszczególnymi elementami takimi jak np. koła i nadwozie.



Rysunek 3.5: Edytor Unity wraz z modelem małej, bezzałogowej platformy autonomicznej.

Zastosowanie środowiska Unity umożliwiło ponadto zachowanie wieloplatformowości rozwiązania. Na obecnym etapie, docelowym systemem uruchomieniowym WST jest system Windows oraz biblioteka DirectX, jednak Unity umożliwia w łatwy sposób uruchomienie symulatora również w systemie Linux z wykorzystaniem OpenGL. Jest to szczególnie istotne ze względu na fakt, że system ROS działa w środowiskach UNIX. W obecnej konfiguracji nie stanowi to problemu, gdyż WST uruchamiane jest sieciowo na osobnym komputerze, a ROS działa na wirtualnej maszynie działającej w środowisku Windows. Jednakże potencjalne wymaganie uruchomienia systemu ROS natywnie na komputerze Linux nie powinno stanowić problemu również dla działania WST na tej samej maszynie, zapewniając przy tym komunikację pomiędzy algorytmami autonomii, a środowiskiem symulacji bez znaczących opóźnień.

Rozdział 4

Wirtualny System Testujący

Rozdział ten opisuje zastosowane w ramach pracy doktorskiej metody, służące opracowaniu środowiska symulacyjnego wykorzystywanego celem oceny zasadności stosowania symulacji pojazdów i sensorów przeznaczonych dla systemów autonomii. Przedstawiono architekturę systemu zapewniającą zgodność danych syntetycznych z systemem autonomii oraz omówiono sposoby generowania wirtualnego świata wraz z modelami pojazdów sterowanych autonomicznie. Poruszono również kwestie implementacji poszczególnych modeli symulacyjnych czujników oraz interfejsu komunikacyjnego wraz z analizą dotyczącą korzyści zastosowania autorskiego rozwiązania.

4.1 Architektura systemu symulacji

Zgodnie z celem doktoratu opracowano w jego ramach Wirtualny System Testujący (WST), czyli środowisko testowe 3D oferujące szereg funkcjonalności zaprojektowanych z myślą o tworzeniu i rozwijaniu systemów autonomicznej jazdy bezzałogowych platform lądowych operujących zarówno na zewnątrz, jak i wewnątrz budynków. System ten powinien spełniać następujące wymagania:

- realistyczne modele czujników - uwzględnienie najważniejszych błędów i zniekształceń danych sensorycznych oraz format danych identyczny z rzeczywi-

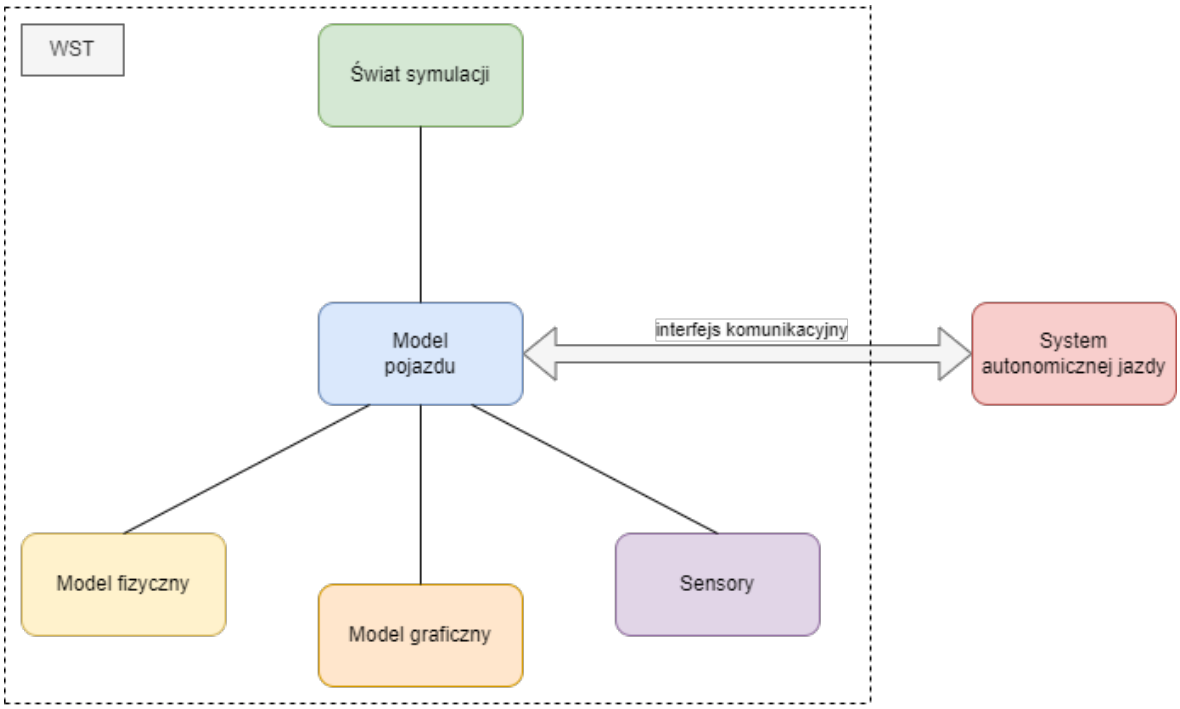
stym,

- brak ograniczeń w zakresie rodzaju pojazdu - symulacja dowolnej wielkości pojazdu o dowolnym napędzie i parametrach dynamiki ruchu,
- swoboda w modelowaniu świata - możliwość tworzenia i importowania modeli świata rzeczywistego, zarówno obszarów otwartych, jak i pomieszczeń,
- odwzorowanie parametrów fizycznych - masa, tarcie, współczynniki drgań i tłumienia itp.,
- wysoki stopień realizmu warstwy graficznej.

Takie podejście wymagało przemyślanej architektury systemu symulacji, badań sensorów i odpowiedniej implementacji poszczególnych komponentów. Istotną kwestią było również zapewnienie kompatybilności i interoperacyjności syntetycznych danych z przygotowanymi algorytmami. Elementy te poruszone zostaną w kolejnych sekcjach niniejszej pracy.

Architektura WST została w uproszczony sposób przedstawiona na rys. 4.1. Głównym elementem jest tutaj model pojazdu. Operuje on w środowisku symulacji 3D, gdzie jego poszczególne komponenty oddziałują ze sobą symulując działania rzeczywistej platformy wyposażonej w szereg czujników dostarczających danych w określonym formacie. Istotną rolę odgrywa również interfejs, który pozwala na dwukierunkową komunikację z systemem zarządzającym autonomiczną jazdą, odpowiednimi algorytmami i sterowaniem platformą.

Testowanie WST odbywało się z wykorzystaniem Modułowego Systemu Autonomii MSA [110]. Symulator został opracowany z myślą o integracji z systemem autonomii w celu zintensyfikowania prac nad MSA w przypadku braku dostępu do rzeczywistego pojazdu. MSA oferuje autonomiczny przejazd dowolnego pojazdu z punktu A do punktu B działając zarówno w pomieszczeniu, jak i w terenie otwartym. Dzięki modułowej architekturze systemu osiągnięto swobodę w doborze czujników w zależności od rodzaju pojazdu oraz jego potrzeb operacyjnych. Wymusza to jednak ponoszenie znacznego wysiłku przy konfiguracji wielu wersji algorytmów



Rysunek 4.1: Uproszczony schemat architektury Wirtualnego Systemu Testującego.

pełniących tę samą rolę, lecz z wykorzystaniem różnych sensorów. Ponadto wybranie konkretnego zestawu czujników, ich liczby na pojeździe czy dokładnej specyfikacji wymaga przeprowadzenia szeregu testów weryfikujących ich skuteczność w danej konfiguracji. Zadanie to zostało w głównej mierze przerzucone na barki WST, gdzie zmiana charakterystyki czy konfiguracji sensorów dla danego pojazdu nie stanowi problemu przynosząc również korzyści biznesowe polegające na oszczędności czasu i ograniczeniu kosztów.

4.2 Symulacja otoczenia

Modelowaniem 3D nazywamy proces tworzenia matematycznej reprezentacji trójwymiarowego kształtu obiektu [111]. Dzięki temu możliwe jest przedstawianie elementów świata rzeczywistego w środowisku wirtualnym. Istnieje wiele metod modelowania 3D, jednak w kontekście odwzorowywania fizycznych obiektów najważniejsze wydają się:

- projektowanie komputerowe,
- skanowanie laserowe 3D,
- fotogrametria.

Projektowanie komputerowe polega na ręcznym tworzeniu modelu za pomocą dedykowanego oprogramowania. Manipulacja podstawowymi bryłami geometrycznymi, dodawanie bądź usuwanie wierzchołków, tworzenie płaszczyzn i krzywizn to podstawowe operacje prowadzące do skonstruowania wiernej, choć charakteryzującej się pewnymi uproszczeniami, kopii realnego obiektu.

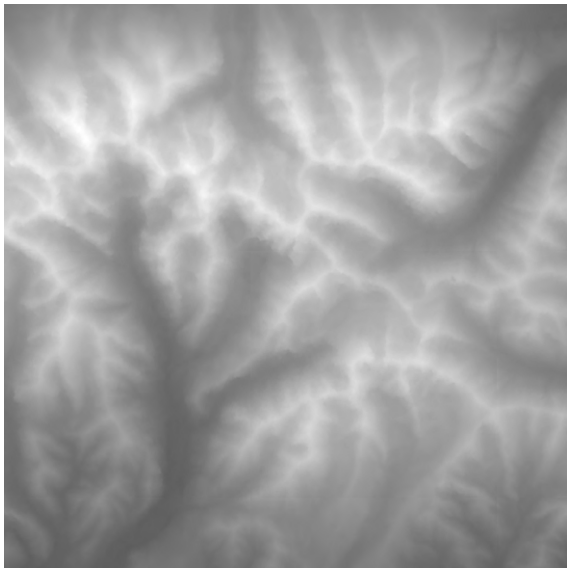
Skanowanie 3D wymaga z kolei użycia zewnętrznego narzędzia jakim jest skaner laserowy. Urządzenie to dzięki wiązce lasera potrafi utworzyć chmurę punktów obejmującą całą powierzchnię skanowanego obiektu. W roli skanera może posłużyć także lidar 3D, którego zasada działania nie różni się w znaczący sposób od skanerów przeznaczonych do modelowania. Następnie wykonywany jest proces teselacji polegający na utworzeniu siatki wielokątów poprzez odpowiednie łączenie punktów. Tworzone wielokąty są zazwyczaj trójkątami, którego połączone wierzchołki tworzą szkielet obiektu [112].

W fotogrametrii rolę skanera laserowego przejmuje aparat fotograficzny. Wykonując sekwencję zdjęć danego obiektu możliwe jest znalezienie punktów charakterystycznych, które następnie, skorelowane względem kolejnych ujęć, umożliwiają wyznaczenie ich pozycji w przestrzeni względem siebie co pozwala osiągnąć podobny efekt, jak w przypadku zastosowania skanera laserowego.

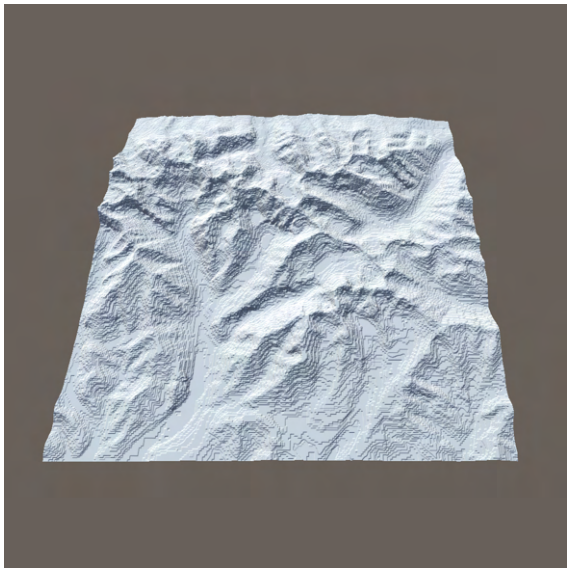
Każda z powyższych metod może posłużyć do skonstruowania świata symulacji WST, który może być modelem przestrzennym pomieszczenia lub większego terenu o dowolnym profilu powierzchni. W przypadku operowania wewnątrz budynku naturalnym rozwiązaniem jest przygotowanie modelu 3D pomieszczenia w dowolnym kompatybilnym formacie. Dodatkowe teksturowanie modelu, czyli naniesienie kolorów i tekstur w celu imitacji prawdziwego wyglądu, pozwala na zwiększenie realizmu symulacji, głównie w przypadku wykorzystania wirtualnych kamer jako danych wejściowych dla systemu autonomii. W pokoju bądź innym wnętrzu znajduje się wiele

obiektów, których geometria odgrywa istotną rolę w kwestii odwzorowania świata rzeczywistego, w związku z czym modelowanie takiego otoczenia na potrzeby symulacji powinno dotyczyć jak największej liczby tego typu elementów, przy zapewnieniu ich wysokiej szczegółowości. Dowolna ze wspomnianych wcześniej metod pozwala na przygotowanie modelu 3D pomieszczenia, które może posłużyć za wirtualne środowisko laboratoryjne, zarówno przedstawiające fikcyjne pomieszczenie, jak i fizyczny obszar, co może być przydatne np. w przypadku wykonywania badań porównawczych.

Podobne mechanizmy dotyczą modelowania świata zewnętrznego. Tak jak poprzednio, korzystając ze standardowych metod, można stworzyć bazę obiektów, które następnie mogą zostać wykorzystane w celu stworzenia wirtualnej mapy przedstawiającej dowolny obszar na świecie, bądź nieistniejące terytorium. Różnicą jest jednak sam proces tworzenia wirtualnego terenu, gdyż oprócz przygotowania modelu 3D można posłużyć się innymi metodami. Najefektywniejszym sposobem na zaprojektowanie dużego, wirtualnego obszaru, który odwzorowywałby rzeczywisty profil terenu jest zastosowanie tzw. map wysokościowych (z ang. *heightmap*). Są to obrazy rastrowe będące dyskretną reprezentacją wysokości terenu przedstawioną w postaci siatki o danej rozdzielczości, której każdy piksel odpowiada rzeczywistej wysokości, często przeskalowanej liniowo. Na rys. 4.2a przedstawiono fragment obszaru na terytorium Polski o wymiarach 15 km na 15 km w postaci mapy wysokości, na której można dostrzec wszelkie nierówności powierzchni, góry i doliny. Bazując na tych danych można wygenerować wirtualny model terenu co przedstawia rys. 4.2b. Jeżeli wiarygodność z rzeczywistym obszarem nie jest przedmiotem symulacji, stworzenie mapy wysokościowej możliwe jest w takim przypadku do zrealizowania za pomocą narzędzi do manipulacji wirtualnym terenem. Obejmują one kilka funkcji pozwalających na zmianę wysokości zgodnie z intencją użytkownika: m.in. obniżanie lub podnoszenie terenu, tworzenie wgłębień czy wygładzanie powierzchni. Ostatnim elementem zagadnienia dotyczącego odwzorowania otoczenia jest nakładanie tekstury, które może przebiegać na dwa sposoby. Po pierwsze, do dyspozycji są narzędzia umożliwiające zabarwianie terenu poprzez łączenie obrazów ilustrujące trawę, ziemię, pola, łąki czy pustynie w sposób manualny lub zautomatyzowany



(a) Mapa wysokościowa.



(b) Teren 3D powstały bazując na mapie wysokościowej.

(c) Fragment terenu z nałożoną teksturą.

Rysunek 4.2: Teren w środowisku symulacyjnym WST.

(rys. 4.2c). Drugą metodą, której celem powinno być zapewnienie wysokiego realizmu, jest dostarczenie jednolitej tekstury terenu opartej na zdjęciach satelitarnych czy lotniczych o odpowiednio wysokiej rozdzielczości, która jednak może okazać się

niewystarczająca, aby osiągnąć odpowiednio wysoki poziom odwzorowania.

Ważnym elementem wpływającym na odbiór grafiki trójwymiarowej jest oświetlenie. Generuje ono cienie czy odbicia i urealnia wirtualną scenę. Manipulacja barwą i intensywnością światła, szczególnie w przypadku terenów otwartych, jest kluczowa przy nadaniu poczucia zmiany pory dnia i wprowadzeniu w symulacji cyklu dnia i nocy, co w przypadkach niektórych pojazdów autonomicznych może odgrywać pewną rolę. Główny udział podczas symulacji oświetlenia przypada programom cieniującym, które mogą być uruchamiane w WST, jednak ich rola może być również inna. Zastosowanie tzw. cieniowania wierzchołkowego (z ang. *vertex shader*) pozwala na manipulację wierzchołkami w przestrzeni 3D, co może być z kolei wykorzystane np. podczas symulacji wiatru i ruchu drzew czy też innej roślinności. Wykorzystanie zaawansowanego silnika renderującego sprawia, że przy obecnej technologii można uzyskać w symulacji bardzo wysoki poziom odwzorowania grafiki.

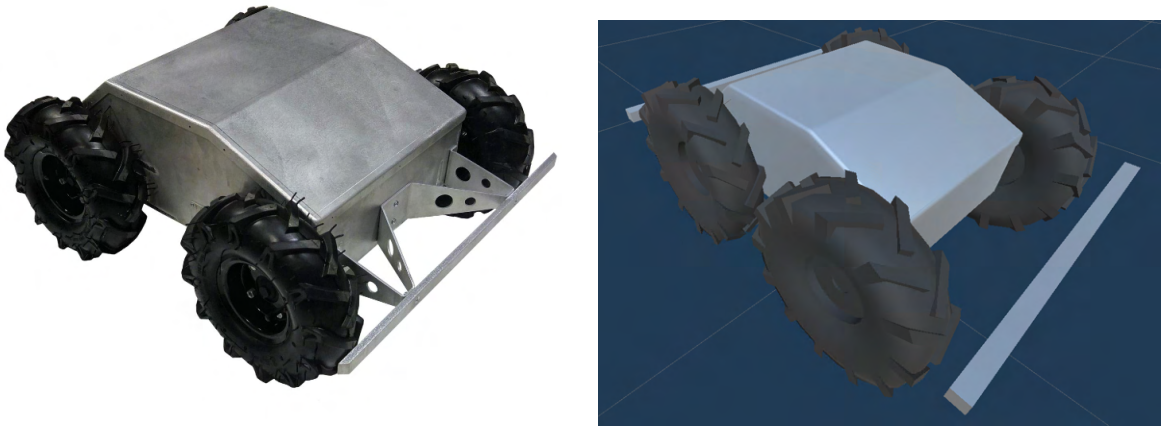
Wirtualny teren powinien zostać dodatkowo wzbogacony o właściwości fizyczne obsługiwane przez tzw. silnik fizyki. Trzy główne elementy odnoszące się do tego zagadnienia to model kolizyjny otoczenia, współczynnik tarcia danych elementów terenu czy pomieszczenia oraz siła grawitacji na danym obszarze. Parametry te mogą być dowolnie modyfikowane, co umożliwia symulowanie w WST różnych warunków, które w świecie rzeczywistym mogą być trudne do osiągnięcia, a mają kluczowy wpływ na dynamikę pojazdu oraz algorytmy autonomicznej jazdy.

4.3 Model pojazdu

Opracowany symulator obsługuje dowolną liczbę modeli pojazdów co pozwala na swobodną pracę nad różnymi projektami jednocześnie. Jako model pojazdu rozumie się model graficzny oraz model fizyczny, które w połączeniu umożliwiają działanie wirtualnych sensorów, sterowanie platformą oraz podgląd na działanie pojazdu w świecie symulacji.

4.3.1 Model graficzny

Istotą modelu graficznego jest wizualizacja pojazdu w środowisku symulacyjnym. Przygotowanie odpowiedniego modelu 3D wymaga zastosowania dowolnej z omówionych wcześniej metod modelowania oraz załadowania odpowiedniej tekstury nadającej wygląd zbliżony do rzeczywistego. Zastosowanie w WST silnika renderującego Unity opartego w tym przypadku na DirectX 11 i obsługującego programy cieniujące (z ang. *shaders*) sprawia, że wirtualny model może wykorzystywać zaawansowane efekty graficzne upodabniające go do rzeczywistego pierwowzoru. Model ten jest wizualną reprezentacją pojazdu w wirtualnym świecie, która umożliwia weryfikację położenia sensorów, w tym np. kontrolę pola widzenia kamer, ale także przeprowadzenie innych badań, które wymagają warstwy graficznej pojazdu.



(a) (b)

Rysunek 4.3: Rzeczywisty robot mobilny (a) oraz model 3D robota w środowisku symulacyjnym (b).

4.3.2 Model fizyczny

WST wykorzystuje silnik fizyki nVidia Physx dokonujący obliczeń w czasie rzeczywistym zaprojektowany z myślą o grach komputerowych i nadawaniu wirtualnym modelom fizycznych właściwości oraz umożliwianiu oddziaływania między nimi

w sposób zbliżony do rzeczywistości. Zdefiniowanie modelu fizycznego polega na podaniu parametrów fizycznych takich jak masa pojazdu, opór, współczynniki tarcia, sprężystości czy tłumienia dla wszystkich brył sztywnych (z ang. *rigid body*), które można wyodrębnić w ramach danego pojazdu oraz określenie ich modeli kolizyjnych umożliwiających detekcję zderzeń pomiędzy obiektami i symulację oddziaływań fizycznych między nimi. Określane są również punkty styku (z ang. *joint*) kolejnych brył składających się na cały pojazd (np. połączenie między kołem a zawieszeniem czy zawias między dwoma ramionami manipulatora) i ich parametrów takich jak zakres ruchu, masa czy limity prędkości i działających sił. Dzięki wykorzystaniu rozbudowanego edytora Unity część parametrów może zostać dobrana eksperymentalnie poprzez wielokrotne dostosowywanie współczynników tarcia, tłumienia czy sprężystości zawieszenia w taki sposób, aby wiernie odwzorować zachowanie symulowanego pojazdu. Sprawia to, że otrzymujemy ostatecznie model o sześciu stopniach swobody, który pozwala na przemieszczanie się po wirtualnej mapie, a dodatkowo sensory umieszczone w predefiniowanych punktach zapewniają interakcję z otoczeniem.

4.3.3 Sensory

Wirtualne sensory są nieodzowną częścią modelu pojazdu w WST. Opracowane środowisko symulacji zapewnia całą gamę urządzeń wykorzystywanych przez platformy autonomiczne takie jak m.in. lidary, kamery czy akcelerometry. Odpowiednio przygotowane parametry pozwalają na symulację dowolnego modelu sensora, którego zachowanie w różny sposób może wpływać na dane algorytmy autonomii jazdy. Szczegóły dotyczące poszczególnych modeli symulacyjnych czujników i ich implementacji znajdują się w sekcji 4.4.

4.4 Modele symulacyjne czujników

W tej sekcji przedstawiono sposoby w jakich najpopularniejsze czujniki wykorzystywane w pojazdach autonomicznych zostały zasymulowane na potrzeby Wir-

tualnego Systemu Testującego. Omówiono również szczegóły implementacyjne wirtualnych sensorów oraz zaprezentowano przykłady wygenerowanych danych. Wśród przygotowanych syntetycznych modeli czujników znalazły się m.in. IMU, enkodery kół wraz z generowaną odometrią, lidary i skanery laserowe, kamery oraz system nawigacji satelitarnej.

4.4.1 IMU

Symulacja IMU została przygotowana głównie na potrzeby ewaluacji algorytmów lokalizacji w przestrzeni. Wiele z algorytmów SLAM czy wyznaczania odometrii wykorzystuje dane z akcelerometru i żyroskopu, aby zwiększać dokładność estymacji położenia.

Akcelerometr zapewnia dane o przyspieszeniach działających na czujnik. Dysponując definicją wektora przyspieszenia jako pochodnej prędkości po czasie $\vec{a} = \frac{d\vec{v}}{dt}$ i drugiej pochodnej drogi po czasie $\vec{a} = \frac{d^2\vec{r}}{dt^2}$, w pierwszym kroku wyznaczana jest zmiana położenia obiektu symulacji pomiędzy dwoma punktami w okresie czasu odpowiadającemu częstotliwości rzeczywistego sensora, czyli prędkość obiektu.

Następnie bazując na wyznaczonej prędkości i wartości poprzedniej obliczane jest przyspieszenie. Ostatnim krokiem jest uwzględnienie wartości przyspieszenia ziemskiego. Analogicznie wyznaczana jest prędkość kątowa obiektu, a więc zmiana przebytego dystansu w danym odcinku czasu. Uzupełnienie stanowi kwaternion obrotu obiektu, który odczytywany jest bezpośrednio z symulacji. Następnie wszystkie wyznaczone wartości obarczane są błędem, którego wartość jest liczbą pseudolosową z zakresu wartości podawanych przez producenta danego modelu sensora lub dowolnych, które mogą zostać wyznaczone odrębnym badaniem.

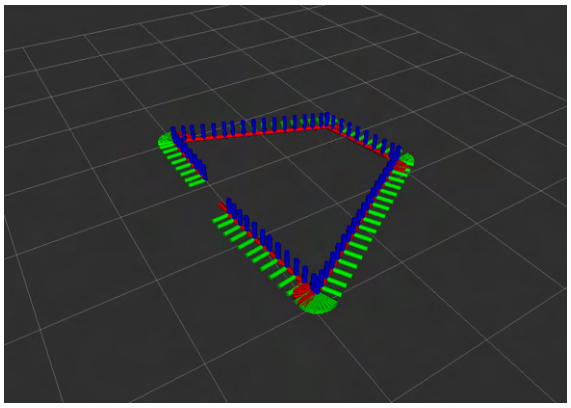
Maksymalna częstotliwość odświeżania danych jest zależna od kroku symulacji silnika fizyki, który został ustawiony na stałą wartość 2.5 ms. Odpowiada to częstotliwości 400 Hz, gdyż jest to wartość zgodna z czujnikami IMU wysokiej klasy dla zastosowań profesjonalnych. Opracowany model IMU obsługuje również generowanie danych z większym interwałem czasowym w celu dostosowania danych do

rzeczywistego sensora, który ma być symulowany.

