Katedra Podstaw Konstrukcji Maszyn

Wydział Mechaniczny Technologiczny

POLITECHNIKA ŚLĄSKA W GLIWICACH

PRACA DYPLOMOWA MAGISTERSKA

System sterowania robotem eksploracyjnym bazujący na głębokich sieciach neuronowych

inż. Paweł POLNIK

Kierunek studiów: *Automatyka i Robotyka*

Specjalizacja: *Projektowanie robotów i urządzeń automatyki*

PROMOTOR

dr inż. Marcin JANUSZKA

OPIEKUN

mgr inż. Mateusz KOSIOR

GLIWICE 2022

Katedra Podstaw Konstrukcji Maszyn

Wydział Mechaniczny Technologiczny

POLITECHNIKA ŚLĄSKA

PRACA DYPLOMOWA MAGISTERKA

RMT6 – 4015/21/22

Student:	Wydział:	Kierunek:	Rodzaj studiów:	Semestr:	Specjalizacja:	Rok akademicki:
Paweł POLNIK	MT	AIR	N2z	II	AB5	2021/2022

Temat pracy:

System sterowania robotem eksploracyjnym bazujący na głębokich sieciach neuronowych

W szczególności należy:

1. Uszczegółowienie koncepcji