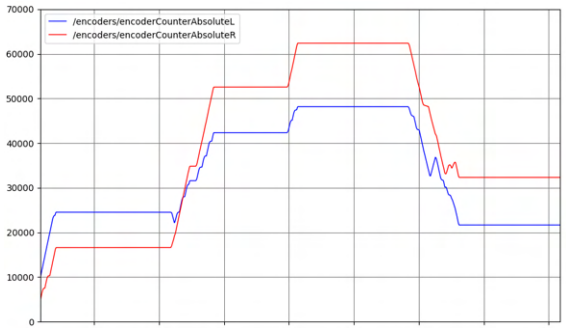
4.4.2 Odometria

Odometria jest przyrostową metodą lokalizacji pojazdów [113] posiłkującą się zazwyczaj danymi z czujnika umieszczonego w kole platformy. Dzięki całkowaniu informacji o przebytym dystansie następuje pomiar i wyznaczenie położenia pojazdu. Zazwyczaj dane odometryczne składają się z dwóch składowych: pozycji oraz wektora prędkości, a opcjonalnie także z odpowiadających im macierzy kowariancji błędów wyznaczonych wartości. Wartości te mogą być wyliczane np. na podstawie danych wejściowych z enkoderów zliczających obroty silnika. Takim klasycznym rozwiązaniem jest zastosowanie w pojeździe silników np. silników bezszczotkowych prądu stałego BLDC, które wyposażone są w czujniki Hall-a informujące o pozycji wirnika oraz wyjście cyfrowe, które sygnalizuje zboczem rosnącym impulsu 5v o obrocie koła o 1 stopień. Następnie impulsy te są zliczane przez sterownik silnika w 64-bitowych licznikach, których wartości stanowią wejście dla algorytmu wyznaczania odometrii. Ten z kolei, w zależności od rodzaju napędu platformy, wyznacza prędkości poszczególnych kół oraz ostatecznie przebyty dystans i trajektorię pojazdu. Niestety drobne zmiany prędkości czy nierówności drogi prowadzą do akumulacji błędów wraz z upływem czasu i odległością przebytą przez robota, powodując w efekcie duże błędy estymacji położenia [113]. Ponadto dane odometryczne wykorzystywane są także na potrzeby algorytmów lokalizacji w przestrzeni, np. SLAM czy nawigacji inercyjnej, dostarczając dodatkowych danych wykorzystywanych do efektywniejszego i dokładniejszego wyznaczania pozycji pojazdu.

Symulacja odometrii (rys. 4.4a) może przebiegać na różnym poziomie abstrakcji w zależności od oczekiwanych rezultatów. I tak, można z symulatora odczytywać pozycję oraz wektor prędkości pojazdu, ale również wspomniane liczniki impulsów, czy nawet same impulsy cyfrowe bądź dane z czujników Hall'a monitorujących pozycję wirnika. Aby uzyskać wyniki bardziej zgodne z rzeczywistością zdecydowano o symulacji liczników enkoderów poprzez obliczanie różnicy kątów pomiędzy obrotem koła w kolejnych krokach symulacji silnika fizyki. Następnie, gdy różnica



(a) Wizualizacja odometrii robota wyznaczonej na podstawie symulowanych danych z enkoderów.



(b) Przykładowe zmiany wartości enkoderów w czasie generowane przez WST.

Rysunek 4.4: Odczyty danych odometrycznych z symulatora.

osiąga wartość zgodną z rozdzielczością kątową rzeczywistych enkoderów, licznik odpowiadający danemu kołu jest inkrementowany lub dekrementowany w zależności od kierunku ruchu, a sama wartość licznika przesyłana do systemu autonomii z częstotliwością zgodną z zastosowanym sterownikiem w rzeczywistym pojeździe. W ten sposób dane odometryczne mogą być wyliczane przez ten sam algorytm, który wykorzystywany jest w rzeczywistym robocie. Rys. 4.4b przedstawia przykładowe wartości generowane w WST przez enkodery zaimplementowane w opisany wyżej sposób. Listing 4.1 przedstawia fragment kodu odpowiedzialny za obliczanie i wysyłanie danych z enkoderów w środowisku symulacji.

```
float minTurnForEncoder = 360.0f / ticksOnFullTurn;
float deltaAngleLeft = Mathf.DeltaAngle(lastAngleLeft, angleLeft);
float deltaAngleRight = Mathf.DeltaAngle(lastAngleRight, angleRight);

if (Mathf.Abs(deltaAngleLeft) > minTurnForEncoder)
{
    wheelEncoders.counterAbsoluteL += AngleToEncoderTick(deltaAngleLeft);
    lastAngleLeft = angleLeft;
}

if (Mathf.Abs(deltaAngleRight) > minTurnForEncoder)
{
```

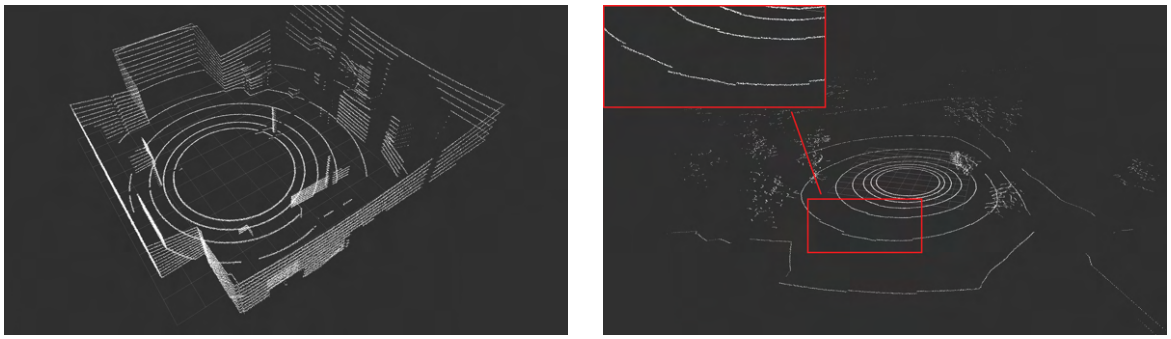
```
    wheelEncoders.counterAbsoluteR += AngleToEncoderTick(deltaAngleRight);
    lastAngleRight = angleRight;
}

if (sensorFrequency.ShouldScanBePerformed())
{
    rosSender.Send(wheelEncoders)
}
```

Listing 4.1: Fragment kodu odpowiedzialny za generowanie odczytów z enkoderów.

4.4.3 Skanery laserowe

Skanery laserowe, nazywane też lidarami, są kluczowym urządzeniem dla pojazdów autonomicznych. Symulacja tych sensorów wymaga zastosowania wydajnego mechanizmu, który pozwala na precyzyjne wyznaczanie odległości do najbliższych obiektów w przestrzeni 3D. W zależności od rodzaju tego sensora wymagane jest przetwarzanie od kilku tysięcy do nawet miliona pomiarów na sekundę. Przykładowy skan lidarowy laboratorium testowego został przedstawiony na rys. 4.5a.



(a) Przykładowy skan lidarowy. (b) Symulacja efektu *rolling shutter*.

Rysunek 4.5: Symulowane skany lidarowe.

Wirtualne lidary umożliwiają rozbudowaną konfigurację w zależności od rzeczywistego sensora, który ma być symulowany. Podstawowym parametrem jest liczba kanałów, która określa ile wirtualnych sensorów jest pionowo ustawionych, a więc odpowiada za rozdzielczość pionową sensora. W przypadku lidarów 2D wartość ta powinna wynosić 1, natomiast lidary 3D oferują zazwyczaj 16 lub 32 kanały. Drugi

z parametrów odnosi się do pola widzenia sensora w poziomie, które może wynosić maksymalnie 360 stopni oraz pole widzenia w pionie, które dotyczy lidarów 3D. Kolejny parametr określa kąt pomiędzy kolejnymi pomiarami obracających się laserów wokół osi pionowej. Jest to de facto rozdzielczość pozioma, która może różnić się w znaczący sposób, w zależności od klasy zastosowanego urządzenia. Przykładowo, najtańsze lidary 2D oferują rozdzielczość poziomą w okolicach 400 punktów na skan. Odpowiada to wartości ok. 0.9 stopnia. Natomiast w przypadku profesjonalnych lidarów wartość ta może sięgać zaledwie 0.25 stopnia co odpowiada 1440 punktów na pełny skan 360 stopni. Istotnym parametrem jest również pole widzenia sensora w dyskretnym odcinku czasu symulacji, co dotyczy opisanego w dalszej części rozprawy efektu *rolling shutter* (rys. 4.5b), co z kolei odpowiada rozdzielczości kątowej przypadającej na blok danych w fizycznym urządzeniu. Dodatkowy parametr określa błędy pomiarowe, każdego z dokonanych odczytów. Są one definiowane w sposób zgodny z parametrami dostarczanymi przez producenta. Tabela 4.1 zawiera zestawienie wszystkich parametrów oraz porównanie różnic przykładowego lidaru z symulowanym. Wśród różnic między symulacją, a rzeczywistym lidarem można zauważyć tryb pomiaru, który nie dotyczy pozyskiwania danych syntetycznych. Natomiast w przypadku rzeczywistego sensora detekcja odbicia pomienia laserowego, a tym samym pomiar może dotyczyć najmocniejszego sygnału bądź ostatniego, który dotrze do detektora w danym oknie czasowym.

Parametr	Jednostka	VLP-16	Symulacja
Liczba kanałów	–	16	16
Pole widzenia w pionie	stopień	30°	30°
Liczba próbek w poziomie	–	3600	3600
Pole widzenia w poziomie	stopień	360°	360°
Rozdzielczość kątowa na blok danych	stopień	2.4°	15°
Zasięg	m	100 m	100 m
Dokładność pomiaru	m	0.03 m	0.03 m
Częstotliwość obrotu	Hz	10	10
Tryb pomiaru	–	najlepszy/ostatni	nd.

Tablica 4.1: Parametry lidaru Velodyne VLP-16 oraz jego odpowiednika w symulacji.

Symulacja lidaru, a efekt *rolling shutter*

Termin *rolling shutter effect* zwykło się używać w kontekście fotografii, aby opisać charakterystyczne zniekształcenia zdjęć szybko poruszających się obiektów, których ruch odbywa się prostopadle do osi kamery. Jest to spowodowane budową mechanizmu migawki w aparacie cyfrowym, gdzie zdjęcie otrzymywane jest poprzez odczyt danych z matrycy światłoczułej linia po linii. Bardzo podobny efekt można zaobserwować w kontekście lidaru, chociaż oczywiście przyczyna jest zupełnie inna [114]. Dane lidarowe uzyskiwane są dzięki użyciu nadajników laserowych oraz detektorów ułożonych pionowo, które obracają się z prędkością nawet do 20 razy na sekundę i dostarczają danych, dzięki którym szacowana jest odległość do kolejnych punktów w przestrzeni. Z powodu relatywnie niewielkiej częstotliwości pomiarów lidarów, czas wykonania pełnego skanu otoczenia jest stosunkowo długi (dla standardowego lidaru Velodyne VLP-16 działającego w trybie 10 Hz, pełny skan zajmuje 100 ms). W przypadku, gdy pojazd z zamontowanym lidarem porusza się z daną prędkością występuje efekt przesunięcia kolejnych danych w kierunku ruchu platformy, gdzie efekt nasila się wraz ze wzrostem prędkości. Lidary przesyłają dane w niewielkich pakietach (blokach) danych, w których ze względu na niewielką ilość punktów efekt nie jest zbyt widoczny, ale po konkatenacji danych w pełny skan efekt jest wyraźnie widoczny.

Aby w realistyczny sposób symulować działanie lidaru istotne jest, aby zawrzeć w symulacji efekt *rolling shutter*. Wraz z każdym krokiem symulacji przeliczany jest tylko minimalny fragment skanu, po czym następuje odświeżenie pozycji symulowanego pojazdu, dzięki czemu pełna chmura punktów zawiera dane obarczone błędem zbliżonym do efektu na rzeczywistym lidarze. W zależności od wydajności komputera, na którym uruchomiony jest symulator, pole widzenia sensora w danym pakiecie danych może być dostosowywane, aby w najlepszy możliwy sposób odwzorować działanie rzeczywistego lidaru. Rys. 4.5b przedstawia wpływ tego efektu na otrzymywaną chmurę punktów przy ruchu pojazdu w symulacji. Charakterystykę tego zniekształcenia wykorzystuje m.in. algorytm SLAM [115], który wykorzystuje małe fragmenty skanów lidarowych oraz odczytów z akcelerometru do oszacowa-

Rysunek 4.6: Przykładowy obraz z wirtualnej kamery symulatora.

nia ruchu pojazdu i dostosowania kolejnych danych w celu dokładniejszej estymacji rotacji i translacji osiąganej podczas działania algorytmu lokalizacji i mapowania. W sekcji 5.2 zaprezentowano znaczący wpływ symulacji efektu *rolling shutter* na dokładność wyznaczania pozycji i propagację błędu lokalizacji w czasie, co oznacza, że dzięki implementacji tego mechanizmu w symulatorze ewaluacja algorytmów SLAM wykorzystujących lidary jest możliwa bez rzeczywistego urządzenia.

4.4.4 Kamery

Opracowany symulator działając w pełnym środowisku 3D umożliwia szczegółową wizualizację wirtualnej sceny, po której może poruszać się pojazd. Tym samym otoczenie może być obserwowane przez wirtualne kamery umieszczone w odpowiednich miejscach platformy, które korzystając z silnika renderującego wygenerują obraz o wysokiej jakości imitujący rzeczywistą kamerę, który może posłużyć do dalszej obróbki przez system autonomii.

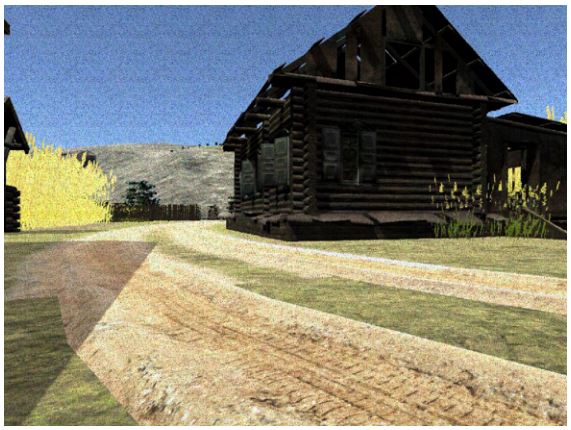
Dostępnych jest wiele rodzajów kamer, które znajdują zastosowanie przy opracowywaniu algorytmów autonomicznej jazdy. Podstawowym rozwiązaniem jest jednoobiektywowa kolorowa kamera, której symulacja dotyczy kilku parametrów wpływających na rozmiar przekazywanego obrazu, jakość oraz imitację zniekształceń, które w rzeczywistym urządzeniu spowodowane są niedoskonałościami matrycy bądź obiektywu. Na rys. 4.6 przedstawiono fragment wirtualnej mapy. Wśród głównych parametrów obrazu należy wyróżnić:

- rozdzielczość,
- liczba klatek na sekundę,
- jakość kompresji,
- pole widzenia kamery.

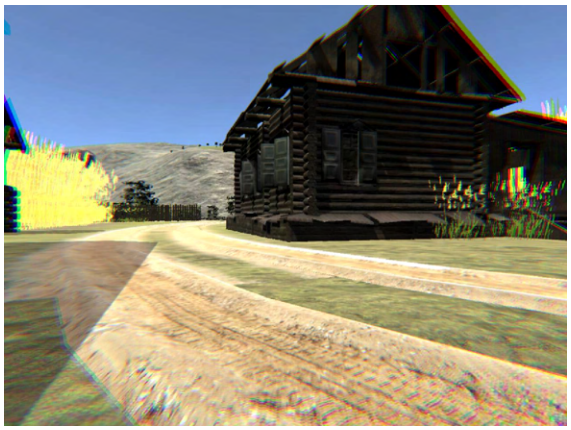
Dodatkowo, dzięki zastosowaniu silnika renderującego z obsługą programów cieniujących, możliwe jest przeprowadzenie tzw. *post-processingu* na już wygenerowanej klatce obrazu w celu wprowadzenia zniekształceń charakterystycznych dla cyfrowych kamer. Każdy z dostępnych efektów (na rys. 4.7 przedstawiono kilka z nich) może zostać sparametryzowany wedle rzeczywistego modelu kamery, a należą do nich:

- szum (ziarnienie) (rys. 4.7a),
- aberracja chromatyczna (rys. 4.7b),
- dystorsja beczkowa (rys. 4.7c) lub poduszkowa,
- winietowanie (rys. 4.7d),
- głębia ostrości.

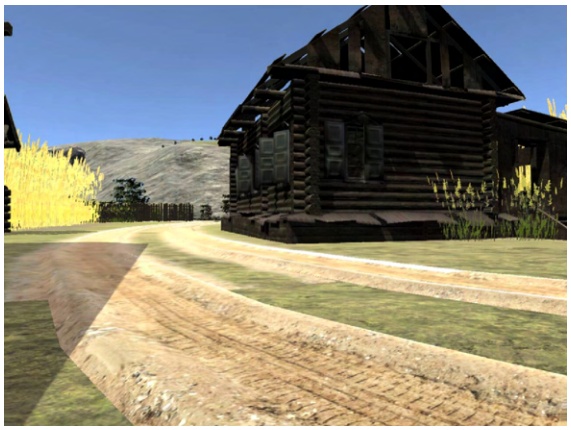
Drugi rodzaj kamery, który często jest wykorzystywany przy systemach autonomii to kamera stereowizyjna. Jej symulacja polega na przygotowaniu w modelu dwóch wirtualnych kamer, których ułożenie zgodne jest z konstrukcją rzeczywistego



(a) Szum (ziarnienie).

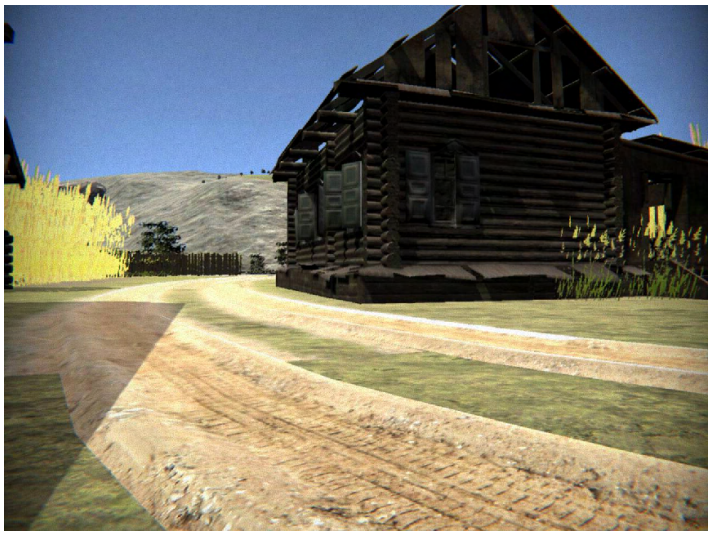


(b) Aberracja chromatyczna.



(c) Dystorsja beczkowa.

(d) Winietowanie.



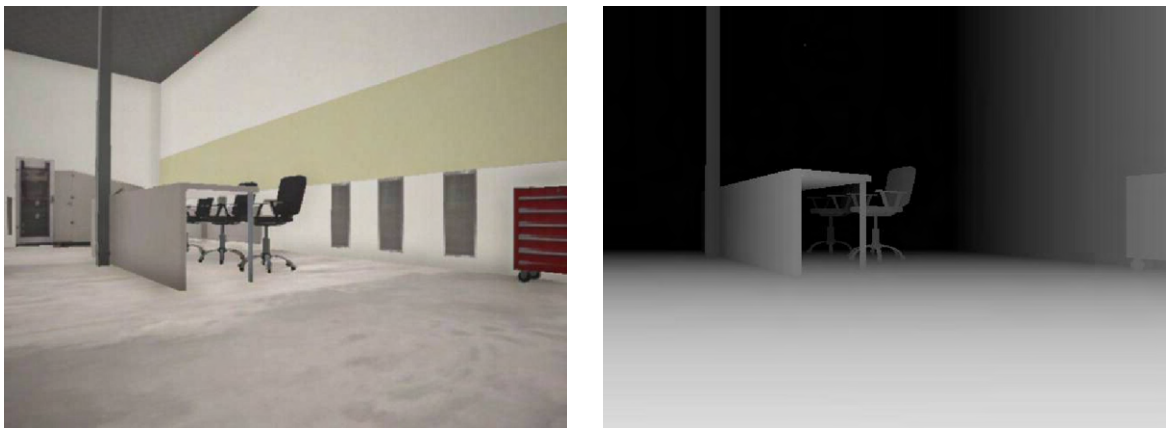
(e) Kilka połączonych efektów symulujących obraz z rzeczywistej kamery.

Rysunek 4.7: Symulacja zniekształceń na obrazie z wirtualnych kamer w WST.

modelu, a więc istotne jest precyzyjne umiejscowienie czujników w przestrzeni 3D. W zależności od pożądanej konfiguracji zniekształcenia obrazu mogą być zdefiniowane osobno dla lewej i prawej kamery lub jednocześnie dla obu.

Kolejny model wirtualnej kamery dotyczy kamery głębi. Istnieje wiele rodzajów urządzeń oraz algorytmów, które w rezultacie potrafią uzyskać mapę głębi zawierającą informacje o głębokości poszczególnych pikseli na obrazie [116]. Wśród rozwiązań sprzętowych dominują kamery bazujące na ToF (z ang. *Time-of-Flight*), czyli pomiarze czasu jakie pokonuje światło od nadajnika do odbiornika, lub takie wyposażone w miniaturowe lidary wykonane w technologii MEMS. Mapa głębi wyznaczona może być również za pomocą kamery stereowizyjnej dzięki porównywaniu punktów charakterystycznych i ich położenia. Metoda ta jednak w dużym stopniu zależna jest od warunków oświetleniowych, dlatego też możliwe jest zastosowanie dodatkowego nadajnika światła podczerwonego w celu uzyskania dokładniejszych wyników. Jednakże w przypadku symulacji kamery głębi, szczegóły działania rzeczywistego czujnika odgrywają drugorzędną rolę, gdyż podczas implementacji wirtualnego modelu tego typu sensora celem było uzyskanie mapy głębi, której wygląd nie różni się zasadniczo w zależności od wyboru rozwiązania. Aby uzyskać ten efekt wykorzystano bufor głębokości generowany przez silnik renderujący w celu poprawnego, z perspektywy wirtualnej kamery, wyświetlania trójwymiarowej sceny. Zawiera on informacje o współrzędnej Z dla każdego piksela wyrenderowanego obrazu. Efekt osiągnięto za pomocą programu cieniującego, który dzięki wykorzystaniu karty graficznej potrafi odczytać informacje z tzw. buforu głębokości bez konieczności wykonywania dodatkowych obliczeń. Na rys. 4.8 przedstawiono obraz z wirtualnej kamery oraz uzyskaną mapę głębi. Konfiguracja wirtualnego sensora sprowadza się do zdefiniowania takich samych parametrów jak w przypadku standardowej kamery oraz dodatkowo minimalnego i maksymalnego dystansu w jakim wyznaczana jest głębia obrazu.

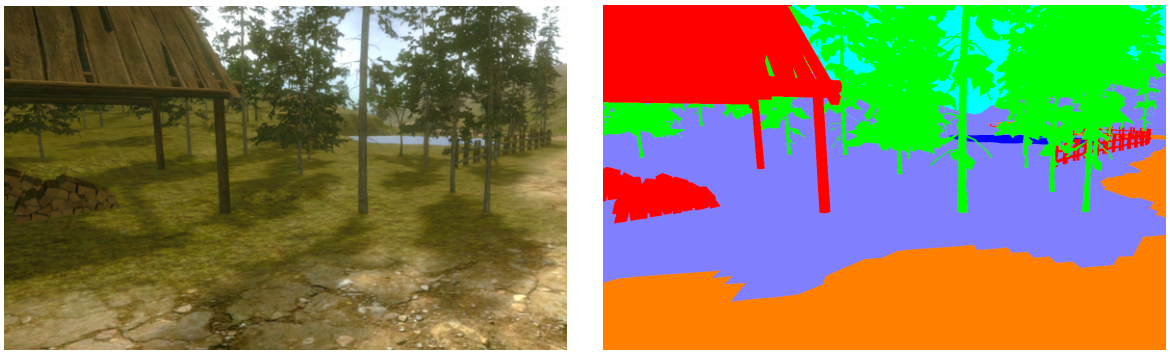
Ostatni z zaimplementowanych modeli kamery nie dotyczy rzeczywistego urządzenia. Odpowiada on bowiem za symulację działania algorytmu segmentacji semantycznej obrazu. Jest to kolorowa mapa semantyczna, której kolejne wartości pikseli oznaczają klasę obiektu, który znajduje się na obrazie ze standardowej ka-



(a) (b)

Rysunek 4.8: Obraz z wirtualnej kamery oraz odpowiadająca mu mapa głębi.

mery. Rozwiązanie to przydatne jest szczególnie, gdy symulator służy do generowania danych treningowych dla głębokich sieci neuronowych wykonujących segmentację semantyczną. Na rys. 4.9 przedstawiono obraz z wirtualnej kamery oraz wygenerowaną mapę semantyczną. Implementacja tego typu rozwiązania możliwa była dzięki zastosowaniu niestandardowego potoku renderowania, który, w zależności od wirtualnej kamery, dla poszczególnych modeli w symulacji stosował inny zestaw materiałów i tekstur oraz nie stosował programów cieniujących uwzględniających dostępne na scenie oświetlenie.

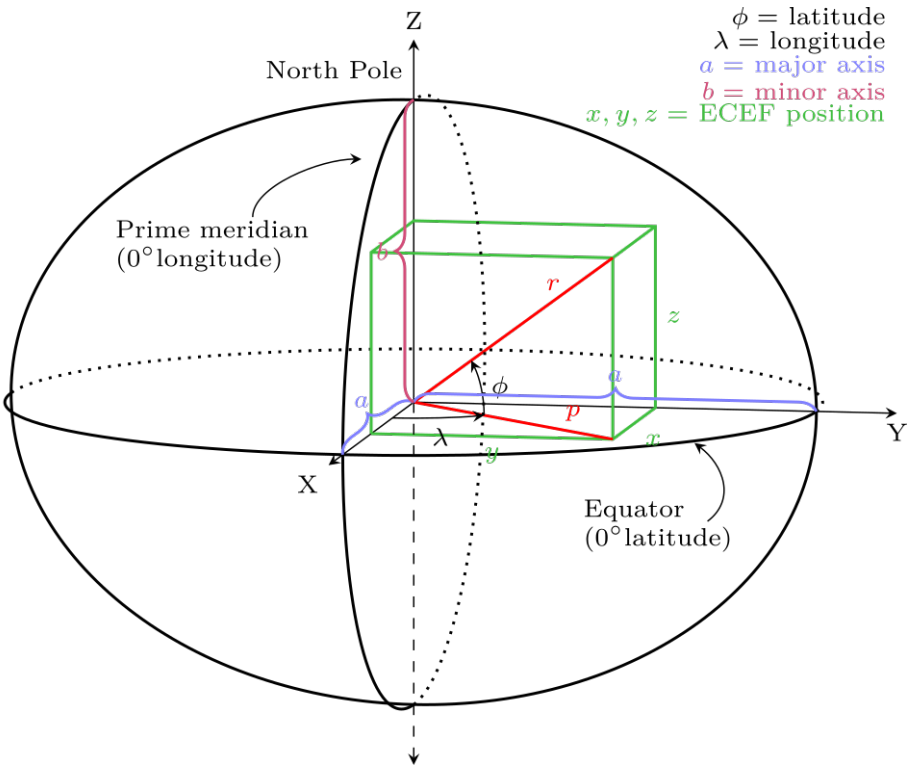


(a) (b)

Rysunek 4.9: Obraz z wirtualnej kamery oraz odpowiadająca mu mapa segmentacji.

4.4.5 GNSS

Działanie systemów nawigacji satelitarnej opiera się na wysyłaniu sygnałów radiowych z dużej liczby satelitów umieszczonych na orbicie okołoziemskiej. Odbiornik na ziemi dokonując pomiaru czasu dotarcia sygnału określa odległość od poszczególnych satelitów. Jako, że sygnał GNSS oprócz precyzyjnego znacznika czasowego nadania sygnału zawiera również informacje o położeniu satelitów na niebie i ich teoretycznej trajektorii ruchu odbiornik jest w stanie wyznaczyć pozycję w postaci długości, szerokości i wysokości geograficznej, a następnie przeliczyć ją do wybranego układu odniesienia. Ogólnoświatowym standardem wykorzystywanym na potrzeby m.in. geodezji czy nawigacji satelitarnej jest system WGS-84 opisujący kształt Ziemi i szczegółowe parametry elipsoidy.



Rysunek 4.10: Zależność między układem współrzędnych ECEF, a LLA [117].

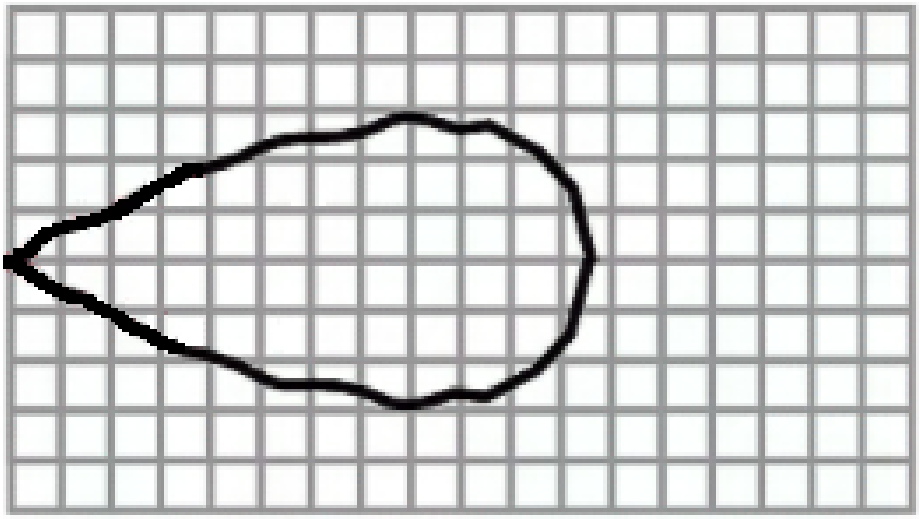
Symulacja odbiornika GNSS nie uwzględnia propagacji sygnału w wirtualnym otoczeniu z punktów w przestrzeni 3D znajdujących się w odległości kilkunastu tysięcy kilometrów. Zamiast tego dokonywane są przekształcenia matematyczne

pomiędzy różnymi systemami odniesienia. W pierwszym kroku należy zdefiniować długość i szerokość geograficzną oraz wysokość n.p.m., którą przyjmujemy w punkcie startu robota w symulacji XYZ = (0,0,0). Następnie wartości te należy przeliczyć na kartezjański układ współrzędnych ECEF (z ang. *Earth-centered, Earth-fixed coordinate system*). Na rys. 4.10 przedstawiono zależność pomiędzy układem ECEF, a LLA (z ang. *latitude, longitude, altitude*), a dzięki parametrom określonym przez standard WGS-84 możliwe jest swobodne przeliczanie pozycji między dwoma układami. Każde odświeżenie danych z wirtualnego sensora GNSS wymaga zaktualizowania pozycji ECEF wyznaczonej na początku i ponownego przekształcenia jej do układu współrzędnych LLA. Dodatkowo, każdy pomiar może zostać obarczony błędem pomiarowym, który jeszcze przed aktualizacją danych do współrzędnych w formacie XYZ dodaje wartości pseudolosowe o rozkładzie normalnym odpowiadające deklarowanej dokładności pomiarowej. Wśród parametrów symulacji dotyczących GNSS dostępny jest dodatkowy parametr określający częstotliwość odświeżania danych, który w przypadku rzeczywistych odbiorników nawigacji satelitarnej oscyluje zazwyczaj między 1 Hz, a 5 Hz.

4.4.6 Ultradźwiękowy czujnik odległości

Czujniki odległości bazujące na emitowaniu sygnałów ultradźwiękowych i ich detekcji są popularnym rozwiązaniem w robotyce ze względu na niski koszt i prostotę otrzymywanych danych. W zależności od zastosowanego sensora czujniki te zwracają odległość do wykrytej przeszkody w ich polu działania. Przygotowanie modelu sensora w symulacji wymagało opracowania metody detekcji przeszkód na określonym obszarze.

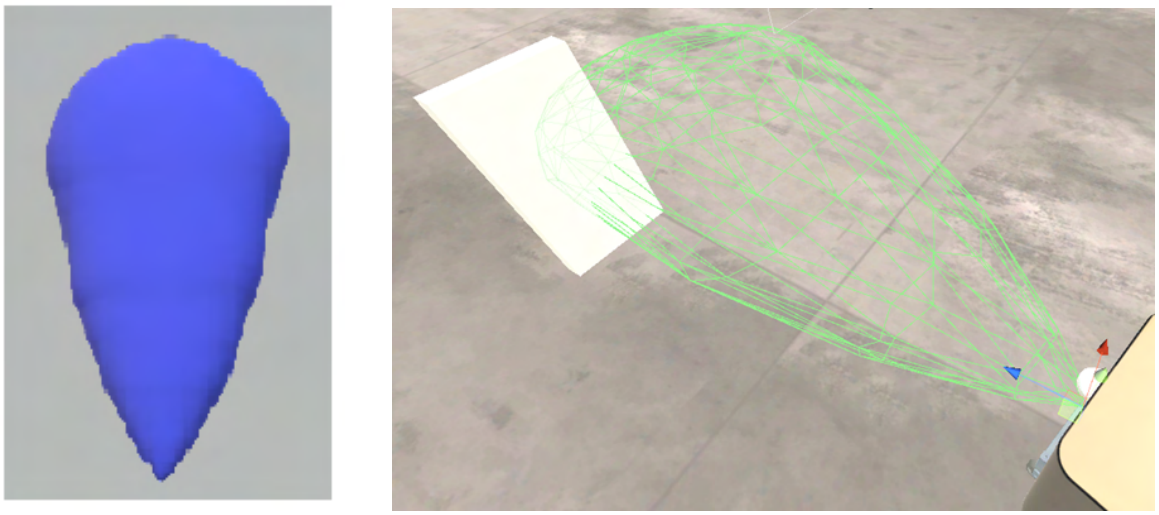
Początkowe rozwiązanie zaimplementowane w WST opierało się na technice śledzenia promieni (z ang. *raytracing*) działającej w sposób podobny do wirtualnego modelu sensora LiDAR 2D. W związku z tym wymagało wielu promieni skanujących w polu widzenia określonym zgodnie z modelem czujnika. Takie rozwiązanie wymagało jednak sporej mocy obliczeniowej, a ponadto było dość zawodne. W symulacji pojedynczy promień ma punktowy obszar detekcji co sprawia, że czujnik



Rysunek 4.11: Obszar detekcji ultradźwiękowego czujnika odległości SR04T według noty katalogowej producenta.

ultradźwiękowy zasymulowany wyłącznie przy pomocy techniki raytracingu działa jedynie w płaszczyźnie 2D. Ponadto jego obszar skanowania jest wyznaczony poprzez wycinek kołowy, co jest niezgodne z rzeczywistym sensorem. Dla przykładu wykorzystano sensor SR04T o zintegrowanym z odbiornikiem emiterze ultradźwięków, którego obszar detekcji przedstawiono na rys. 4.11.

Aby rozwiązać powyżej opisane problemy postanowiono wykrywać przeszkody w inny sposób. W tym celu zamodelowano wirtualny model obszaru skanowania sensora odległości tak jak przedstawiono na rys. 4.12a. Następnie zastosowano funkcję detekcji kolizji pomiędzy modelami 3D w silniku Unity, której implementacja opiera się na bibliotece symulowania zachowań fizycznych PhysX. Dzięki temu możliwe okazało się precyzyjne wykrycie obiektu, który przecina wirtualny model detekcji umiejscowiony w punkcie odpowiadającym rzeczywistemu sensorowi. Taka informacja z kolei pozwoliła na pomiar odległości do najbliższego punktu obiektu, który został wykryty we wspomnianym obszarze detekcji. W tym przypadku jednak pomiar ten wymagał jedynie obliczenia odległości od wirtualnego punktu pomiaru do właściwego punktu na wykrytej przeszkodzie. Na rys. 4.12b przedstawiono wizualizację detekcji obiektu w obszarze skanowania wirtualnego czujnika ultradźwiękowego.



(a) Model obszaru detekcji. (b) Wirtualny czujnik odległości w WST.

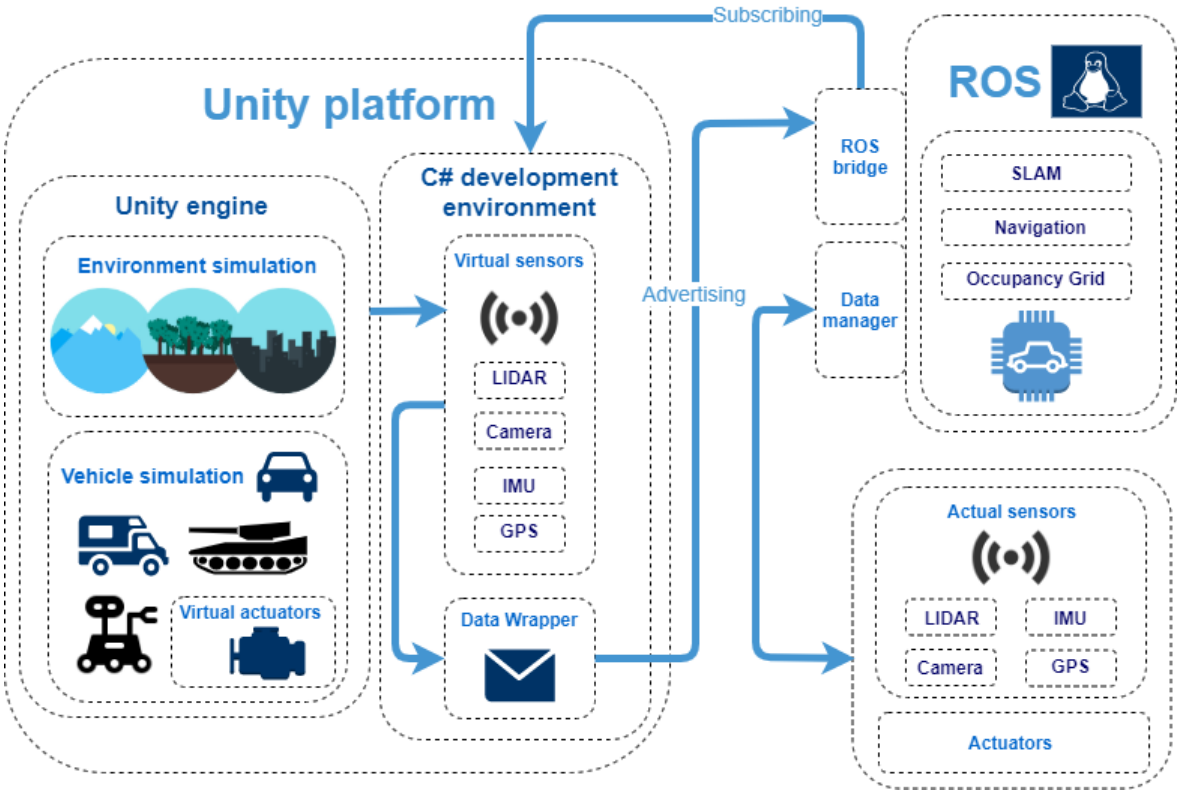
Rysunek 4.12: Symulacja ultradźwiękowego czujnika odległości.

4.5 Interfejs komunikacyjny UDP Bridge

Jednym z głównych założeń przy projektowaniu WST było zapewnienie pełnej transparentności pomiędzy rzeczywistymi danymi z fizycznych urządzeń znajdujących się na pojeździe, a wirtualnymi pochodzącymi z symulatora. W tym celu niezbędne było zagwarantowanie nie tylko zgodnego formatu danych, ale również wydajnego protokołu współpracującego z całym systemem. Opracowany UDP Bridge umożliwia przesyłanie symulowanych danych sensorycznych z symulatora do systemu autonomii z zachowaniem formatu danych i transparentności względem rzeczywistych urządzeń. W sekcji 5.5 przedstawiono szczegóły wykorzystania UDP Bridge w opracowanym symulatorze, porównano prędkość działania z konkurencyjnym rozwiązaniem oraz przedstawiono przykładowe zastosowanie przesyłania danych z symulacji na potrzeby porównania działania algorytmów SLAM opartych na danych z kamerach oraz czujników LiDAR.

WST opracowany w ramach doktoratu pozwala na pełną integrację z ROS poprzez autorskie rozwiązanie UDP Bridge zgodnie ze schematem wskazanym na rys. 4.13. W momencie rozpoczęcia prac dostępna była również biblioteka ROS# [118] oferująca podobne funkcjonalności. Przeprowadzono jednak szereg badań i testów,

mających na celu weryfikację działania obu rozwiązań i określenia, które z nich oferuje większą wydajność przy przesyłaniu danych między symulatorem, a systemem ROS. Ze względu na stosowanie WST wraz z ROS w obrębie jednego komputera, problem ewentualnej utraty pakietów i tym samym niezawodności UDP Bridge nie stanowi istotnego problemu.



Rysunek 4.13: Diagram komunikacji między WST, a systemem autonomii.

UDP Bridge wykorzystuje protokół pakietów użytkownika UDP (z ang. *User Datagram Protocol*), który nie gwarantuje dostarczenia kolejnych datagramów. Ze względu na możliwości wysyłania dużej ilości danych z symulatora zdecydowano się na zastosowanie tego bezpołączeniowego protokołu, aby zminimalizować narzuty czasowe, które w przypadku TCP (a ang. *Transmission Control Protocol*) pojawiają się przy nawiązywaniu połączenia, utrzymywaniu sesji czy retransmisji. Nagłówek UDP jest ograniczony do zaledwie 4 bajtów, gdyż zawiera jedynie numer portu docelowego i źródłowego, długość pakietu oraz sumę kontrolną. Pozwala to również zwiększyć liczbę danych, które mogą zostać przesłane pojedynczym pakietem. UDP

Bridge oferuje dwukierunkową komunikację między WST a symulatorem poprzez opracowane biblioteki zarówno dla systemu ROS, jak i platformy .NET, na której oparty jest symulator.

Nazwa pola	Typ danych	Rozmiar w bajtach	Opis
MsgType	int8_t	1	typ wiadomości
HeaderSize	int8_t	1	rozmiar nagłówka w bajtach
SeqNumber	int32_t	4	numer sekwencyjny - każda nowa wiadomość dysponuje kolejnym indeksem
PartNumber	int8_t	1	numer fragmentu wiadomości, stosowany przy podziale dużych wiadomości
PartsMax	int8_t	1	liczba wszystkich fragmentów danej wiadomości
TimeSec	int32_t	4	czas w formacie POSIX - część całkowita
TimeNsec	int32_t	4	czas w formacie POSIX - część ułamkowa
TopicSize	int8_t	1	rozmiar pola zawierającego nazwę tematu
LinkSize	int8_t	1	rozmiar pola zawierającego nazwę linku w drzewie transformacji
Topic	string	TopicSize	nazwa tematu, na którym będzie publikowana wiadomość
Link	string	LinkSize	nazwa linku w drzewie transformacji
Padding	nd.	0/1/2/3	wyrównanie długości nagłówka do wielokrotności 4

Tablica 4.2: Struktura nagłówka wiadomości przesyłanej przez UDP Bridge.

Pakiety przesyłane przez UDP Bridge dysponują dodatkowym nagłówkiem o zmiennej długości, który zawiera szereg elementów niezbędnych do pełnej integracji danych wirtualnych z ROS oraz efektywną dwukierunkową komunikację między ROS a WST. W tabeli 4.2 przedstawiono strukturę nagłówka danych, którego minimalna długość wynosi 20 bajtów. Każdy typ wiadomości ma określony indeks, na podstawie którego parsowane są dane przesyłane w pakiecie poza częścią nagłówkową. UDP Bridge pozwala przesyłać wiele typów danych obsługiwanych w ROS, w tym m.in.:

- *PointCloud2* - pomiary pozyskiwane z lidaru, w zależności od modelu sensora, może to być fragment lub cała chmura punktów 3D,
- *Imu* - odczyty zawierające dane z akcelerometru o działających na obiekt przyspieszeniach oraz z żyroskopu z informacjami o prędkościach kątowych,
- *CompressedImage* - skompresowany obraz w formacie JPEG,
- *NavSatFix* - odczyty z nawigacji satelitarnej w postaci długości i szerokości geograficznej oraz wysokości, a także status systemu,
- *WheelEncoders* - dwie wartości całkowite określające liczniki enkoderów kół w pojeździe,
- *Range* - pomiary z czujników odległościowych zawierające dystans, zasięg działania, pole widzenia oraz rodzaj urządzenia.

Nagłówek zawiera również informację o nazwie tematu, w ramach którego mają być publikowane dane w ROS oraz nazwie połączenia w drzewie transformacji, którego dotyczą przesyłane dane. UDP Bridge umożliwia również wymianę danych innych typów w celu efektywnej komunikacji z ROS oraz pozwala na realizację aplikacji użytkownika zarządzającego rzeczywistym pojazdem autonomicznym niezależnie od symulatora. Udostępniane w tym celu dane to m.in.:

- *Subscribe* - pakiet kontrolny zawierający informację z prośbą o subskrybowanie wiadomości w ramach danego tematu w ROS,
- *Clock* - czas służący do synchronizowania kolejnych pomiarów dokonywanych w symulatorze,
- *Twist* - dwa wektory z prędkością liniową i kątową, które pełnią rolę standardowych danych wejściowych dla sterownika silników lub modelu fizycznego w WST,
- *PoseStamped* - pozycja w przestrzeni i czasie, zazwyczaj zawiera pozycję robota względem mapy,

- *OccupancyGrid* - dwuwymiarowa mapa otoczenia zawierająca informacje o przeszkodach, wraz z *PoseStamped* pozwala na wizualizację pojazdu w konkretnym miejscu na mapie,
- *ServiceRequest* - uniwersalny pakiet zawierający szczegóły zapytania dla dowolnego serwisu w ROS,
- *ServiceResponse* - uniwersalny pakiet zawierający odpowiedź pochodzącą z wywołania serwisu w ROS.

Zasada działania biblioteki ROS# odbiega w znaczący sposób od UDP Bridge. Do połączenia z ROS stosowany jest protokół o nazwie *WebSocket* najwyższej warstwy aplikacji modelu OSI wykorzystujący TCP i zapewniający kompatybilność z HTTP. Ponadto dane przesyłane są za pośrednictwem formatu o nazwie JSON (z ang. *JavaScript Object Notation*), który w przeciwieństwie do danych binarnych powoduje znaczną nadmiarowość danych oraz ogranicza maksymalną ilość danych, które można przesłać w pojedynczym pakiecie, co jest problematyczne, szczególnie w przypadku przesyłania z symulatora takich danych, jak obraz z wirtualnych kamer bądź dane lidarowe, zwłaszcza z lidarów 3D.

```
while (ROS is OK){

    if (Input packet in Input queue) {
        Pop Input packet from Input queue;
        Get Header from Input packet;
        Get Message from Input packet;

        if (Message is fragmented) {
            Merge Message;
        }

        if (Message is complete) {
            Check Message timestamp;
            Get Type of message from Header;
            Process Message based on Type;
        }
    }
```

```
    if (Output packet in Output queue) {
        Push Output packet to Output queue;
    }
}
```

Listing 4.2: Pseudokod wątku przetwarzania danych.

Z perspektywy implementacji UDP Bridge kluczowe było zapewnienie najwyższej wydajności opracowywanego rozwiązania. W związku z tym jest to wielowątkowy program napisany w C++ jako węzeł systemu ROS. Na potrzeby zastosowania kodu w symulacji przygotowano interfejs programistyczny dla platformy .NET oraz Unity w języku C#. Program ten składa się z dwóch wątków współpracujących ze sobą: pierwszy odpowiada za odbieranie i wysyłanie danych z gniazda UDP, drugi natomiast umożliwia przetwarzanie danych i generowanie odpowiedzi w ramach pakietów wyjściowych. Przekazywanie informacji pomiędzy głównymi wątkami aplikacji odbywa się za pomocą kolejki zaimplementowanej bezpiecznie wielowątkowo. Na listingu 4.2 przedstawiono główną pętlę przetwarzania otrzymanych danych i przygotowania odpowiedzi, a na listingu 4.3: pseudokod nakreślający mechanizm odbioru i wysyłania danych.

```
while (Socket is open){

    Get Input buffer from UDP socket;

    if (size of Input buffer \textbf{greater than} 0) {
        Create Input packet from Input buffer;
        Push Input packet to Input queue;
    }

    while (Output packet in Output queue) {
        Pop Output packet from Output queue;
        Create Output packet from Output buffer;
        Send Output buffer to UDP socket;
    }
}
```

Listing 4.3: Pseudokod wątku odbierania/wysyłania danych.

4.6 Podsumowanie

Przygotowanie odpowiedniego środowiska symulacyjnego wymaga przemyślanej architektury systemu oraz poprawnie zdefiniowanych wymagań. Najważniejszymi elementami są model pojazdu wraz z wirtualnymi sensorami takimi jak LiDAR, IMU czy kamery. Istotną rolę pełni model otoczenia, który wchodzi w interakcję z pojazdem oraz zapewnia dane dla poszczególnych czujników. Ponadto zastosowanie autorskiego interfejsu komunikacyjnego zapewnia wymianę danych z systemem zarządzającym autonomiczną platformą oraz jej algorytmami.

Modelowanie otoczenia na potrzeby symulatora może przebiegać z wykorzystaniem różnych metod takich jak projektowanie komputerowe bądź fotogrametria. Zastosowanie dodatkowych mechanizmów jak np. oświetlenie i programy cieniujące pozwala na uzyskanie wysokiego poziomu realizmu projektowanej sceny. Dzięki temu symulator w wierny sposób naśladuje zachowanie wirtualnych kamer. Dodatkowo otoczenie jest wzbogacane o właściwości fizyczne poprzez wprowadzenie modeli kolizyjnych obiektów, współczynników tarcia nawierzchni czy siły grawitacji.

Pojazd i jego wierne odwzorowanie w symulacji jest zasługą zastosowania modelu graficznego, który zapewnia wizualizację platformy oraz modelu fizycznego pozwalającego na działanie wirtualnych sensorów czy sterowanie platformą. WST umożliwia tworzenie i wykorzystanie dowolnej liczby modeli pojazdów umożliwiając wykonywanie różnorodnych testów. Zachowanie rzeczywistych rozmiarów platformy jezdnej pozwala na precyzyjne pozycjonowanie wirtualnych sensorów, umożliwiając tym samym dobór ich parametrów oraz punktów mocowania w celu znalezienia optymalnego rozwiązania.

Szeroka paleta wirtualnych sensorów jest najważniejszym elementem przygotowanego środowiska. Konfigurowalne modele takich sensorów jak LiDAR, kamera, IMU, GNSS czy odometria pozwalają na dobór odpowiednich czujników w zależności od wymagań. Modele symulacyjne wyposażono dodatkowo w możliwość dodawania błędów pomiarowych oraz cech charakterystycznych dla rzeczywistych czujników. Generowanie przez symulator precyzyjnych danych sensorycznych może posłużyć do testów poszczególnych algorytmów autonomii jazdy lub kompleksowej

weryfikacji rozwiązania przygotowanego dla platformy autonomicznej.

W ramach prac nad WST opracowano również interfejs komunikacyjny UDP Bridge stanowiący sposób wymiany danych pomiędzy środowiskiem ROS będącym głównym rdzeniem systemu autonomii, a symulatorem. Zastosowanie protokołu UDP oraz predefiniowanych ramek przesyłanych wiadomości zapewnia uniwersalność rozwiązania i najwyższą wydajność podczas wymiany dużej ilości informacji takich jak dane lidarowe czy obrazy z kamery.

Zagadnienia poruszone w tym rozdziale oraz wyniki przeprowadzonych prac zostały również opublikowane w następujących czasopismach:

- Sobczak Ł., Filus K., Domańska J., Domański A., Building a Real-Time Testing Platform for Unmanned Ground Vehicles with UDP Bridge, Sensors, 22 (21), 2022.
- Sobczak Ł., Filus K., Domański A., Domańska J., LiDAR Point Cloud Generation for SLAM Algorithm Evaluation, Sensors, 21 (10), 2021.

Rozdział 5

Analiza i ewaluacja algorytmów autonomii jazdy w środowisku symulacyjnym

Opracowanie symulatora na potrzeby doktoratu było kluczowe, aby w odpowiedni sposób przeprowadzić ocenę zasadności wykorzystania symulacji komputerowej w pracach badawczo-rozwojowych nad systemami autonomicznej jazdy oraz dokonać ewaluacji określonych algorytmów autonomii. W niniejszym rozdziale opisane zostały wybrane zagadnienia i problemy, których rozwiązania zostały opracowane z wykorzystaniem Wirtualnego Systemu Testującego. Uzyskane wyniki poddano ocenie i weryfikacji z użyciem środowiska symulacyjnego.

5.1 Wstępne przetwarzanie danych sensorycznych

Jednym z podstawowych założeń opracowywanego środowiska testowego było symulowanie danych sensorycznych w taki sposób, aby możliwe było opracowywanie algorytmów operujących na każdym etapie działania systemów autonomii jazdy. Pierwszym z takich etapów jest wstępne przetwarzanie danych sensorycznych (z ang. *preprocessing*) w celu uzyskania bardziej wiarygodnych odczytów mogących być przedmiotem dalszego przetwarzania. Tego typu przetwarzanie jest niezbędne,

aby zapewnić odpowiedniej jakości informacje wejściowe dla algorytmów opisanych w sekcji 2.3.

Generowanie wirtualnych odczytów lidarowych o wysokiej zgodności z rzeczywistością pozwala m.in. na przygotowanie implementacji oraz testy algorytmów filtracji uzyskiwanych chmur punktów. Opracowywane z wykorzystaniem symulatora algorytmy mogą być z powodzeniem stosowane w środowisku produkcyjnym dzięki regulacji stopnia i charakterystyki zaszumienia syntetycznych danych oraz zapewnieniu formatu danych zgodnego z rzeczywistym urządzeniem.

W ramach prac przygotowano m.in. filtr dolnoprzepustowy i górnoprzepustowy dla czujnika LiDAR służący do odrzucania pomiarów, których odczyt sięga poniżej lub powyżej danej wartości progowej dobranej w zależności od specyfikacji danego urządzenia i jego właściwości. Dzięki symulacji błędów pomiarowych w Wirtualnym Systemie Testującym możliwe było zaobserwowanie, w czasie rzeczywistym, skutków działania implementowanego algorytmu, ułatwiając tym samym jego proces tworzenia i testowania. Podobne korzyści zaobserwowano podczas opracowywania filtru wysokości działającego w chmurze punktów z lidaru 3D. W tym przypadku, dysponując wirtualnym modelem pojazdu z rzeczywistym umiejscowieniem czujnika oraz wizualizacją pomieszczenia, wraz z konkretnymi obiektami w przestrzeni, dobór parametrów i weryfikacja działania znacząco upraszcza implementację algorytmu, szczególnie w przypadku ograniczonej dostępności platformy i rzeczywistych sensorów.

Dokładne odwzorowanie w symulatorze pojazdu oraz położenia sensorów umożliwia również wykonanie akumulacji danych z wielu lidarów do wspólnej chmury punktów celem dalszych obliczeń. Dzięki zachowaniu zbliżonych parametrów lidarów do rzeczywistych urządzeń można tego dokonać korzystając jedynie ze środowiska symulacyjnego, co upraszcza i przyspiesza docelowe wdrożenie rozwiązania na fizyczny pojazd. Umożliwia to również zweryfikowanie wytypowanych punktów montażowych poszczególnych czujników przed finalnym umieszczeniem ich na platformie.

Do zakresu wstępnego przetwarzania danych sensorycznych można zaliczyć rów-

nież kalibrację czujników. Wirtualny System Testujący może więc zostać wykorzystany do przetestowania procedur kalibracyjnych poszczególnych sensorów. W ramach pracy wielokrotnie przeprowadzano kalibrację kamery w celu uzyskania parametrów wewnętrznych, takich jak macierz kamery czy współczynniki dystorsji. Odpowiednio przygotowana kalibracja była wykonywana w symulacji w celu weryfikacji jej poprawności oraz dalszej analizy uzyskanych parametrów, wykorzystywanych np. w algorytmach SLAM opierających się na danych z kamer mono- lub stereowizyjnych.

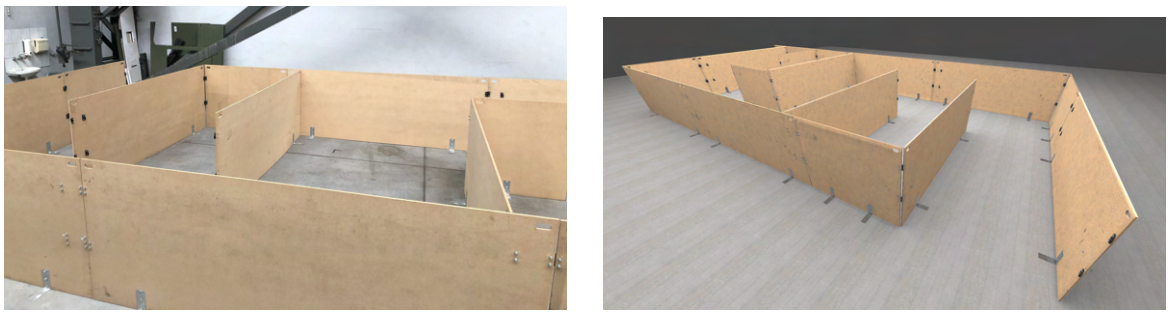
5.2 Ewaluacja algorytmów SLAM

Kolejnym zastosowaniem opracowanego symulatora jest możliwość ewaluacji algorytmów SLAM. Dzięki generowaniu danych syntetycznych o parametrach zbliżonych do rzeczywistych, możliwe jest przeprowadzanie oceny skuteczności algorytmu lokalizacji. To z kolei pozwala na selekcję optymalnych parametrów czy konfiguracji sprzętowych w określonych warunkach środowiskowych lub przy wykorzystaniu konkretnych typów czujników.

W ramach pracy przeanalizowano zagadnienia i metody służące generowaniu wiarygodnych chmur punktów (z ang. *point clouds*), które mogą służyć do ewaluacji algorytmów SLAM. Istotne dla stworzenia dokładnej symulacji jest zawarcie błędów fizycznego urządzenia w syntetycznych danych. Porównanie danych uzyskanych z rzeczywistego sensora z zaszumionymi danymi symulacyjnymi pokazuje, że podobieństwo znacząco wzrasta po uwzględnieniu błędów fizycznego urządzenia. W przypadku czujnika LiDAR dodatkowo widoczne jest to przy imitowaniu tzw. efektu *rolling shutter*. Uwzględnienie tego efektu w symulacji powoduje zwiększenie dokładności lokalizacji wyznaczanej za pomocą algorytmu SLAM. Podobny wpływ można zauważyć w przypadku uwzględnienia błędów poszczególnych odczytów odległości. Przedstawione w niniejszej sekcji wyniki pokazują, że tego typu próby prowadzące do upodobnienia syntetycznych danych, do danych rzeczywistych, są kluczowe w procesie ewaluacji algorytmów.

W procesie oceny działania algorytmu SLAM posłużono się implementacją o nazwie Cartographer [115]. Danymi wejściowymi były chmury punktów z lidaru oraz dane z czujnika IMU. Eksperymenty przeprowadzono na dwóch torach testowych wykonując wielokrotne przejazdy i uśredniając wyniki. Tor nr 1 obejmował przejazd w pomieszczeniu laboratoryjnym (przedstawionym na rys. 5.2) na odcinku 4 metrów z pięcioma punktami pomiarowymi umieszczonymi co 1 metr. Natomiast drugi tor był krótkim labiryntem o wymiarach ok. 7 metrów na 5 metrów z sześcioma punktami pomiarowymi. Oba obiekty zostały odwzorowane w środowisku symulacyjnym.

W pierwszej kolejności przeprowadzono porównanie dokładności symulacji chmury punktów 3D z rzeczywistymi pomiarami w celu zweryfikowania dokładności syntetycznie generowanych danych. Algorytm opracowany w tym celu przedstawiono na rys. 5.4. W pierwszym kroku usuwane są nieznaczące punkty, które w przypadku testowej trasy w labiryncie są punktami znajdującymi się powyżej określonej wysokości. Dla każdego z wygenerowanych punktów znajdowane są odpowiadające punkty w zbiorze uzyskanym z rzeczywistego urządzenia, a następnie obliczane średnie odległości pomiędzy odpowiadającymi punktami. Całość operacji wykonywana jest w każdym punkcie kontrolnym, a wyniki są uśredniane. Otrzymana wartość jest ostateczną miarą, która opisuje średnią różnicę pomiędzy pomiarami. Na wykresie 5.3 przedstawiono średni błąd poszczególnych rzeczywistych chmur punktów i ich symulowanych odpowiedników. Analizując uzyskane dane można zauważyć, że błąd jest niewielki (rzędu kilku milimetrów). Ponadto wyraźnie widać, że zaszumienie

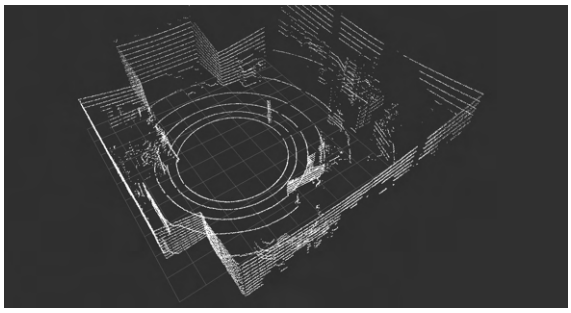


(a) Zdjęcie rzeczywistego toru. (b) Zrzut ekranu z symulacji.

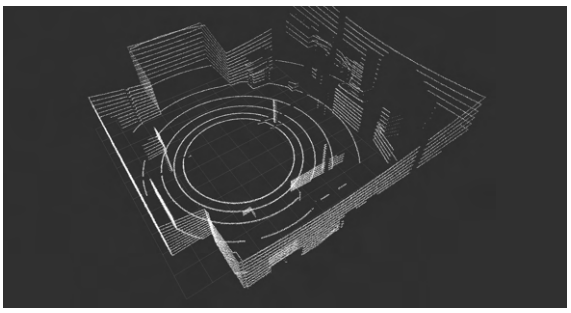
Rysunek 5.1: Tor testowy - labirynt - rzeczywisty i symulacyjny.

(a) Zdjęcie laboratorium.

(b) Zrzut ekranu symulowanego laboratorium w WST.



(c) Chmura punktów z lidaru Velodyne VLP-16.



(d) Chmura punktów uzyskana w symulacji.

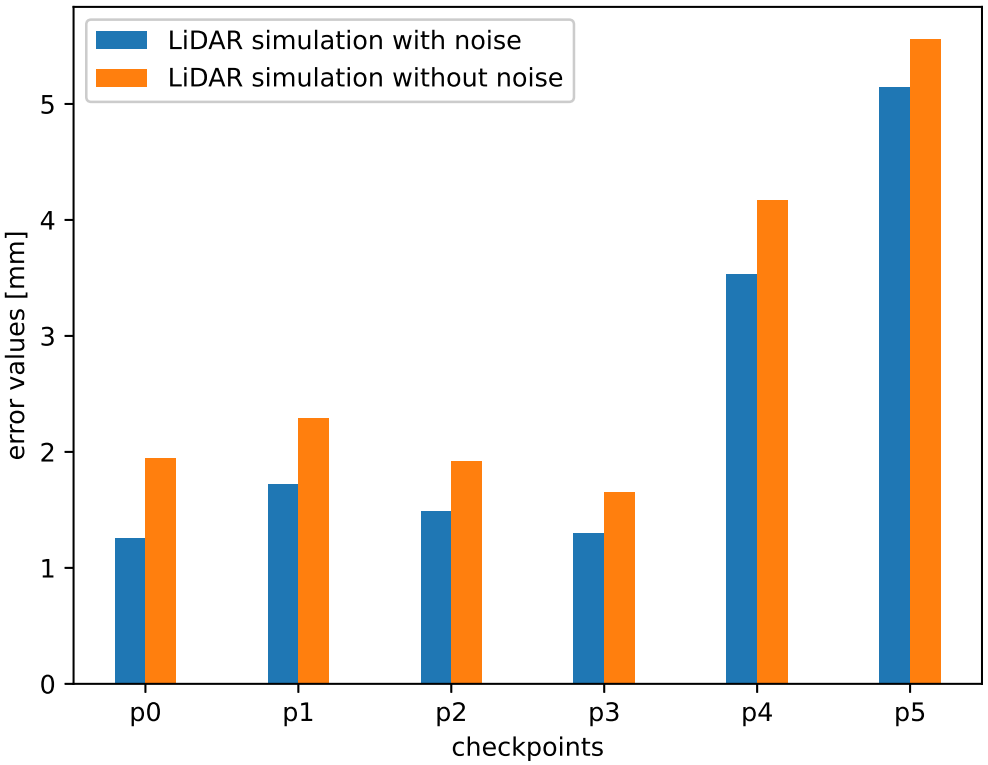
Rysunek 5.2: Przykładowa chmura punktów uzyskana z rzeczywistego urządzenia oraz ze środowiska symulacji.

danych i obarczenie ich dodatkowym błędem pomiarowym zwiększa dokładność.

Celem wyznaczenia błędu lokalizacji w przestrzeni obliczana jest suma kwadratów odległości pomiędzy każdą oszacowaną pozycją p oraz punktem referencyjnym p' według następującego wzoru [119]:

$$\epsilon(p_{1:N}) = \sum_{i=1}^{N} (p_i \ominus p'_i)^2 \tag{5.1}$$

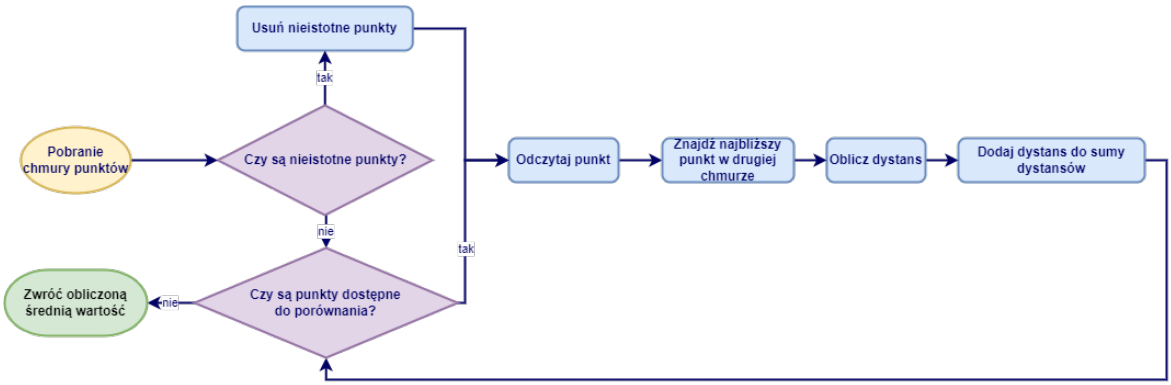
Na wykresie 5.5a przedstawiono błąd lokalizacji w kilku wybranych punktach pomiarowych testowej trasy. Dla tego przypadku dane symulacyjne przekazywane do algorytmu SLAM odpowiadały idealnemu modelowi czujnika LiDAR. Na wykresie można zauważyć wyraźną różnicę poziomów uzyskiwanych błędów oraz różne nachylenie linii trendu kolejnych błędów, co pozwala stwierdzić, że dane syntetyczne nie odpowiadają danym rzeczywistym w stopniu wystarczającym do prawidłowej weryfikacji działania algorytmu SLAM za pomocą symulatora. Wykres 5.5b za-



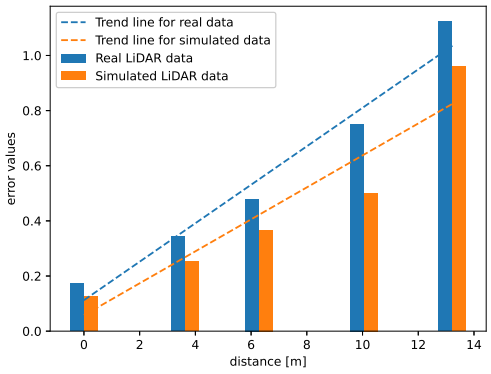
Rysunek 5.3: Średni błąd pomiędzy punktami dla rzeczywistej i symulowanej chmury punktów.

wiera dane syntetyczne uzupełnione o błąd pomiarowy każdego kolejnego odczytu oraz symulację efektu *rolling shutter* dodatkowo zakłócającego pomiary. W tym przypadku poziom uzyskiwanych błędów, jak również nachylenie linii trendu, jest zbliżone, co potwierdza zasadność konieczności uwzględniania błędów w syntetycznych danych wraz z przyjętą metodą. Wykres 5.6 przedstawia wzrost błędu w czasie podczas dłuższej trasy testowej, uwidaczniając tym samym rozbieżności pomiędzy symulowanymi danymi z sensora idealnego oraz takimi obarczonymi zakłóceniami podobnymi do rzeczywistego urządzenia.

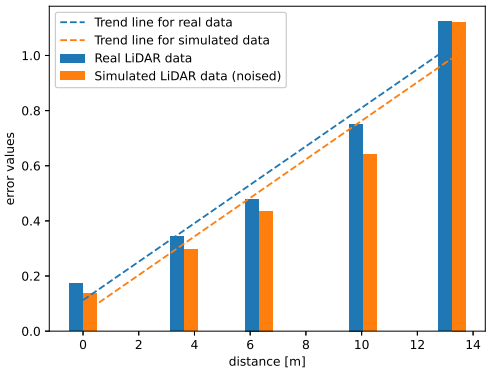
Dysponując symulatorem generującym wiarygodne dane wykorzystywane na potrzeby ustalania pozycji w wirtualnym, bądź rzeczywistym świecie, kolejnym zastosowaniem symulacji w kontekście algorytmów SLAM jest ewaluacja algorytmów i optymalizacja sensorów wykorzystywanych do wyznaczania lokalizacji. W ramach



Rysunek 5.4: Algorytm wykorzystywany do oceny dokładności dwóch chmur punktów 3D.



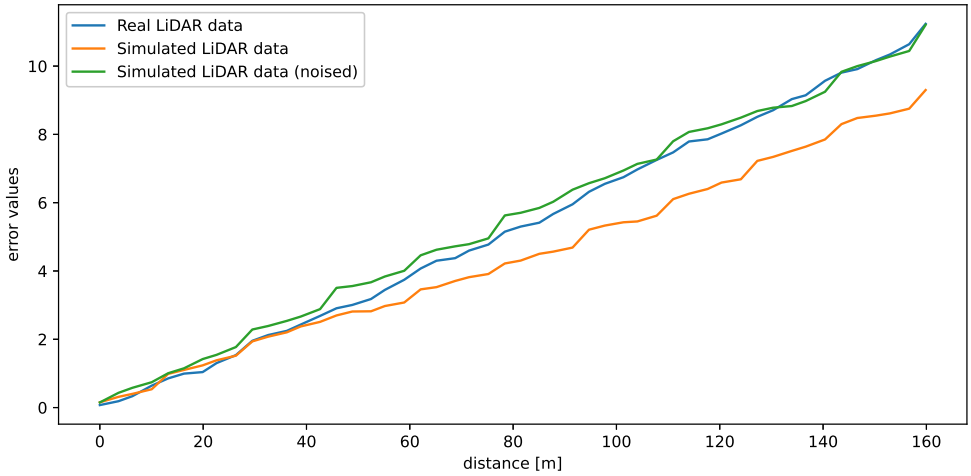
(a) Dane symulacyjne idealne.



(b) Dane symulacyjne obarczone błędami pomiarowymi oraz efektem *rolling shutter*.

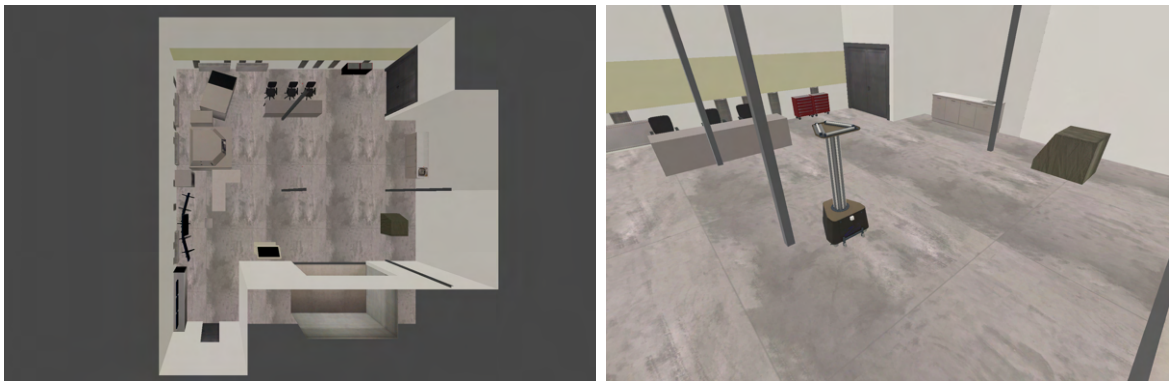
Rysunek 5.5: Dokładność SLAM dla danych rzeczywistych i symulowanych.

prac dokonano, przy wykorzystaniu opracowanego środowiska symulacyjnego, analizy i porównania wielu konfiguracji sprzętowych na podstawie implementacji wybranego algorytmu SLAM. Umożliwia on wykonywanie lokalizacji i mapowania zarówno w trybie 2D, jak i 3D, podczas której głównym źródłem danych jest czujnik laserowy typu LiDAR 2D lub 3D. Dodatkowo algorytm może być wspierany przez odczyty z odometrii, której źródłem są enkodery umieszczone w kołach pojazdu. Istotną rolę w dokładniejszym ustalaniu pozycji odgrywa również czujnik IMU, który zwraca informację o liniowych przyspieszeniach działających na pojazd oraz prędkościach kątowych. W tym przypadku dokonano pomiarów wykorzystując dwa wirtualne modele pomieszczeń obejmujących laboratorium (rys. 5.7) będące śred-



Rysunek 5.6: Dokładność SLAM dla danych rzeczywistych i symulacyjnych na całym odcinku pokonanej trasy testowej.

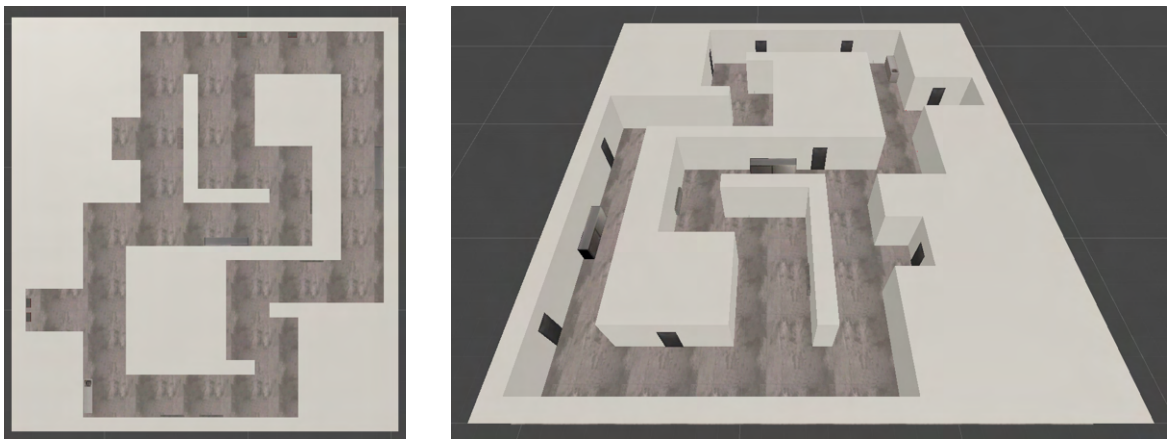
niego rozmiaru pomieszczeniem oraz długi, kręty korytarz (rys. 5.8) przypominający labirynt. W obu przypadkach skupiono się na zbadaniu jakości pozycjonowania w pomieszczeniach zamkniętych w trybie działania 2D (pomijając obliczenia zmian położenia w osi pionowej).



(a) (b)

Rysunek 5.7: Symulowane pomieszczenie laboratorium.

Przeprowadzone eksperymenty polegały na porównaniu jakości budowanych map przez algorytm oraz przede wszystkim na zbadaniu dokładności wyznaczonej trajektorii ruchu pojazdu, w odniesieniu do wartości referencyjnych (z ang. *ground truth*), które dostarcza symulator. Do tego celu użyto dwóch standardowych metryk oceny



(a) (b)

Rysunek 5.8: Symulowany korytarz.

dokładności SLAM: względny błąd pozycji (z ang. *Relative Pose Error - RPE*) oraz całkowity błąd trajektorii (z ang. *Absolute Trajectory Error - ATE*) [120]. *ATE* ocenia różnicę między pozycją wyznaczoną, a rzeczywistą, podczas gdy miara *RPE* jest określana na podstawie różnicy między oszacowanym, a rzeczywistym ruchem w przestrzeni. Ruch można wyrazić za pomocą zbioru pozycji rzeczywistej trajektorii $Q_1, Q_2, ..., Q_n \in SE(3)$ i tej oszacowanej $P_1, P_2, ..., P_n \in SE(3)$, gdzie n jest liczbą wyznaczonych pozycji. Z kolei $SE(3)$ można wyrazić parą (R, t), w której ($R \in SO(3)$, $t \in \mathbb{R}^3$). Natomiast macierz homogeniczna M może być wykorzystywana do reprezentacji danej transformacji w następujący sposób:

$$M = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \tag{5.2}$$

Całkowity błąd trajektorii *ATE* porównuje całkowite odległości między rzeczywistą, a wyznaczoną trajektorią. Trajektorie te mogą być zdefiniowane w różnych układach współrzędnych, co sprawia, że niezbędnym jest ich przeskalowanie za pomocą transformacji S przedstawionej w metodzie Horna [121]. Wyznaczenie *ATE* w danej chwili czasu i może być zdefiniowane jako:

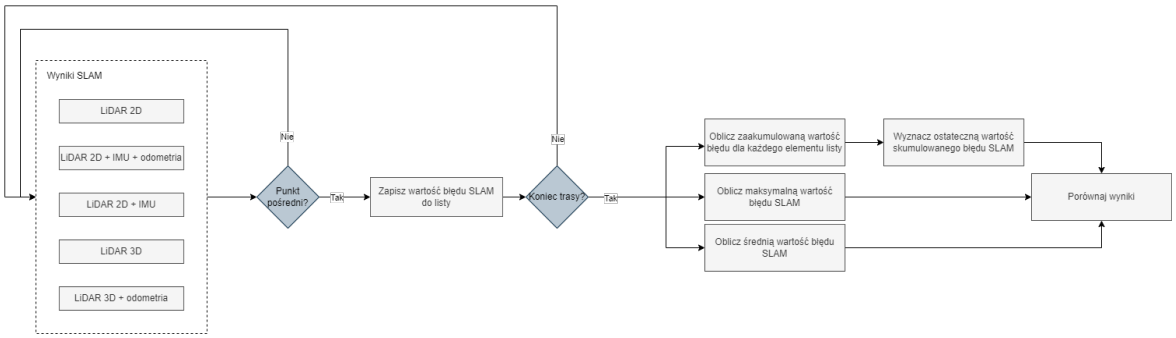
$$F_i = Q_i^{-1} S P_i \tag{5.3}$$

Następnie, korzystając ze składowej translacji macierzy $trans(F_i)$, można wyznaczyć statystyki danej normy $||trans(F_i)||$, takie jak wartość średnia czy maksymalna.

Natomiast względny błąd pozycji *RPE* określa lokalną dokładność trajektorii w danym oknie czasowym Δ. Wartość ta opisuje odchylenie trajektorii i może być wykorzystana np. do oceny dokładności SLAM podczas operacji tzw. domknięcia pętli (z ang. *loop closure*). *RPE* w danej chwili czasu i można zdefiniować jako:

$$E_i = (Q_i^{-1} Q_{i+\Delta})^{-1} (P_i^{-1} P_{i+\Delta}) \tag{5.4}$$

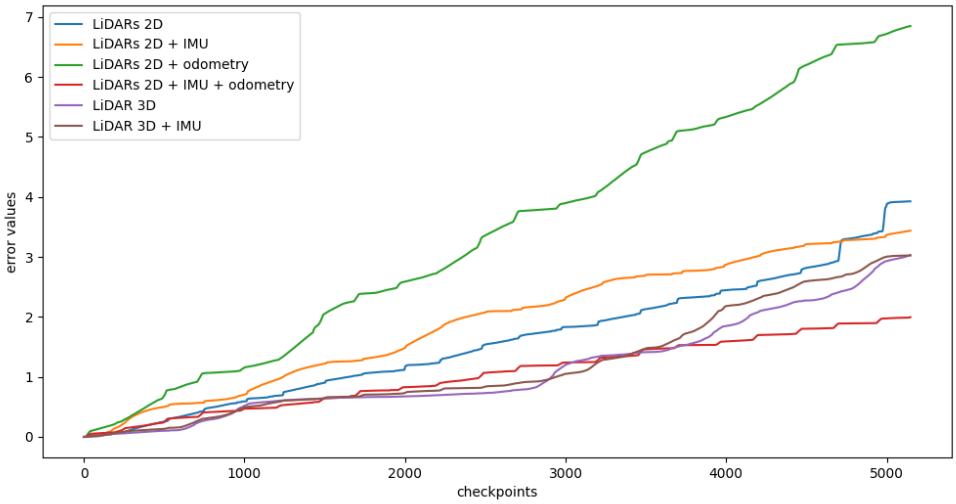
W tym przypadku *RPE* można wyznaczyć zarówno dla składowej translacji, jak i rotacji. W omawianym przykładzie wykorzystano jedynie część translacji $trans(E_i)$, co w większości przypadków jest wystarczające [120] do efektywnego przedstawienia statystyk normy $||trans(E_i)||$, takich jak średnia czy maksimum.



Rysunek 5.9: Metodologia oceny jakości lokalizacji.

Bazując na uzyskanych wynikach obliczanych w przedstawiony sposób, wyznaczono szereg statystyk pomiarów *ATE* i *RPE* takich jak średnie, maksymalne czy końcowe wartości dokładności lokalizacji dla obu testowych przejazdów. Na rysunku 5.9 przedstawiono metodologię wykorzystywaną do wyznaczania jakości pozycjonowania reprezentowanej przez *ATE*.

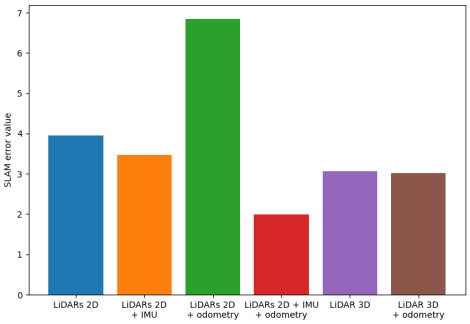
Wyniki uzyskano porównując działanie SLAM i wyznaczając wartości *ATE* i *RPE* dla 8 konfiguracji sprzętowych: zestawu lidarów 2D oraz pojedynczego lidaru 3D, a także dołączając dodatkowo odczyty z IMU i/lub odometrii. Wykres 5.10 przed-



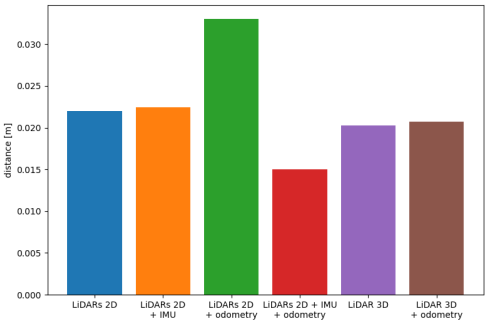
Rysunek 5.10: Zmiana wartości błędu SLAM w czasie.

stawia wzrost błędu SLAM w czasie dla każdej z badanych konfiguracji sensorów. Ponadto wyznaczono całkowitą, średnią oraz maksymalną wartość błędu dla wszystkich konfiguracji sprzętowych przeznaczonych dla algorytmu SLAM co zaprezentowano na wykresach 5.11.

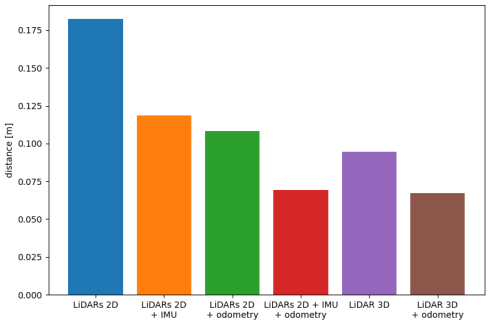
Analizując wiele uzyskanych wyników pod względem dokładności lokalizacji, najbardziej odpowiednią konfiguracją dla pomieszczeń średniej wielkości takich jak np. laboratoria, sale szpitalne czy przestrzenie biurowe okazał się zestaw lidarów 2D wspomaganych poprzez IMU oraz odometrię. W przypadku dłuższych korytarzy zastosowanie lidaru 3D pozwalało uzyskać nieco lepsze wartości *ATE* przy jednocześnie gorszych wynikach dla estymacji lokalnej trajektorii w porównaniu do wyników uzyskanych dla konfiguracji z lidarami 2D, IMU i odometrią. Biorąc pod uwagę uzyskane wyniki oraz fakt, że tego typu konfiguracja jest bardziej efektywna również pod względem kosztowym, została ona wybrana jako wystarczająca dla robota poruszającego się wewnątrz pomieszczeń o określonej specyfikacji. Powyższe wyliczenia i wnioski zostały uzyskane jedynie z wykorzystaniem opracowanego środowiska symulacyjnego, a ich słuszność została dodatkowo potwierdzona przy wdrożeniu wytypowanej konfiguracji na rzeczywistym robocie mobilnym.



(a)



(b)



(c)

Rysunek 5.11: Wykresy przedstawiające: (a) całkowity, (b) średni oraz (c) maksymalny błąd dla różnych konfiguracji algorytmu SLAM.

5.3 Dobór czujników dla algorytmów SLAM

W ramach ewaluacji algorytmów SLAM przedstawiono metodologię oceny i porównania algorytmów bazujących na danych z czujników LiDAR, IMU i odometrii. Wykorzystując opracowane w ramach doktoratu środowisko symulacyjne zaproponowano procedurę testową pozwalającą na znalezienie najlepszej konfiguracji czujników dla algorytmów lokalizacji i mapowania. Przedstawiono również wady i zalety takiego podejścia. Poniżej omówiono 7 głównych kroków prowadzących do prawidłowego doboru czujników w środowisku symulacyjnym.

1. Określenie wymagań

   Pierwszym krokiem jest specyfikacja wymagań oraz zebranie informacji o środowisku działania i przypadkach użycia danego pojazdu. Należy określić roz-

miar i typ otoczenia (np. małe laboratorium, duży park) oraz jego najbardziej charakterystyczne obiekty takie jak np. meble, elementy architektoniczne czy też roślinność. W przypadku pojazdu istotne jest posiadanie modelu 3D pojazdu w rzeczywistej skali, szczegółów dotyczących zastosowanego napędu i jego parametrów, a także potencjalnych punktów montażowych sensorów. Na potrzeby porównań danych czujników niezbędne jest również określenie zbioru urządzeń wraz z ich parametrami.

2. Zdefiniowanie metryk
   Metryki do oceny SLAM można podzielić na ilościowe (ATE, RPE) i jakościowe (jakość generowanej mapy, odwzorowanie charakterystycznych punktów). Dla metryk ilościowych warto również wyznaczyć zbiór statystyk (średnia, minimum, maksimum), które pozwolą na dokładniejsze i bardziej przejrzyste przedstawienie wyników.

3. Budowa środowiska testowego
   Bazując na określonych wymaganiach należy przygotować odpowiednie modele środowiska, pojazdu oraz dostępnych czujników. Następnie należy zapewnić połączenie pomiędzy symulatorem i dostarczanymi przez niego danymi sensorycznymi, a algorytmem, czy całym systemem podlegającym testom. W przypadku WST opracowanego w ramach niniejszej pracy zastosowano UDP Bridge opisany w sekcji 4.5. Dodatkowo pomocne może okazać się opracowanie metody odczytywania referencyjnych danych, tzw. *ground truth*, bezpośrednio z symulacji oraz wizualizacja, graficzne przedstawienie i podgląd uzyskiwanych danych.

4. Zdefiniowanie trasy przejazdu
   Aby móc poprawnie zweryfikować działanie algorytmu SLAM ważne jest odpowiednie zdefiniowanie drogi, po której powinien poruszać się pojazd w symulacji, aby osiągnąć wszystkie punkty charakterystyczne wirtualnego środowiska oraz zapewnić odpowiednią liczbę pomiarów dla uzyskania wiarygodnych wyników.

5. Pomiary

   Właściwe przeprowadzenie pomiarów wymaga pokonywania zdefiniowanej ścieżki dla każdej konfiguracji sensorów, zarówno dla różnych modeli czujników, jak i punktów ich mocowania na pojeździe. Należy zapisywać uzyskiwane pomiary wraz z odpowiadającymi im danymi referencyjnymi, a także dodatkowymi wynikami, np. stworzoną mapą otoczenia.

6. Przygotowanie wyników

   W tym kroku należy przeprowadzić analizę uzyskanych wyników i określić wartości błędów pozycjonowania oraz zwizualizować wygenerowane mapy otoczenia i przeanalizować ich ogólną jakość zgodnie ze zdefiniowanymi kryteriami.

7. Porównanie wyników

   Ostatecznie uzyskane wyniki mogą posłużyć do bezpośredniego porównania danych konfiguracji sprzętowych sensorów i wyłonienia najlepszego rozwiązania w założonym scenariuszu użycia, biorąc również pod uwagę dodatkowe czynniki takie jak np. koszt danych sensorów czy narażenie na awarię podczas eksploatacji.

Ostatnim, dodatkowym krokiem przeprowadzanym na dalszych etapach projektu pojazdu autonomicznego powinna być weryfikacja wybranej platformy sprzętowej na rzeczywistym robocie i potwierdzenie poprawności wyników uzyskanych w symulacji.

Opisane podejście do przeprowadzenia testów w środowisku symulacyjnym zapewnia szereg korzyści, ale i ma swoje ograniczenia. Do niewątpliwych zalet należy zaliczyć łatwość automatyzacji testów, ich powtarzalność i zapewnienie stałych warunków środowiskowych, dostęp do poprawnych danych referencyjnych czy efektywność kosztowa, zważywszy na brak potrzeby nadmiarowego zakupu dodatkowych sensorów przeznaczonych jedynie do testów. Natomiast wśród wad tego typu testowania algorytmów SLAM w symulatorze można wyróżnić złożony i czasochłonny proces tworzenia środowiska wirtualnego odwzorowującego rzeczywisty obszar, jak również stosunkowo niedokładne symulacje fizyki pojazdu i interakcji z otoczeniem,

które pomimo coraz dokładniejszych technik i wyższej wydajności nie zastępują w pełni testów docelowej platformy w warunkach rzeczywistych.

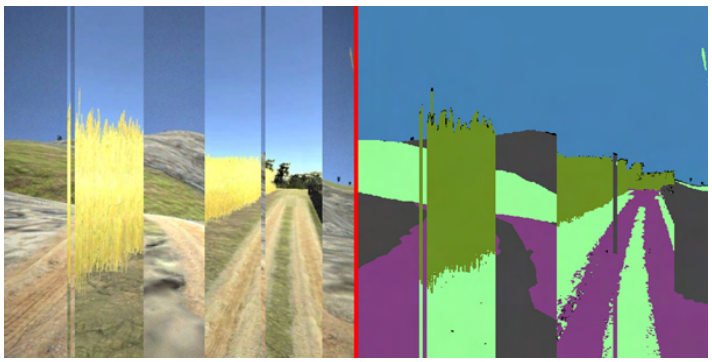
5.4 Tworzenie danych treningowych

Dysponowanie pojazdem wyposażonym w szereg czujników, odwzorowanym w wirtualnym świecie, niesie za sobą szereg korzyści, nie tylko przy testowaniu opracowywanych rozwiązań, ale może wspierać również proces tworzenia danych treningowych dla zaawansowanych algorytmów sztucznej inteligencji. Jak opisano w sekcji 4.4.4, w WST zaimplementowano wirtualny model kamery generującej kolorową mapę semantyczną wraz z obrazem RGB stanowiącą parę treningową dla głębokich sieci neuronowych dokonujących segmentacji semantycznej obrazu.

Rysunek 5.12: Zestaw obrazów treningowych wygenerowanych w WST zawierających obraz z wirtualnej kamery, mapę semantyczną oraz wizualizację danych.

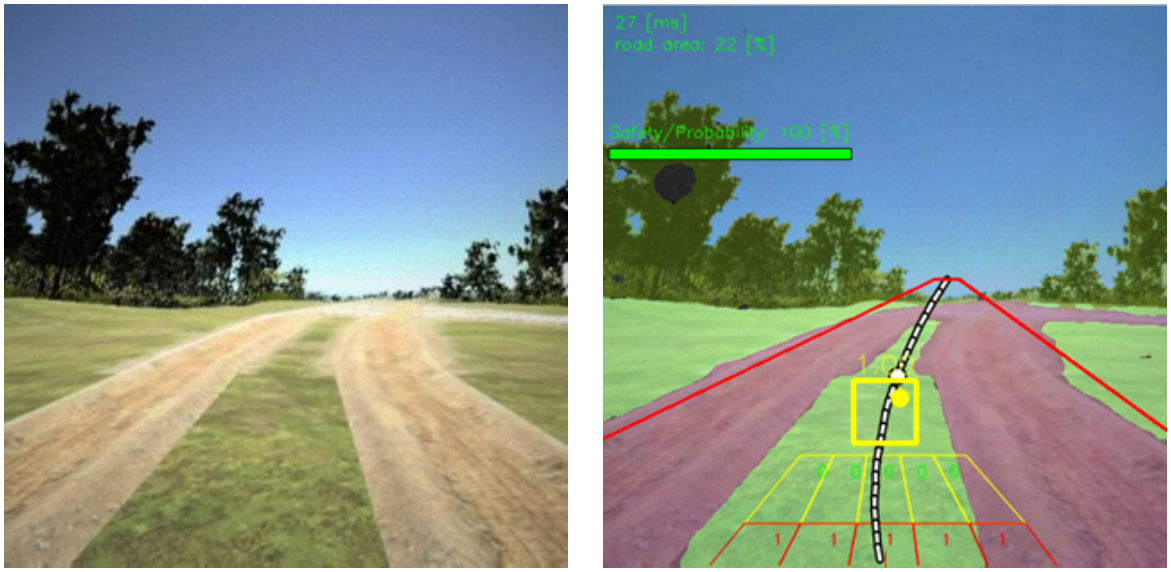
Tego typu syntetyczne obrazy stanowią dobrej jakości uzupełnienie rzeczywistych danych treningowych, oferując przy tym możliwość automatyzacji generowania i oznaczania danych. Umożliwia to tym samym uzyskanie znacznie większej ilości danych, niż w przypadku danych rzeczywistych, których odpowiednie oznaczenie wymaga długiej i żmudnej pracy. Wygenerowane w ten sposób dane zostały wykorzystane do treningu konwolucyjnej sieci neuronowej BiSeNet [122] w celu detekcji na obrazie obszarów przejezdnych na polnej, nieutwardzonej drodze. Zbiór danych treningowych i walidacyjnych został powiększony o syntetyczne dane pochodzące z symulatora, co zapewniło większą skuteczność działania sieci neuronowej podczas przejazdów w witualnym świecie, nie powodując przy tym obniżonej sku-

teczności segmentacji na rzeczywistych obrazach. Na rys. 5.12 przedstawiono przykład wygenerowanych danych treningowych dla głębokiej sieci neuronowej zdolnej do segmentacji semantycznej obrazu zawierający obraz z wirtualnej kamery, mapę semantyczną zawierającą kilka predefiniowanych klas jak np. droga, roślinność czy niebo oraz połączenie tych danych dla lepszej wizualizacji. Ponadto oprogramowanie symulatora potrafi wygenerować rozszerzony zbiór danych zawierających nienaturalne obrazy uzyskane za pomocą ich przekształceń jak np. pokazane na rys. 5.13, co zwiększa zbiór treningowy i potencjalnie może umożliwić uzyskanie lepszych wyników segmentacji obrazów. Trening wspomnianej sieci BiSeNet pozwolił na uzyskanie zadowalających wyników realizując tym samym główną funkcję jaką była detekcja drogi na obrazie. Zaimplementowano algorytm utrzymywania na pasie drogi z wykorzystaniem metody *Vanishing Point* [123]. Przykładową detekcję drogi za pomocą nauczonej sieci przedstawiono na rys. 5.14b wraz z algorytmem utrzymywania jazdy wzdłuż wykrytego obszaru przejezdnego.



Rysunek 5.13: Augmentacja danych treningowych wygenerowanych w WST.

Kolejnym zaimplementowanym mechanizmem w opracowanym symulatorze jest symulacja detekcji obiektów na wirtualnym obrazie z kamery. Moduł ten dostarcza informacji o położeniu wybranych obiektów na wygenerowanej klatce obrazu, dzięki czemu może posłużyć do zautomatyzowanego tworzenia zbiorów treningowych sieci do wykrywania i klasyfikacji obiektów takich jak np. YOLO [48] czy R-CNN [124]. Ten element symulatora został wykorzystany przy opracowywaniu algorytmu do podążania za wskazanym obiektem. Dzięki symulacji położenia obiektu na obrazie z kamery oraz odległości od sensora odpowiedni algorytm sterowania kieruje pojazdem utrzymując daną odległość przed pojazdem. Dane te na rzeczywistym



(a) (b)

Rysunek 5.14: Obraz z wirtualnej kamery oraz wynik działania segmentacji semantycznej wraz z algorytmem utrzymywania na pasie drogi.

pojeździe dostarcza kamera głębi 3D wraz z konwolucyjną siecią neuronową wykrywającą i klasyfikującą obiekty. Wykorzystanie symulatora pozwoliło w tym przypadku na implementację algorytmu bez uciążliwych testów w terenie, szczególnie przy ograniczonym dostępie do platformy, na której system miał być uruchomiony.

5.5 Analiza komunikacji symulatora z systemem autonomii

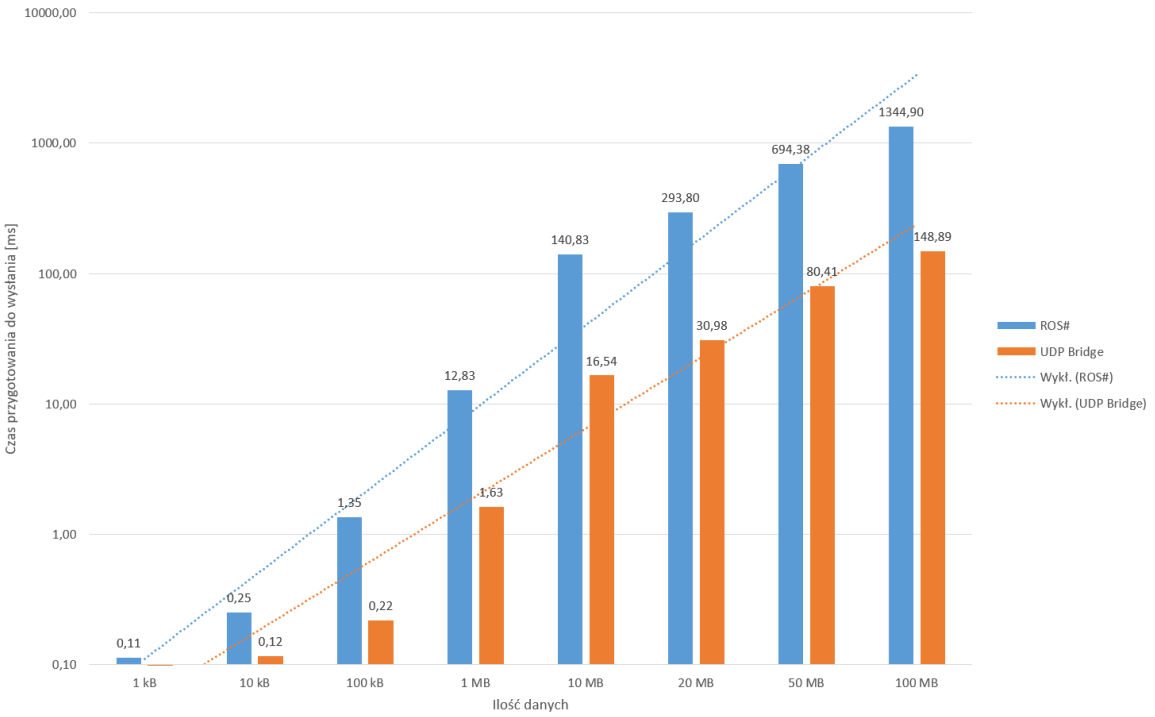
Jedną z głównych zalet opracowanego UDP Bridge jest niewielki czas potrzebny do wysłania wiadomości z symulatora do ROS. Na rys. 5.15 przedstawiono wykres porównujący wpływ ilości danych do wysłania na czas potrzebny do nadania tych danych za pośrednictwem ROS# oraz UDP Bridge. Jako czas wymagany do nadania danych określa się czas, w którym wykonywane są operacje alokacji i inicjalizacji struktur danych, odświeżenia nagłówków wiadomości i dodatkowych metadanych oraz operacje przesyłu danych za pośrednictwem wybranego protokołu, którego wykonanie jest wymagane przed rozpoczęciem nadawania kolejnej wiadomości, co

sprawia, że działania te nie mogą zostać zrównoleglone. Zarówno dla małej, jak i dużej ilości danych występuje zauważalna różnica na korzyść autorskiego rozwiązania. Przy przesyle do 100 kB danych proces wysłania danych trwa maksymalnie 1.5 milisekundy przy zastosowaniu ROS# oraz ok. 1/4 milisekundy wykorzystując UDP Bridge. Mimo wyraźnej, 6-krotnej różnicy w czasie wykonywania, obie wartości są akceptowalne nie wpływając znacząco na wydajność symulacji większości wirtualnych sensorów. Niemniej jednak wysyłanie wiadomości o rozmiarze powyżej 1 MB stanowi dla ROS# poważne wyzwanie, oferując od 8- do niemal 10-krotnie gorszą wydajność względem UDP Bridge. Takie opóźnienia w znaczący sposób ograniczają pracę WST, co zauważalne było przy symulacji takich czujników jak wszelkiego rodzaju kamery czy lidary 3D. Fakt generowania nawet 10 MB danych i czas wymagany na przesyłanie danych do systemu sprawiał, że niemożliwe było zapewnienie pożądanego odświeżania danych przy jednoczesnym działaniu innych wirtualnych czujników oraz złożonego środowiska 3D. Dlatego też zdecydowano się na implementację UDP Bridge, który rozwiązuje te problemy umożliwiając swobodną symulację wielu czujników i gwarantując przy tym aktualizację danych na poziomie odpowiadającym rzeczywistym sensorom.

Podsumowując - dla każdej ilości danych opracowany UDP Bridge pozwala na zdecydowanie szybsze wysłanie danych z symulacji do właściwego systemu autonomii. W wyraźny sposób widać zdecydowaną przewagę autorskiego rozwiązania transportu danych z symulatora wprost do systemu ROS. Zaprezentowane na wykresie dane są wyrażone w skali logarytmicznej, uwypuklając przy tym szybkość transmisji danych szczególnie w przypadku większej ilości informacji.

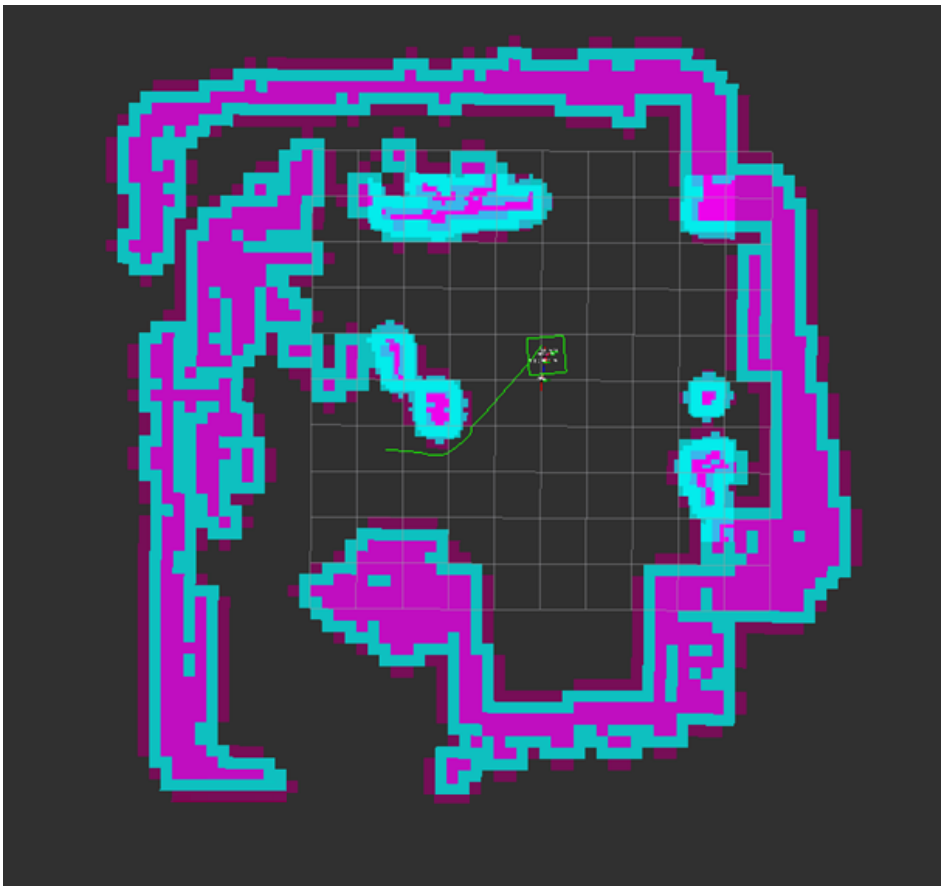
5.6 Przygotowanie algorytmów nawigacji

Stosując symulator do wyznaczania pozycji w przestrzeni, zrozumienia otaczającego świata oraz szeroko pojętych prac programistycznych wokół systemów autonomii, naturalnym pozostaje jego zastosowanie do opracowywania i testowania algorytmów nawigacji.



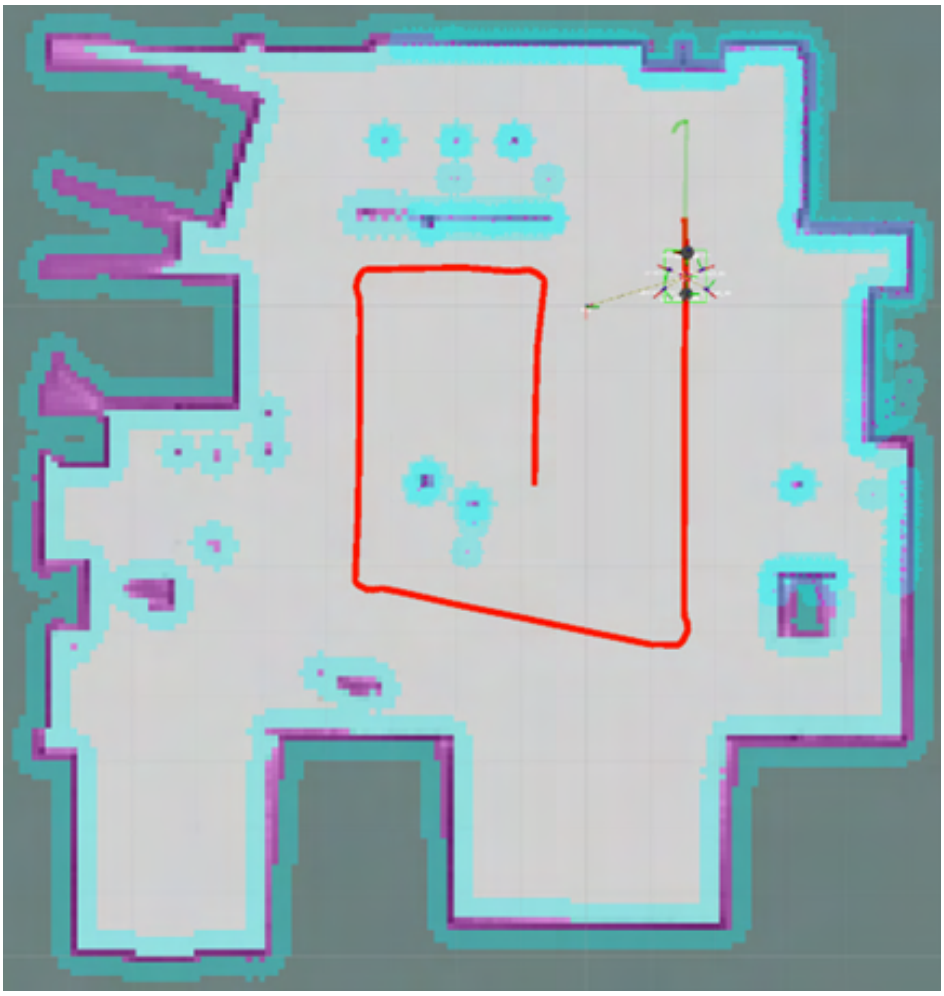
Rysunek 5.15: Wykres prezentujący w skali logarytmicznej zależność między ilością danych, a czasem wymaganym do przygotowania danych do wysłania.

W ramach opracowywanego systemu można wyróżnić kilka komponentów oprogramowania, które dzięki syntetycznym danym wygenerowanym w symulatorze pozwalają na zrealizowanie funkcjonalności autonomii jakim jest autonomiczny dojazd do wybranego punktu docelowego wraz z omijaniem przeszkód. Bazując na generowanej chmurze punktów z czujnika laserowego LiDAR możliwe jest tworzenie ogólnej mapy przeszkód będącej siatką zajętości terenu wokół pojazdu. Takie informacje pełnią istotną rolę przy wyznaczaniu trasy przejazdu i unikaniu oraz wymijaniu przeszkód pojawiających się na drodze. Dzięki zastosowaniu symulacji dobór parametrów mapy odbywa się w powtarzalnych warunkach, a rozdzielczość czy rozmiar może być w łatwy sposób wyznaczony eksperymentalnie. Na rys. 5.16 przedstawiono przykładową mapę wygenerowaną poprzez skanowanie otoczenia sensorem LiDAR w rzeczywistym pomieszczeniu laboratoryjnym. Rys. 5.17 przedstawia to samo pomieszczenie wraz z mapą przeszkód wygenerowaną w WST. Zawiera ona także dodatkowe informacje takie jak mapa wygenerowana przez algorytm SLAM, czy też predefiniowana trasa przejazdu.



Rysunek 5.16: Rzeczywista mapa przeszkód pomieszczenia laboratoryjnego.

W podobny sposób można przeprowadzić parametryzację tzw. plannerów, odpowiedzialnych za wyznaczenie trasy, dysponując danymi z siatki zajętości oraz pozycją i prędkością pojazdu. Pełna symulacja pojazdu pozwala na dostosowanie typu wyznaczania ścieżki do celu (np. algorytm Dijkstra lub A*), a także pozostałych parametrów dotyczących m.in. tolerancji w dążeniu do wyznaczonego celu, współczynników funkcji kosztu do optymalizacji planowania trasy, a także wartości odnoszących się do fizyki robota takich jak maksymalna czy minimalna prędkość, a nawet charakterystyka skręcania pojazdu, które to elementy są odwzorowane w modelu stosowanym w symulatorze i odpowiadają w dużym stopniu rzeczywistości. Zastosowanie symulacji umożliwia także weryfikację czy opracowywanie pozostałych komponentów nawigacji autonomicznej jak np. przejazd po predefiniowanej ścieżce pokazany na rys. 5.17 czy nawigacja oparta na nawigacji satelitarnej i punktach geograficznych.



Rysunek 5.17: Mapa przeszkód pomieszczenia laboratoryjnego wygenerowana na podstawie danych z WST wraz z predefiniowaną trasą przejazdu oraz dodatkową warstwą wizualizacji mapy.

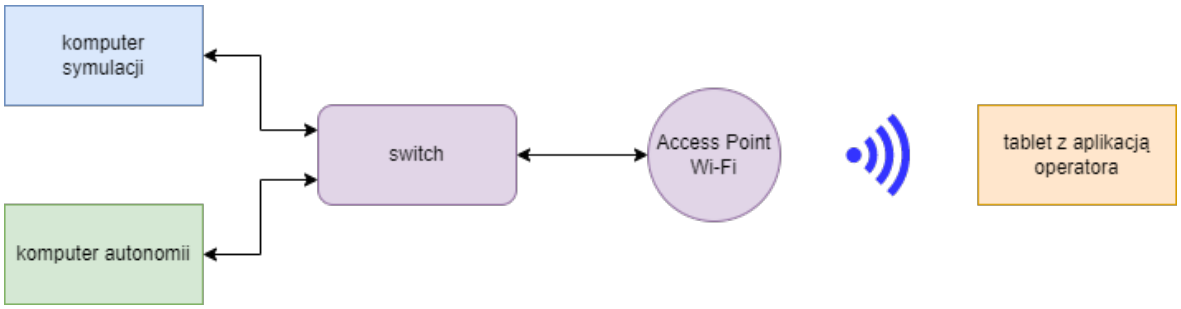
5.7 Opracowanie aplikacji operatora

Odpowiednie zarządzanie systemem autonomii i informowanie o jego stanie jest kluczowe dla bezpieczeństwa zautomatyzowanego przejazdu, nad którym powinien czuwać operator obserwujący postępy misji dzięki aplikacji zarządzającej danym pojazdem autonomicznym. Do realizacji tego typu celu zastosowano opracowany symulator, za pomocą którego zaprojektowano i zweryfikowano działanie takiej aplikacji w rzeczywistych warunkach.

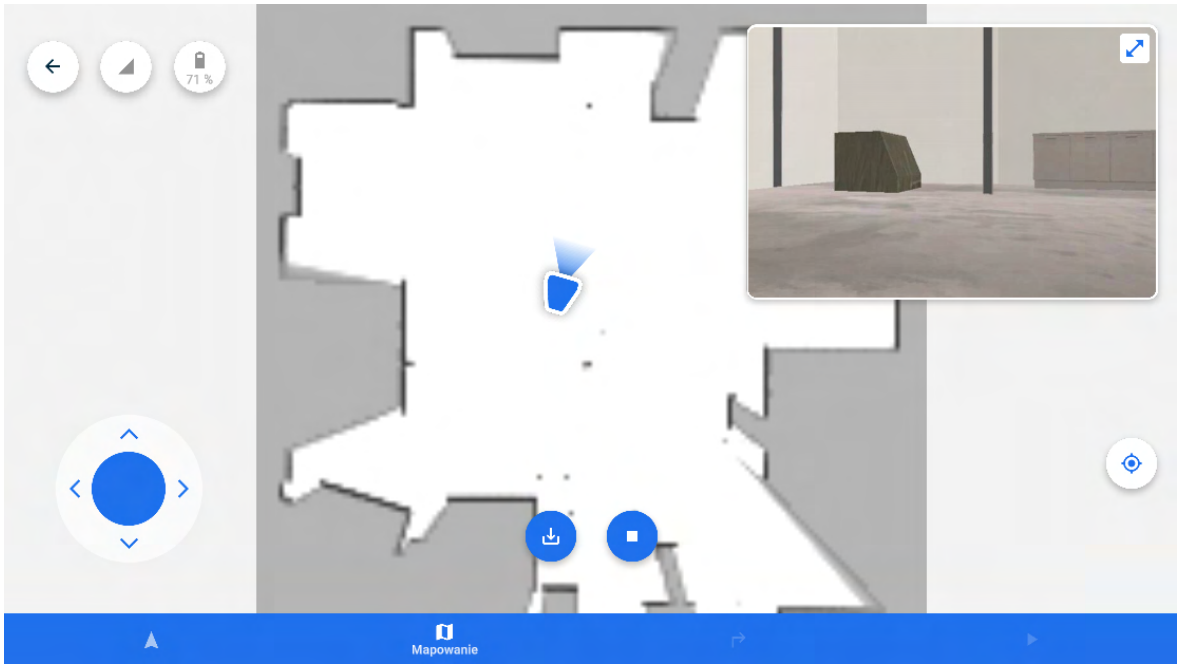
Zastosowanie elementów integracji symulacji z rzeczywistym sprzętem pozwo-

liło na przeprowadzanie tzw. testów HIL (z ang. *Hardware-in-the-Loop*), których architekturę przedstawiono na rys. 5.18. Zastosowanie oprogramowania autonomii na jednostce sterującej robota i wykorzystanie syntetycznych danych z symulatora w sposób niemal identyczny jak w przypadku rzeczywistych danych, było możliwe dzięki modułowości przygotowywanego rozwiązania polegającej m.in. na stosowaniu odpowiednich formatów danych czy protokołów komunikacyjnych oraz wykorzystaniu modułu UDP Bridge opisanego w sekcji 4.5.

Pierwszym z elementów aplikacji zarządzającej pojazdem autonomicznym opra-



Rysunek 5.18: Schemat integracji symulatora z systemem autonomii na potrzeby testów aplikacji operatora.



Rysunek 5.19: Tryb mapowania pomieszczenia i podgląd mapy oraz obrazu z kamery w aplikacji zarządzającej autonomią z wykorzystaniem środowiska symulacyjnego.

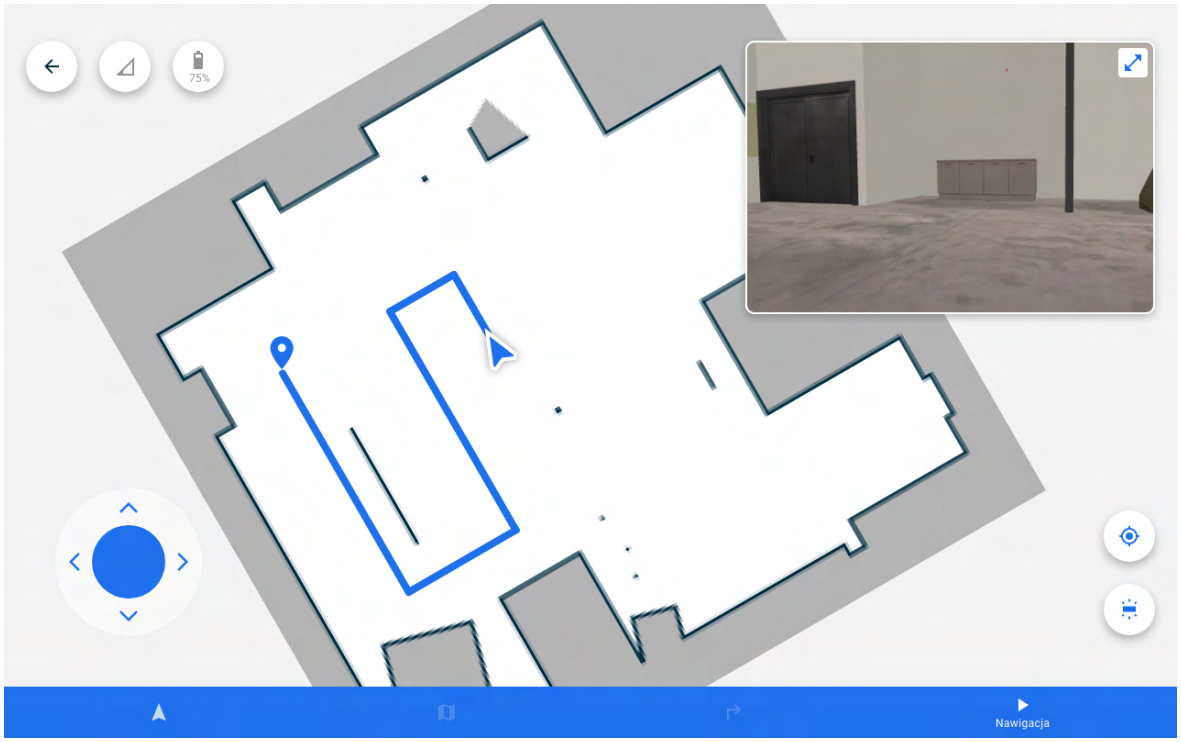
cowanych z wykorzystaniem symulatora była wizualizacja otoczenia na podstawie danych z kamery oraz mapy generowanej przez algorytm SLAM. Wirtualne sensory dzięki integracji z ROS dostarczają danych w formacie zgodnym z całym systemem autonomii. Dzięki temu generowanie mapy wykonywane przez algorytm SLAM odbywało się w sposób identyczny jak w przypadku rzeczywistego pojazdu, a sama wizualizacja dla operatora polegała na odpowiednim przedstawieniu danych zawartych w wiadomości typu *Occupancy Grid*. Jednak w przypadku kamery oraz zachowania kompatybilności ze sprzętem niezbędne było przygotowanie dodatkowego elementu. Rzeczywisty pojazd wyposażony jest w kamerę IP do obserwacji otoczenia strumieniującą dane za pomocą standardu RTSP (z ang. *Real Time Streaming Protocol*), niezależnie od systemu ROS. Aby uzyskać jak najmniejsze opóźnienia obrazu w aplikacji zarządzającej systemem, zdecydowano się na bezpośrednie dekodowanie strumienia wideo. Aby w aplikacji wyświetlić podgląd z wirtualnej kamery, niezbędne było enkodowanie sygnału do standardu H.264 oraz jego transmisja protokołem RTSP. Dzięki tym operacjom operator pojazdu autonomicznego może zorientować się, za pomocą aplikacji, jak wygląda otoczenie wirtualnego pojazdu w symulacji, w dokładnie taki sam sposób, jak w przypadku rzeczywistego robota, co zostało przedstawione na rys. 5.19.

Kolejnym aspektem, w którym istotną rolę spełnia aplikacja zarządzająca systemem, jest sterowanie i nadzorowanie przejazdu autonomicznego. Jak wspomniano we wcześniejszych sekcjach pracy, symulator dostarcza danych dla algorytmów lokalizacji i nawigacji umożliwiając tym samym ich działanie w sposób identyczny, jak z wykorzystaniem rzeczywistych urządzeń. Wykorzystując dane syntetyczne, opracowano funkcjonalność wskazywania punktu docelowego na mapie, wyznaczania przybliżonej drogi do celu oraz wskazywania pozycji pojazdu względem mapy na pokonywanej trasie. Na rys. 5.20 przedstawiono stan aplikacji, w którym możliwe jest obserwowanie postępów robota w autonomicznej jeździe do celu z wykorzystaniem systemu symulacji. Operator w każdej chwili może zatrzymać i przejąć sterowanie nad robotem w przypadku zauważenia anomalii w ruchu pojazdu, zapewniając dodatkowy stopień kontroli bezpieczeństwa.

Zastosowanie jednostki centralnej będącej rzeczywistym, osobnym komputerem

pozwoliło traktować system jako fizyczne urządzenie, co z kolei umożliwiło testy funkcjonalności niezwiązanych bezpośrednio z autonomią, takich jak wstępna konfiguracja systemu, zarządzanie plikami (np. zapisanymi mapami) czy zabezpieczenie oprogramowania przed nieautoryzowanym dostępem.

Dzięki integracji symulatora z aplikacją zarządzającą autonomią możliwe jest również wykorzystanie opracowanego środowiska w celu szkolenia operatorów pojazdów autonomicznych. Przygotowanie dodatkowych map 3D imitujących rzeczywiste obszary wykorzystania systemu w praktyce pozwoli w znacznym stopniu ograniczyć czas potrzebny do zapoznania się z rzeczywistym pojazdem wyposażonym w autonomię, zmniejszając tym samym koszt i zwiększając efektywność szkoleń.



Rysunek 5.20: Pozycja robota względem wyznaczonej trasy przejazdu oraz postęp nawigacji w aplikacji zarządzającej autonomią z wykorzystaniem środowiska symulacyjnego.

5.8 Podsumowanie

Przygotowany symulator pozwolił na przeprowadzenie analizy i ewaluacji wybranych zagadnień i algorytmów autonomii jazdy w celu oceny zasadności stosowania symulacji komputerowej w pracach badawczo-rozwojowych nad systemami autonomicznego sterowania.

Generowanie danych syntetycznych z wielu, różnego rodzaju sensorów umożliwiło bezpośrednią pracę z danymi w celu ich wstępnego przetwarzania. W przypadku danych lidarowych zaimplementowano algorytmy filtracji chmur punktów, które przetestowano obserwując w czasie rzeczywistym zachowanie poszczególnych parametrów. Dokładne odwzorowanie punktów montażowych na modelu pojazdu pozwoliło na dobranie i weryfikację konfiguracji odpowiedzialnej za utworzenie wspólnej chmury punktów z wielu lidarów. Ponadto korzystając z wirtualnych modeli kamer przetestowano procedurę kalibracji kamery, którą wykorzystano następnie w rzeczywistych urządzeniach.

Jednym z ważniejszych zastosowań symulatora było umożliwienie ewaluacji algorytmów SLAM. Wiarygodne generowanie chmur punktów pozwoliło na uzyskanie porównywalnych błędów lokalizacji ATE i RPE dla danych syntetycznych i rzeczywistych, potwierdzając dokładność generowanych danych lidarowych. Pozwoliło to również na przeprowadzenie wielu eksperymentów w celu dobrania odpowiedniej konfiguracji sprzętowej dla danego algorytmu SLAM w określonych warunkach środowiskowych. Zaproponowano również procedurę testową pozwalającą na znalezienie najlepszej konfiguracji czujników dla algorytmów lokalizacji i mapowania.

Wszechstronność symulatora potwierdzono również opracowując moduł pozwalający na tworzenie danych treningowych. W podstawowym zastosowaniu możliwy jest zapis obrazów z wirtualnej kamery wraz z odpowiadającą jej mapą semantyczną. Tak przygotowany zbiór danych może zostać wprost zastosowany na potrzeby segmentacji semantycznej obrazu z wykorzystaniem modeli głębokich sieci neuronowych, co zostało wykonane skutecznie trenując sieć BiSeNet, której wynik działania zastosowano jako dane wejściowe dla algorytmu utrzymywania toru jazdy.

W ramach analizy działania systemu przeprowadzono również badanie wydajności komunikacji symulatora z systemem autonomii. Wpływ ilości danych do wysłania na czas potrzebny do nadania tych danych okazał się kilkukrotnie mniejszy w przypadku autorskiego rozwiązania UDP Bridge w porównaniu do konkurencyjnego systemu wymiany informacji.

Zweryfikowano również możliwość zastosowania WST do opracowania i konfiguracji algorytmów nawigacji. Dzięki kompleksowej symulacji danych elementów pojazdu autonomicznego, możliwe było uruchomienie wszystkich najważniejszych algorytmów autonomii w głównym procesie przetwarzania. Pozwoliło to na weryfikację systemu nawigacji oraz dobór i dostosowanie odpowiednich parametrów bez dostępu do rzeczywistej platformy.

Opracowany symulator istotną rolę odegrał również przy tworzeniu aplikacji operatora. Zapewnienie strumieniowania wideo poprzez protokół RTSP pozwoliło zasymulować działanie rzeczywistej kamery, zastosowanej na pojeździe, dla którego aplikacja była przygotowywana. Dzięki działaniu symulatora w czasie rzeczywistym, testy aplikacji operatora odbywały się w głównej mierze poprzez połączenie z symulacją i sterowanie wirtualnym pojazdem. Umożliwiło to zwielokrotnienie stanowisk testowych bez stosowania wielu rzeczywistych platform autonomicznych.

Zagadnienia poruszone w tym rozdziale oraz wyniki przeprowadzonych prac zostały również opublikowane w następujących czasopismach:

- Sobczak Ł., Filus K., Domańska J., Domański A., Finding the best hardware configuration for 2D SLAM in indoor environments via simulation based on Google Cartographer, Scientific Reports, 12 (1), 2022.
- Sobczak Ł., Filus K., Domańska J., Domański A., Building a Real-Time Testing Platform for Unmanned Ground Vehicles with UDP Bridge, Sensors, 22 (21), 2022.
- Sobczak Ł., Filus K., Domański A., Domańska J., LiDAR Point Cloud Generation for SLAM Algorithm Evaluation, Sensors, 21 (10), 2021.

Rozdział 6

Wdrożenie systemu autonomii

Rezultatem prac podjętych w ramach doktoratu było opracowanie symulatora dla pojazdów autonomicznych typu BPL. Dzięki temu możliwe okazało się wdrożenie systemu autonomii w rzeczywistym pojeździe, który realizuje określone funkcje i podejmuje działania w sposób samodzielny, a którego elementy oprogramowania zostały opracowane, testowane i ewaluowane w środowisku wirtualnym. Rozdział ten poświęcono omówieniu zaproponowanego rozwiązania i przedstawieniu raportu z wdrożenia systemu autonomicznego sterowania wytworzonego w środowisku symulacyjnym.

6.1 Opis problemu

W zakres wdrożenia Wirtualnego Systemu Testującego, opracowanego w ramach niniejszej pracy, weszła implementacja systemu symulacji oraz integracja z wybranymi algorytmami jazdy autonomicznej. Głównym przyczynkiem dla opracowania symulatora dla pojazdów autonomicznych była potrzeba zweryfikowania działania wybranych algorytmów SLAM, wykorzystujących dane LiDAR, na małych pojazdach poruszających się w terenie lub w pomieszczeniach. Ze względu na brak odpowiedniego robota, który pozwalałby na testy lokalizacji zarówno w pomieszczeniach, jak i w terenie, jedynym zasadnym rozwiązaniem było opracowanie systemu symulacji. Jednocześnie, dysponując wieloma modelami sensorów różnego typu, możliwe

było ich zbadanie i wierne odwzorowanie w wirtualnym środowisku. Wraz z postępem prac, symulator był rozszerzany o kolejne elementy i funkcjonalności. Implementacja sterowania wirtualnym pojazdem z poziomu autonomii pozwoliła na aktywowanie modułu nawigacji, który obejmuje tworzenie mapy przeszkód, wyznaczanie optymalnej trasy przejazdu oraz generowanie komend sterowania w celu utrzymania wyznaczonej trajektorii. Pozytywne zweryfikowanie rozwiązania oraz dalsze problemy z dostępnością fizycznej platformy sprawiły, że prace nad symulatorem były kontynuowane. W ten sposób opracowano, opisane wcześniej, elementy WST umożliwiające zbieranie danych treningowych dla sieci neuronowych oraz integrację z aplikacją operatora i możliwość przeprowadzania kompleksowych testów autonomii bez dostępu do fizycznego pojazdu. Tak przygotowany system wdrożono na rzeczywisty pojazd, uruchamiając wszelkie opracowane moduły oprogramowania bez znaczących modyfikacji, potwierdzając tym samym użyteczność systemu symulacji. Wśród uruchomionych funkcjonalności znalazły się opisywane wcześniej algorytmy: SLAM do lokalizacji i budowania mapy otoczenia, tworzenie mapy przeszkód na potrzeby nawigacji, wyznaczanie i autonomiczna jazda po ścieżce do wskazanego celu z omijaniem statycznych i dynamicznych przeszkód, a także podążanie za wskazanym obiektem, np. człowiekiem.

6.2 Konstrukcja platformy autonomicznej

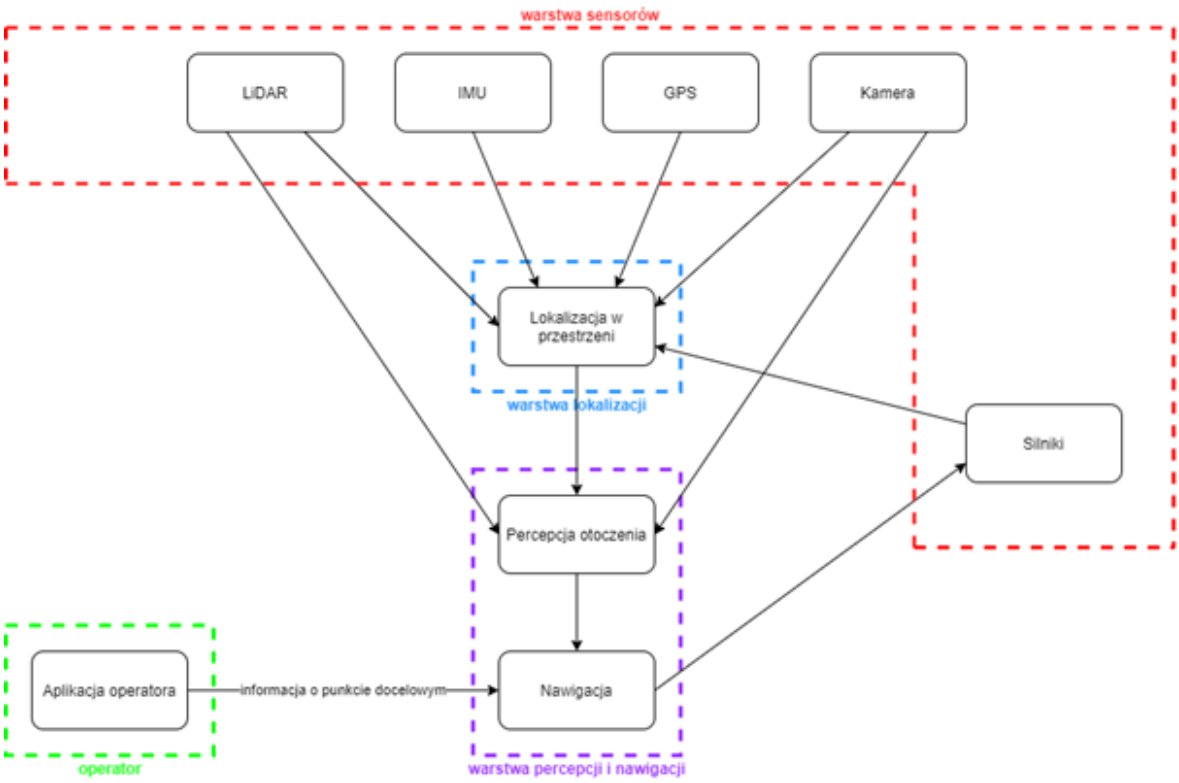
Zastosowana architektura sprzętowa i systemowa wynikała wprost z prac prowadzonych w symulatorze. Dysponując szczegółowym modelem 3D pojazdu, wykorzystano środowisko symulacji do optymalnego umiejscowienia poszczególnych sensorów. Dzięki temu możliwe było zweryfikowanie kilku możliwości i dobranie takiej konfiguracji, aby zastosowane czujniki pokrywały swoim zasięgiem cały obszar wokół pojazdu, uzyskując przy tym najlepsze rezultaty działania poszczególnych algorytmów autonomii. Ostatecznie wytypowano następujące sensory:

- 3 czujniki LiDAR 3D - zastosowano jeden główny skaner laserowy umieszczony w centralnej części pojazdu na podwyższeniu do dookólnej obserwacji

oraz dwa pomocnicze po bokach w celu wyeliminowania martwych stref głównego urządzenia oraz zwiększenia dokładności pomiarów w najbliższym otoczeniu pojazdu. Zbierane na bieżąco, z wysoką częstotliwością, chmury punktów znalazły zastosowanie zarówno przy lokalizacji pojazdu z wykorzystaniem SLAM, jak i przy budowaniu mapy przeszkód będącej kluczowym elementem wyznaczania bezpiecznej i optymalnej ścieżki dojazdu do wskazanego punktu w przestrzeni.

- kamera 3D - użycie kamery 3D na przedzie pojazdu pozwala na detekcję i klasyfikację obiektów na obrazie, a dodatkowa informacja o głębi zapewnia precyzyjne pozycjonowanie i śledzenie wykrytych obiektów, co znalazło zastosowanie w algorytmie podążania za wskazanym obiektem.
- IMU - dzięki zastosowaniu precyzyjnego układu typu MEMS składającego się z akcelerometru, żyroskopu oraz magnetometru zwiększono dokładność SLAM, którego implementacja wykorzystuje dane o prędkościach i przemieszczeniu pojazdu w przestrzeni. Dodatkowo dane te mogą być wykorzystane przez filtr EKF (z ang. *Extended Kalman Filter*), który potrafi wygładzić i ustabilizować wyznaczoną lokalizację, znacząco zwiększając przy tym precyzję wykonywanych ruchów.
- GNSS - system nawigacji satelitarnej uzupełnia dane lokalizacji o istotną informację o globalnym położeniu, co wykorzystywane jest m.in. do operowania autonomicznego w ramach współrzędnych geograficznych. Ułatwia to zarządzanie systemem w terenie otwartym np. poprzez wskazanie punktu docelowego w globalnym układzie współrzędnych wyrażonych za pomocą długości i szerokości geograficznej.

Na rys. 6.1 przedstawiono uproszczony diagram komponentów oprogramowania uwzględniający poszczególne algorytmy autonomii oraz elementy zarządzające systemem. Oprócz algorytmów omówionych we wcześniejszych rozdziałach niniejszej pracy, warto w tym miejscu zwrócić uwagę na maszynę stanów zarządzającą całym systemem. Utworzone stany działania odpowiadają poszczególnym funkcjonalnościom, które mogą być uruchamiane za pomocą aplikacji operatora w odpowiedniej



Rysunek 6.1: Uproszczony schemat komponentów oprogramowania wdrożonego systemu MSA.

kolejności. Przykładowo, gdy pojazd ma operować po predefiniowanej trasie przejazdu, niezbędne jest załadowanie wcześniej utworzonej mapy otoczenia lub uruchomienie algorytmu do mapowania przestrzeni. Podobnie gdy robot ma przemieszczać się za wskazanym obiektem niezbędne jest jego wybranie spośród obiektów wykrytych za pomocą metod sztucznej inteligencji. Maszyna stanów zarządzająca systemem pilnuje bezpiecznego wykonywania kolejnych operacji autonomicznych oraz, dodatkowo, zapewnia prawidłowe działanie pojazdu również w przypadku utraty połączenia z operatorem czy też awarii aplikacji zarządzającej. W krytycznej sytuacji następuje przełączenie do odpowiedniego stanu, tak aby wykonywane zadanie zostało w bezpieczny sposób przerwane lub, gdy zajdzie taka potrzeba, dokończone awaryjnie z zachowaniem przewidzianej procedury.

6.3 Uruchomienie i testy

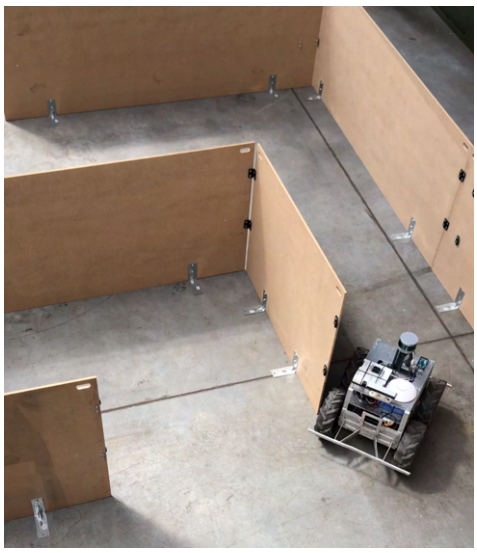
Pierwsze uruchomienie systemu autonomicznego poza środowiskiem symulacyjnym polega zazwyczaj na zweryfikowaniu poprawności działania poszczególnych komponentów oprogramowania. Dzięki wizualizacji danych z zamontowanych na pojeździe czujników oraz pomiarowi częstotliwości publikowania wiadomości w systemie potwierdzono sprawność warstwy sensorycznej. Dzięki temu oraz kompatybilności na poziomie danych, uruchomienie algorytmów autonomii, które były testowane z użyciem symulatora, przebiegło bez znaczących problemów.

W pierwszej kolejności uruchomiono algorytm SLAM. Zastosowanie symulatora do dobrania odpowiednich parametrów konfiguracji zastosowanej implementacji o nazwie Cartographer pozwoliło na szybkie uruchomienie systemu lokalizacji i mapowania. Jedyna niezbędna zmiana dotyczyła parametru, który określa liczbę skanów lidaru przypadającą na pojedynczy pełen skan otoczenia. Wynikało to z różnic pomiędzy rzeczywistym urządzeniem, a symulatorem. Ze względu na wydajność komputera symulacji wprowadzone zostało ograniczenie częstotliwości z jaką dokonywana jest aktualizacja silnika fizyki i pomiar odległości w danych punktach oraz przesyłanie danych do algorytmu. W tym przypadku jest to 24 lub 36 razy na jeden skan lidarowy przy wykorzystaniu wydajnego środowiska sprzętowego pozwalającego osiągnąć generowanie nawet 360 klatek na sekundę. Rzeczywisty sensor dokonuje pomiaru znacznie częściej, wykonując każdy pomiar w osobnym oknie czasowym nieprzekraczającym kilku mikrosekund, przy czym samo wysłanie danych odbywa się około 150 razy na jeden obrót lidaru. Takie wartości wymagałyby od komputera symulacji osiągnięcia odświeżania na poziomie niemal 1500 klatek na sekundę co jest wartością nieosiągalną dla obecnej generacji GPU. Jednak w praktyce, szczególnie dla pojazdów poruszających się z małą prędkością, różnice są niewielkie, co sprawia, że samo dostosowanie wspomnianego parametru nie powoduje pogorszenia estymacji położenia robota w przestrzeni czy utraty jakości budowanej mapy. Zostało to wykazane w rozdziale 5 i sekcji poświęconej ewaluacji algorytmów SLAM 5.2.

W przypadku modułu nawigacji do wskazanego punktu nie były wymagane

żadne zmiany, zarówno w samych plikach źródłowych oprogramowania, jak i w parametrach. Budowanie mapy przeszkód opierając się na danych lidarowych oraz wyznaczanie ścieżki do wskazanego punktu na mapie działało w sposób zgodny z zachowaniem zaobserwowanym przy wykorzystaniu symulatora. Niewielkie rozbieżności można było zaobserwować w kontekście tzw. lokalnego plannera, który jest odpowiedzialny za dostarczanie komend sterowania w celu podążania wyznaczoną trasą. Ze względu na brak dokładnej specyfikacji napędu platformy robotycznej, na której wdrażano system, poczyniono w symulatorze pewne założenia dotyczące zakresu prędkości oraz dostępnych wartości przyspieszeń, co wymagało drobnych korekt w celu upłynnienia ruchu pojazdu i spokojniejszego pokonywania zakrętów. Jednak nie wpływało to na samą funkcjonalność dojazdu do wskazanego punktu i w pełni pozwalało pokonać wyznaczoną ścieżkę autonomicznie omijając pojawiające się przeszkody.

Analogicznie w przypadku oprogramowania sterującego realizującego podążanie za wybranym obiektem nie były wymagane żadne zmiany, ani dostrajanie parametrów. Wstępnie skonfigurowany kontroler PID zapewnił odpowiednią skuteczność w utrzymywaniu dystansu do danego obiektu i kierunku w jakim należało podążać. W kontekście wykrywania obiektów oraz ich klasyfikacji nie przeprowadzano testów w środowisku symulacyjnym, podobnie jak w przypadku oprogramowania



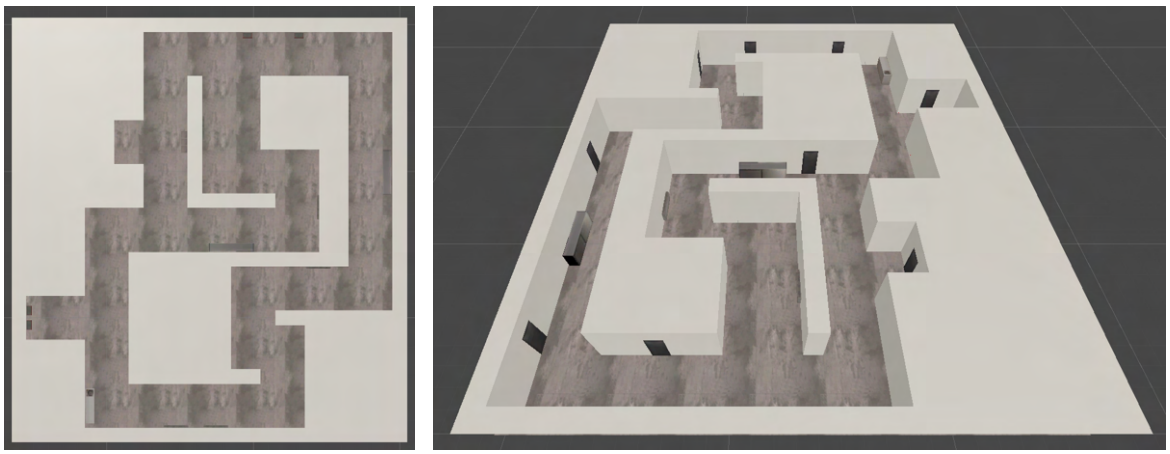
Rysunek 6.2: Mała platforma autonomiczna podczas pokonywania toru testowego.

do śledzenia obiektów w przestrzeni z wykorzystaniem kamery głębi. Komponenty te zostały opracowane osobno, poza opracowanym symulatorem.

W celu oceny i weryfikacji działania zaimplementowanego systemu zbudowano tor testowy pozwalający na przejazd małej platformie autonomicznej. Powstała ona na podstawie zamodelowanego obiektu przestrzennego w symulacji WST. Na rys. 6.2 przedstawiono poruszającego się robota, który za pomocą lidarów mapuje tor i określa swoją pozycję, co pozwala mu na dotarcie do określonego punktu na końcu labiryntu. Sama nawigacja do punktu oparta jest na przygotowanych mechanizmach tworzenia mapy przeszkód oraz wyznaczania trasy do celu bazując na algorytmie A* lub Dijkstra. Tor ten wykorzystano także podczas ewaluacji algorytmów SLAM, którą opisano w sekcji 5.2.

Rysunek 6.3: Robot do dekontaminacji w symulatorze WST.

W ramach wdrożenia wyników niniejszej pracy zrealizowano również projekt robota autonomicznego do dekontaminacji pomieszczeń zamkniętych. Jego celem było oczyszczanie w bezpieczny sposób miejsc potencjalnie zakażonych za pomocą zamontowanych na pojeździe świetlówek UV dużej mocy, które oczyszczają wszelkie powierzchnie wystawione na działanie promieni ultrafioletowych w określonym zasięgu działania zależnym od zastosowanych lamp. Dzięki zastosowaniu lidarów 2D oraz czujnika IMU realizowane jest mapowanie i lokalizacja w przestrzeni, co po-



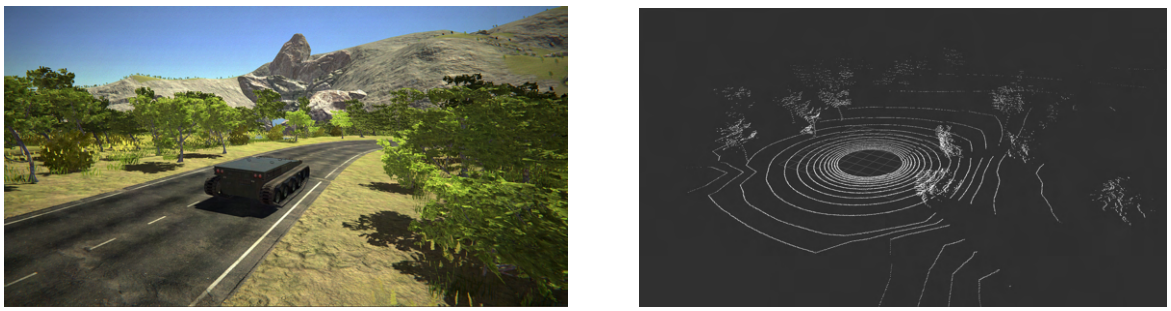
(a) (b)

Rysunek 6.4: Symulacyjny model 3D korytarza wykorzystywanego do testów algorytmów SLAM.

zwala na uruchomienie algorytmów nawigacji budujących mapę zajętości otoczenia oraz umożliwia wyznaczenie trasy odkażania danego pomieszczenia. Dzięki temu proces ten odbywa się w sposób zautomatyzowany bez udziału człowieka zapewniając tym samym bezpieczeństwo personelowi. Realizacja tego projektu również została wykonana za pomocą stworzonego symulatora WST oraz modelu robota i przykładowego pomieszczenia przedstawionego na rys. 6.3. Zastosowanie symulacji pozwoliło na wczesnym etapie projektu wytypować odpowiednią konfigurację czujników w taki sposób, aby algorytm SLAM działał z najlepszą skutecznością zachowując przy tym poziom kosztów zakupu sensorów na rozsądnym poziomie. Na podstawie rezultatów tego projektu opisano w szczegółowy sposób proces ewaluacji, zarówno dokładności tworzonej mapy otoczenia, jak i precyzji lokalizacji w pomieszczeniach. Takie podejście wymagało sprawdzenia działania algorytmu w różnych warunkach, dlatego też oprócz wirtualnej mapy pomieszczenia laboratoryjnego w WST, stworzono dodatkowy model 3D symulując korytarz, w którym robot mógłby potencjalnie się poruszać. Rys. 6.4 przedstawia opisany korytarz w widoku 3D oraz z tzw. lotu ptaka. Widać na nim długie, proste fragmenty, które pozwoliły na weryfikację działania lokalizacji w niesprzyjających warunkach, w których istotną rolę odgrywa zasięg lidara 2D.

Wyniki prac zastosowano również do uruchomienia pojazdu gąsienicowego, któ-

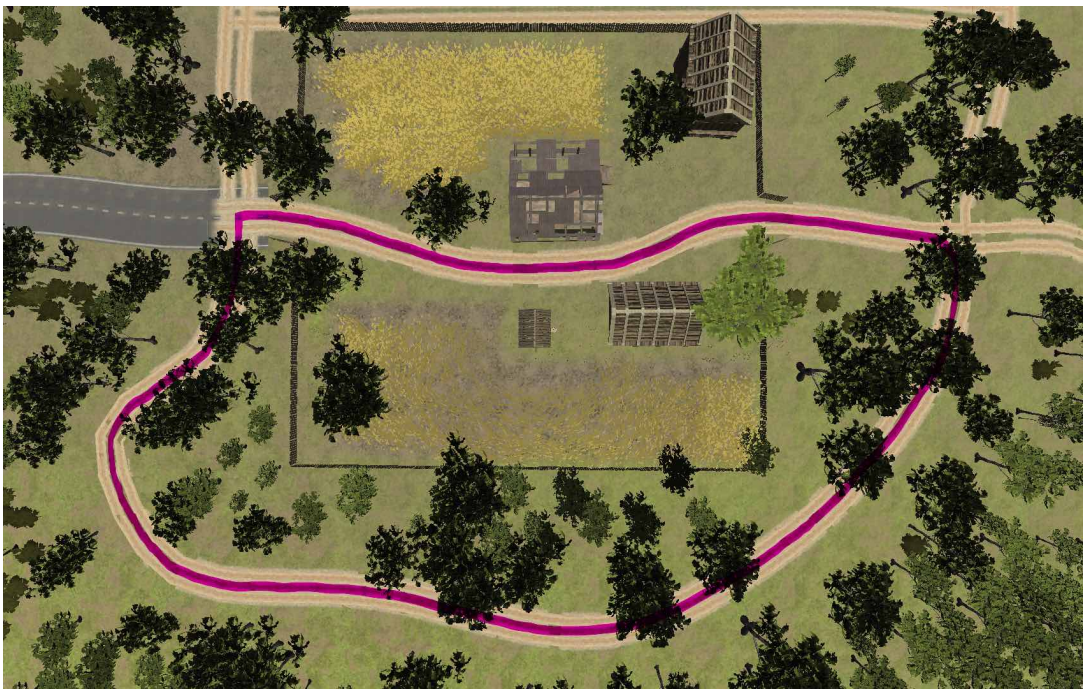
rego potencjalnym zastosowaniem miało być autonomiczne pokonywanie tras terenowych z punktu A do punktu B. Wszelkie prace, ze względu na niedostępność platformy testowej, zrealizowano opierając się na symulacjach w środowisku WST. Bazując na poprzednich doświadczeniach wytypowano rodzaje sensorów wymaganych dla platformy, ich pożądane parametry operacyjne oraz punkty montażowe na pojeździe tak aby uzyskać jak najlepsze rezultaty. Ze względu na wymiary platformy było kluczowe, aby zapewnić widoczność najważniejszych sensorów przy minimalizacji tzw. martwych stref, w których czujniki nie umożliwiałyby detekcji przeszkód tym samym prowadząc potencjalnie do niebezpiecznych sytuacji podczas jazdy autonomicznej. Na rys. 6.5a przedstawiono terenową mapę testową z odcinkiem drogi asfaltowej, która posłużyła do weryfikacji wielu algorytmów opisywanych w niniejszej pracy, przede wszystkim algorytmów SLAM. Rys. 6.5b przedstawia skan lidarowy uzyskany dzięki wirtualnemi sensorowi w WST, będącemu źródłem danych dla lokalizacji i mapowania w przestrzeni. Zastosowanie wirtualnej kamery symulującej tę, w którą potencjalnie można wyposażyć pojazd, pozwoliło na zweryfikowanie działania SLAM również na danych wizyjnych. Pozwoliło to również na weryfikację i ocenę, który ze sposobów pozycjonowania w przestrzeni jest bardziej odpowiedni, oraz porównanie generowanej trajektorii. Uzyskane wyniki przedstawiono na rys. 6.6. Zastosowanie mapy terenowej zawierającej wirtualne modele drzew i krzewów oraz przedstawiającej tereny wiejskie odpowiadało teoretycznym warunkom operacyjnym platformy gąsienicowej. Dodatkowe testy algorytmów nawigacji oraz stero-



(a) (b)

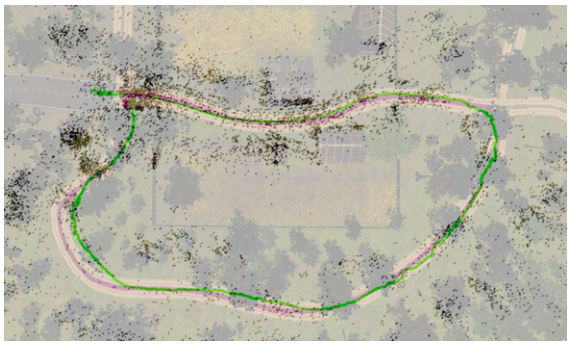
Rysunek 6.5: Autonomiczny pojazd gąsienicowy w środowisku WST (a) oraz chmura punktów z wirtualnego lidaru 3D (b).

wania, a także komunikacji z operatorem przeprowadzone w WST, pozwoliły na ich wdrożenie na docelową platformę gotową do przeprowadzenia testów w rzeczywistym terenie.



(a) Rzeczywista trasa pojazdu.

(b) Trajektoria wygenerowana przez SLAM Cartographer.



(c) Trajektoria wygenerowana przez ORB-SLAM2.

Rysunek 6.6: Trajektorie uzyskane w WST.

Rozdział 7

Podsumowanie pracy

Celem niniejszej pracy było stworzenie mechanizmów symulacyjnych, które pozwoliłyby na ewaluację wybranych algorytmów autonomii jazdy. W pracy przedstawiono szereg możliwych zastosowań symulacji komputerowej na potrzeby opracowywania systemów autonomii jazdy bezzałogowych platform lądowych. Wśród najistotniejszych zagadnień należy wymienić ewaluację algorytmów SLAM oraz dobór odpowiednich sensorów w zależności od wymagań stawianych systemowi autonomicznemu. Opracowany w ramach pracy autorski symulator umożliwia również generowanie syntetycznych danych sensorycznych oraz sterowanie dowolnymi pojazdami typu BPL w czasie rzeczywistym. Przeprowadzone badania wykazały zasadność stosowania symulacji na różnych etapach prac badawczo-rozwojowych nad systemami autonomii. Dodatkowo wykazano, że odwzorowanie cech charakterystycznych dla danych sensorów pozwala zauważalnie zwiększyć podobieństwo zachowań konkretnych algorytmów autonomii operujących na syntetycznych danych, do tych, które wykorzystują dane z rzeczywistych sensorów. Przekłada się to na czas potrzebny na testy rzeczywistego pojazdu oraz łatwość wdrożenia systemu w finalnym produkcie. Ponadto, zrealizowany w ramach pracy symulator pozwala na kompleksowe wykorzystanie na różnych etapach opracowywania systemu autonomii jazdy począwszy od fazy koncepcyjnej, poprzez badania i rozwój, na wdrożeniu kończąc, ograniczając tym samym do minimum potrzebę korzystania z rzeczywistego pojazdu.

W ramach pracy opracowano modele kilku typów sensorów przeznaczonych dla systemów autonomii takich jak LiDAR, kamera czy IMU. W literaturze można znaleźć wiele przykładów zastosowania symulatorów do testów algorytmów sterowania czy wnioskowania, a także generowania danych treningowych dla modeli sztucznej inteligencji. Możliwe jest jednak operowanie na niższym poziomie abstrakcji, niejako uzupełniając zakres wykorzystania symulacji również o początkowe fazy prac badawczo-rozwojowych. Algorytmy lokalizacji i mapowania SLAM, systemy zarządzające wykonywaniem misji autonomicznych czy też wstępne przetwarzanie danych sensorycznych to niektóre z potencjalnych zastosowań symulacji komputerowej i obszarów, które mogą być rozwijane przy ograniczonym dostępie do rzeczywistego pojazdu podczas prac nad autonomią. Uzyskane w pracy wyniki wskazują, że dzięki odpowiedniej symulacji poszczególnych czujników, dane algorytmy opracowane na potrzeby autonomii i wykorzystujące syntetyczne dane zachowują się w sposób zbliżony do tych operujących na rzeczywistych danych.

Zagadnienie zastosowania symulacji komputerowej na potrzeby opracowywania systemów autonomii, nawet w przypadku ograniczenia do bezzałogowych pojazdów lądowych to szerokie zagadnienie wymagające działań w wielu obszarach. W ramach pracy poruszono kilka kluczowych aspektów przygotowując szereg zaprezentowanych rozwiązań. Na przykładzie LiDAR-u zaprezentowano, że dokładne odwzorowanie rzeczywistego działania sensora, formatu dostarczanych danych czy innych parametrów zapewnia symulację danych na poziomie pozwalającym uzyskać wyniki działania algorytmów zbliżone do ich odpowiedników operujących na danych rzeczywistych.

Dalszy rozwój pomysłów przedstawionych w ramach niniejszej pracy powinien dotyczyć szybkości wykonywania niektórych obliczeń. Dokonane do tej pory działania optymalizacyjne koncentrowały się w głównej mierze na przyspieszeniu symulacji danych sensorycznych. Jednak wciąż pozostają otwarte zagadnienia z zakresu grafiki komputerowej i wydajnego renderowania złożonych obiektów 3D czy też symulacji fizyki, które powinny cechować się wyższą wydajnością obliczeń bez wpływu na jakość generowanych danych.

Bibliografia

[1] Szczęch L. Baszuk K. Zeszyty naukowe akademii marynarki wojennej. *Zeszyty Naukowe Akademii Marynarki Wojennej*, 51(4):7–12, 2010.

[2] Evan Ackerman. Lockheed's Robotic Trucks Pass Real-World Military Convoy Test. `https://spectrum.ieee.org/lockheeds-robotic-trucks-pass-real-world-military-convoy-test`. [Online; accessed 10-October-2022].

[3] Josh Luckenbaugh. Rheinmetall Unveils New Autonomous, Amphibious Vehicle. `https://www.nationaldefensemagazine.org/articles/2022/10/10/rheinmetall-unveils-autonomous-amphibious-vehicle`. [Online; accessed 10-October-2022].

[4] Mayank Bansal, Alex Krizhevsky, and Abhijit S. Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. *CoRR*, abs/1812.03079, 2018.

[5] Sivabalan Manivasagam, Shenlong Wang, Kelvin Wong, Wenyuan Zeng, Mikita Sazanovich, Shuhan Tan, Bin Yang, Wei-Chiu Ma, and Raquel Urtasun. Lidarsim: Realistic lidar simulation by leveraging the real world. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11167–11176, 2020.

[6] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on Robot Learning*, pages 1–16, 2017.

[7] A. Lafrance. Your Grandmother's Driverless Car. `https://www.theatlantic.com/technology/archive/2016/06/beep-beep/489029/`, 2016. [Online; accessed 10-September-2021].

[8] L. Earnest. Stanford Cart. `https://web.stanford.edu/~learnest/sail/oldcart.html`, 2012. [Online; accessed 10-September-2021].

[9] R. D. Leighty. Darpa alv summary. *Defense Technical Information Center*, 1986.

[10] EUREKA. Programme for a european traffic system with highest efficiency and unprecedented safety. `http://www.eurekanetwork.org/project/id/45`, 2016. [Online; accessed 19-May-2020].

[11] M. Novak. DARPA Tried to Build Skynet in the 1980s. `https://gizmodo.com/darpa-tried-to-build-skynet-in-the-1980s-1451000652`, 2013. [Online; accessed 10-September-2021].

[12] D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. *Neural Information Processing Systems*, 1988.

[13] T. R. Gullstrand. Prometheus. *International Journal of Vehicle Design*, 4(10):428–431, 1989.

[14] J. Delcker. The man who invented the self-driving car (in 1986). `https://www.politico.eu/article/delf-driving-car-born-1986-ernst-dickmanns-mercedes/`, 2018. [Online; accessed 11-September-2021].

[15] E. Dickmanns. Vehicles capable of dynamic vision. *International joint conference on Artifical intelligence*, pages 1577–1592, 1997.

[16] H. Kjellander. Arbiter and simulation for team caltech in darpa grand challenge. 2004.

[17] S. Thrun et al. Stanley: The robot that won the darpa grand challenge. *Journal of Field Robotics*, 9(23):661–692, 2006.

[18] Brian P. Gerkey, Richard T. Vaughan, and Andrew Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. pages 317–323, 2003.

[19] M. Montemerlo et al. Junior: The stanford entry in the urban challenge. *Journal of Field Robotics*, 9(25):569–597, 2008.

[20] U. Ozguner et al. Simulation and testing environments for the darpa urban challenge. pages 222–226, 2008.

[21] Peter Stone, Patrick Beeson, Tekin Mericli, and Ryan Madigan. Darpa urban challenge technical report: Austin robot technology, June 2007. Available from http://www.darpa.mil/grandchallenge/rules.asp.

[22] B. Upcroft et al. Darpa urban challenge technical paper: Sydney-berkeley driving team. Technical report, University of Sydney; University of Technology, Sydney; University of California, Berkeley, 2007.

[23] M. Harris. How Google's Autonomous Car Passed the First U.S. State Self-Driving Test. `https://spectrum.ieee.org/how-googles-autonomous-car-passed-the-first-us-state-selfdriving-test`, 2014. [Online; accessed 21-December-2021].

[24] Waymo. Waymo Safety Report. `https://waymo.com/safety/`, 2021. [Online; accessed 21-December-2021].

[25] The Waymo Team. Simulation City: Introducing Waymo's most advanced simulation system yet for autonomous driving. `https://blog.waymo.com/2021/06/SimulationCity.html`, 2021. [Online; accessed 28-December-2021].

[26] Z. Yang et al. SurfelGAN: Synthesizing Realistic Sensor Data for Autonomous Driving. *Computer Vision and Pattern Recognition*, 2020.

[27] John M. Scanlon, Kristofer D. Kusano, Tom Daniel, Christopher Alderson, Alexander Ogle, and Trent Victor. Waymo simulated driving behavior in reconstructed fatal crashes within an autonomous vehicle operating domain. *Accident Analysis & Prevention*, 163, 2021.

[28] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics*, pages 621–635. Springer, 2018.

[29] Mathworks. Automated Driving Toolbox. `https://www.mathworks.com/products/automated-driving.html`, 2021. [Online; accessed 28-December-2021].

[30] Ansys. Autonomous System Validation. `https://www.ansys.com/applications/autonomous-vehicle-validation`, 2021. [Online; accessed 28-December-2021].

[31] SAE International. Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles. `https://www.sae.org/standards/content/j3016_202104/`, 2021. [Online; accessed 19-August-2021].

[32] Jenay M. Beer, Arthur D. Fisk, and Wendy A. Rogers. Toward a framework for levels of robot autonomy in human-robot interaction. *Journal of Human-Robot Interaction*, 3(2):74–99, July 2014.

[33] Oshkosh Defense. Team TerraMax Completes Historic DARPA Grand Challenge. `https://oshkoshdefense.com/team-terramax-completes-historic-darpa-grand-challenge-2/`, 2005. [Online; accessed 21-December-2021].

[34] Max Bajracharya, Mark W. Maimone, and Daniel Helmick. Autonomy for mars rovers: Past, present, and future. *Computer*, 41(12):44–50, 2008.

[35] NASA Jet Propulsion Laboratory. NASA's Self-Driving Perseverance Mars Rover 'Takes the Wheel'. `https://www.jpl.nasa.gov/news/nasas-self-driving-perseverance-mars-rover-takes-the-wheel`, 2021. [Online; accessed 21-December-2021].

[36] Vandi Verma and Chris Leger. SSim: NASA Mars Rover Robotics Flight Software Simulation. pages 1–11, 2019.

[37] NASA Jet Propulsion Laboratory. Computer Simulation of Perseverance's First Autonav Drive. `https://mars.nasa.gov/resources/26022/computer-simulation-of-perseverances-first-autonav-drive/`, 2021. [Online; accessed 21-December-2021].

[38] Y. Edan. Design of an autonomous agricultural robot. *Applied Intelligence*, 5(1):41–50, 1995.

[39] T. Torii. Research in autonomous agriculture vehicles in japan. *Computers and Electronics in Agriculture*, 25:133–153, 2000.

[40] K. H. Choi, S. K. Han, S. H. Han, K.-H. Park, K.-S. Kim, and S. Kim. Morphology-based guidance line extraction for an autonomous weeding robot in paddy fields. *Computers and Electronics in Agriculture*, 113:266–274, 2015.

[41] A. Kamilaris and F. X. Prenafeta-Boldú. Deep learning in agriculture: A survey. *Computers and Electronics in Agriculture*, 147:70–90, 2018.

[42] Dan Carney. Laser-Armed Robot Terminates Weeds Without Herbicide Carbon Robotics. `https://www.designnews.com/automation/laser-armed-robot-terminates-weeds-without-herbicide`, 2021. [Online; accessed 21-December-2022].

[43] Will Knight. John Deere's Self-Driving Tractor Stirs Debate on AI in Farming. `https://www.wired.com/story/john-deere-self-driving-tractor-stirs-debate-ai-farming/`, 2022. [Online; accessed 04-January-2022].

[44] Safaa Sindi and Roger Woodman. Autonomous goods vehicles for last-mile delivery: Evaluation of impact and barriers. In *23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6, 2020.

[45] Toivo Tänavsuu. Estonian delivery robots are transforming the world. `https://estonia.ee/delivery-robots-created-by-estonian-engineers-are-transforming-the-world/`, 2017. [Online; accessed 04-January-2022].

[46] Chris Albrecht. Delivers AI Robots Making Deliveries in Turkey. `https://thespoon.tech/delivers-ai-robots-making-deliveries-in-turkey/`, 2021. [Online; accessed 04-January-2022].

[47] Lex Fridman. MIT 6.S094: Deep Learning for Self-Driving Cars 2018, Lecture 2: Self-Driving Cars. `https://deeplearning.mit.edu`. [Online; accessed 22-January-2022].

[48] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[49] R. Mur-Artal and J. M. M. Montiel. Orb-slam: a versatile and accurate monocular slam system. In *IEEE Transactions On Robotics*, pages 1147–1163, 2016.

[50] C. Premebida, J. Carreira, J. Batista, and U. Nunes. Pedestrian detection combining rgb and dense lidar data. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4112–4117, 2014.

[51] S. Thrun et al. Towards fully autonomous driving: Systems and algorithms. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 163–168, 2011.

[52] J. Han, D. Kim, M. Lee, and M. Sunwoo. Enhanced road boundary and obstacle detection using a downward-looking lidar sensor. In *IEEE Transactions On Vehicular Technology*, pages 971–985, 2012.

[53] M. Himmelsbach, T. Luettel, and H.-J. Wuensche. Real-time object classification in 3d point clouds using point feature histograms. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 994–1000, 2009.

[54] F. Moosmann, O. Pink, and C. Stiller. Segmentation of 3d lidar data in non-flat urban environments using a local convexity criterion. In *IEEE Intelligent Vehicles Symposium*, pages 215–220, 2009.

[55] Christophe Blanc, Romuald Aufrère, Laurent Malaterre, Jean Gallice, and Joseph Alizon. Obstacle detection and tracking by millimeter wave radar. *IFAC Proceedings Volumes*, 37(8):322–327, 2004. IFAC/EURON Symposium on Intelligent Autonomous Vehicles, Lisbon, Portugal, 5-7 July 2004.

[56] Tao Wang, Nanning Zheng, Jingmin Xin, and Zheng Ma. Integrating millimeter wave radar with a monocular vision sensor for on-road obstacle detection applications. *Sensors*, 11(9):8992–9008, 2011.

[57] Ningbo Long, Kaiwei Wang, Ruiqi Cheng, Weijian Hu, and Kailun Yang. Unifying obstacle detection, recognition, and fusion based on millimeter wave radar and rgb-depth sensors for the visually impaired. *Review of Scientific Instruments*, 90(4):044102, 2019.

[58] Fahad Jibrin Abdu, Yixiong Zhang, Maozhong Fu, Yuhan Li, and Zhenmiao Deng. Application of deep learning on millimeter-wave radar signals: A review. *Sensors*, 21(6), 2021.

[59] Aisha Mohammed, Aliyu Abdullahi, and Amina Ibrahim. Development of a prototype autonomous electric vehicle. *Journal of Robotics and Control (JRC)*, 2, 01 2021.

[60] R.W. Wall, J. Bennett, and G. Eis. Creating a low-cost autonomous vehicle. In *IEEE 2002 28th Annual Conference of the Industrial Electronics Society. IECON 02*, volume 4, pages 3112–3116 vol.4, 2002.

[61] Marco Claudio De Simone, Zandra Betzabe Rivera, and Domenico Guida. Obstacle avoidance system for unmanned ground vehicles by using ultrasonic sensors. *Machines*, 6(2), 2018.

[62] Sungyoung Jung, Jungmin Kim, and Sungshin Kim. Simultaneous localization and mapping of a wheel-based autonomous vehicle with ultrasonic sensors. *Artificial Life and Robotics*, 14(2):186, Dec 2009.

[63] Shaojiang Zhang, Yanning Guo, Qiang Zhu, and Zhiyuan Liu. Lidar-imu and wheel odometer based autonomous vehicle localization system. In *2019 Chinese Control And Decision Conference (CCDC)*, pages 4950–4955, 2019.

[64] Veli Ilci and Charles Toth. High definition 3d map creation using gnss/imu/lidar sensor integration to support autonomous vehicle navigation. *Sensors*, 20(3), 2020.

[65] S. Sukkarieh, E.M. Nebot, and H.F. Durrant-Whyte. A high integrity imu/gps navigation loop for autonomous land vehicle applications. *IEEE Transactions on Robotics and Automation*, 15(3):572–578, 1999.

[66] Rajesh Saini, Manish Laxman Karle, Ujjwala Shailesh Karle, Fenin Karuvelil, and Mangesh Ramesh Saraf. Implementation of multi-sensor gps/imu integration using kalman filter for autonomous vehicle. In *Symposium on International Automotive Technology 2019*. SAE International, jan 2019.

[67] Justin Kruger, Arno Rogg, and Ramon Gonzalez. Estimating wheel slip of a planetary exploration rover via unsupervised machine learning. In *2019 IEEE Aerospace Conference*, pages 1–8, 2019.

[68] Xuesu Xiao, Yulin Zhang, Haifeng Li, Hongpeng Wang, and Binbin Li. Camera-imu extrinsic calibration quality monitoring for autonomous ground vehicles. *IEEE Robotics and Automation Letters*, 7(2):4614–4621, 2022.

[69] Francois Caron, Emmanuel Duflos, Denis Pomorski, and Philippe Vanheeghe. Gps/imu data fusion using multisensor kalman filtering: introduction of contextual aspects. *Information Fusion*, 7(2):221–230, 2006.

[70] Amol P. Dhongade and M.A. Khandekar. Gps and imu integration on an autonomous vehicle using kalman filter (labview tool). In *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*, pages 1122–1125, 2019.

[71] Yahui Liu, Xiaoqian Fan, Chen Lv, Jian Wu, Liang Li, and Dawei Ding. An innovative information fusion method with adaptive kalman filter for inte-

grated ins/gps navigation of autonomous vehicles. *Mechanical Systems and Signal Processing*, 100:605–616, 2018.

[72] Hao Jing, Yang Gao, Sepeedeh Shahbeigi, and Mehrdad Dianati. Integrity monitoring of gnss/ins based positioning systems for autonomous vehicles: State-of-the-art and open challenges. *IEEE Transactions on Intelligent Transportation Systems*, 23(9):14166–14187, 2022.

[73] Sagar Dasgupta, Tonmoy Ghosh, and Mizanur Rahman. A reinforcement learning approach for global navigation satellite system spoofing attack detection in autonomous vehicles. *Transportation Research Record: Journal of the Transportation Research Board*, page 036119812210955, jun 2022.

[74] Aleksander Nowak. Dynamic gnss mission planning using dtm for precise navigation of autonomous vehicles. *Journal of Navigation*, 70(3):483–504, 2017.

[75] Sagar Behere and Martin Torngren. A functional architecture for autonomous driving. In *First International Workshop on Automotive Software Architecture (WASA)*, pages 3–10, 2015.

[76] C. Urmson et al. Autonomous driving in urban environments - boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008.

[77] S.H. Jung and C.J. Taylor. Camera trajectory estimation using inertial sensor measurements and structure from motion results. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages II–732 – II–737, 2001.

[78] M.B. Alatise. Pose estimation of a mobile robot based on fusion of imu data and vision data using an extended kalman filter. *Sensors*, 17(10):2164, 2017.

[79] P. Wang, P. Chen, Y. Yuan, D. Liu, Z. Huang, X. Hou, and G. Cottrell. Understanding convolution for semantic segmentation. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1451–1460. IEEE Computer Society, 2018.

[80] Mennatullah Siam, Sara Elkerdawy, Martin Jagersand, and Senthil Yogamani. Deep semantic segmentation for automated driving: Taxonomy, roadmap and challenges, 2017.

[81] Michael Treml, J. Arjona-Medina, T. Unterthiner, R. Durgesh, F. Friedmann, P. Schuberth, A. Mayr, M. Heusel, M. Hofmarcher, M. Widrich, B. Nessler, and S. Hochreiter. Speeding up semantic segmentation for autonomous driving. In *Proceedings of the MLITS, NIPS Workshop*, 2016.

[82] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495, 2017.

[83] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):640–651, 2017.

[84] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. volume 1, pages 1097–1105, 2012.

[85] C. Szegedy et al. Going deeper with convolutions. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[86] Ping-Min Hsu, Ming-Hung Li, and Yi-Feng Su. Object detection and recognition by using sensor fusion. pages 56–60, 2014.

[87] Maxim Likhachev, David Ferguson, Geoffrey Gordon, Anthony Stentz, and Sebastian Thrun. Anytime dynamic a*: An anytime, replanning algorithm. pages 262–271, 2005.

[88] Keonyup Chu, Minchae Lee, and Myoungho Sunwoo. Local path planning for off-road autonomous driving with avoidance of static obstacles. *IEEE Transactions on Intelligent Transportation Systems*, 13(4):1599–1616, 2012.

[89] F. Hundelshausen et al. Driving with tentacles - integral structures for sensing and motion. *Journal of Field Robotics*, 25(9):569–597, 2008.

[90] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *The Computing Research Repository (CoRR)*, 2016.

[91] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2149–2154, 2004.

[92] Shyngyskhan Abilkassov. Autonomous pallet distribution in Amazon warehouse. `https://kurshakuz.github.io/projects/amazon-warehouse/`. [Online; accessed 13-September-2022].

[93] Gregory Junker. *Pro OGRE 3D programming*. Apress, 2007.

[94] Jeffrey Craighead, Jenny Burke, and Robin Murphy. Using the unity game engine to develop sarge: A case study. *Computer*, 2007.

[95] Miguel Figueiredo, Rosaldo Rossetti, Rodrigo Braga, and Luís Reis. An approach to simulate autonomous vehicles in urban traffic scenarios. In *Intelligent Transportation Systems (ITSC)*, pages 1–6, 2009.

[96] Francisca Rosique, Pedro J. Navarro, Carlos Fernández, and Antonio Padilla. A systematic review of perception system and simulators for autonomous vehicles research. *Sensors*, 19(3), 2019.

[97] Carla Simulator. Carla Documentation. `https://carla.readthedocs.io/en/stable/`. [Online; accessed 13-September-2022].

[98] William Norris. *Modern steam road wagons*. Longmans, 1906.

[99] Prabhjot Kaur, Samira Taghavi, Zhaofeng Tian, and Weisong Shi. A survey on simulators for testing self-driving cars. 2021.

[100] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Ng. Ros: an open-source robot operating system. volume 3, 2009.

[101] L.H. Etzkorn. *Introduction to Middleware: Web Services, Object Components, and Cloud Computing*. CRC Press, 2017.

[102] Yuya Maruyama, Shinpei Kato, and Takuya Azumi. Exploring the performance of ros2. In *2016 International Conference on Embedded Software (EMSOFT)*, pages 1–10, 2016.

[103] Yeon Kang, Donghan Kim, and Kwangjin Kim. Urdf generator for manipulator robot. In *International Conference on Robotic Computing (IRC)*, pages 483–487, 2019.

[104] Khronos Group. OpenGL Overview. `https://www.khronos.org/opengl/`. [Online; accessed 13-September-2022].

[105] Microsoft. DirectX graphics and gaming. `https://learn.microsoft.com/en-us/windows/win32/directx`. [Online; accessed 13-September-2022].

[106] Khronos Group. Vulkan. `https://www.vulkan.org`. [Online; accessed 13-September-2022].

[107] Epic Games. Unreal Engine. `https://www.unrealengine.com/en-US`. [Online; accessed 13-September-2022].

[108] Unity. Unity Engine. `https://unity.com`. [Online; accessed 13-September-2022].

[109] Nvidia. PhysX SDK. `https://developer.nvidia.com/physx-sdk`. [Online; accessed 13-September-2022].

[110] OBRUM Sp. z o.o. Modułowy System Autonomii. `https://obrum.pl/oferta/system-sterowania-pojazdem-autonomicznym/`. [Online; accessed 21-January-2022].

[111] A. G. Dahlan et al. The research of 3d modeling between visual & creativity. *International Journal of Innovative Technology and Exploring Engineering*, 8(11S2):180–186, 2019.

[112] Grzegorz Szwoch. Synteza i obróbka obrazu. modelowanie obiektów 3d. 2020.

[113] Ireneusz Hallmann. *Lokalizacja robota mobilnego względem automatycznie wybieranych obiektów*. Rozprawa doktorska, Polska Akademia Nauk, Instytut Podstawowych Problemów Techniki, Warszawa, 2007.

[114] Łukasz Sobczak, Katarzyna Filus, Adam Domański, and Joanna Domańska. Lidar point cloud generation for slam algorithm evaluation. *Sensors*, 21(10), 2021.

[115] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2d lidar slam. pages 1271–1278, 2016.

[116] N. Jamwal, N. Jindal, and K. Singh. A survey on depth map estimation strategies. pages 1–5, 2016.

[117] Wikimedia Commons. Diagram of Earth Centered, Earth Fixed coordinates in relation to latitude and longitude. Creative Commons (CC BY-SA 3.0). `https://commons.wikimedia.org/wiki/File:ECEF.svg`. [Online; accessed 30-January-2022].

[118] Martin Bischoff. ROS# User Documentation. `https://github.com/siemens/ros-sharp/wiki`. [Online; accessed 19-January-2022].

[119] Rainer Kümmerle, Bastian Steder, Christian Dornhege, Michael Ruhnke, Giorgio Grisetti, Cyrill Stachniss, and Alexander Kleiner. On measuring the accuracy of slam algorithms. *Autonomous Robots*, 27(4):387, 2009.

[120] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 573–580, 2012.

[121] Berthold KP Horn. Closed-form solution of absolute orientation using unit quaternions. *Josa a*, 4(4):629–642, 1987.

[122] Changqian Yu, Jingbo Wang, Chao Peng, Changxin Gao, Gang Yu, and Nong Sang. Bisenet: Bilateral segmentation network for real-time semantic segmentation, 2018.

[123] C. Rasmussen. Grouping dominant orientations for ill-structured road following. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 1, pages I–I, 2004.

[124] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2013.

Dodatek A

Symbole i skróty przyjęte w pracy

Jeśli w tekście nie wykazano inaczej, stosowane symbole i skróty należy rozumieć jako:

SLAM - Simultaneous Localization And Mapping

BPL - Bezzałogowa Platforma Lądowa

GPS - Global Positioning System

ADAS - Advanced Driver Assistance Systems

AI - Artificial Intelligence

LiDAR - Light Detection And Ranging

RADAR - Radio Detection And Ranging

IMU - Inertial Measurement Unit

GNSS - Global Navigation Satellite System

MEMS - Microelectromechanical System

ROS - Robot Operating System

WST - Wirtualny System Testujący

MSA - Modułowy System Autonomii

ECEF - Earth-centered, Earth-fixed coordinate system

LLA - Latitude, Longitude, Altitude coordinate system

UDP - User Datagram Protocol

TCP - Transmission Control Protocol

RPE - Relative Pose Error - względny błąd pozycji

ATE - Absolute Trajectory Error - całkowity błąd trajektorii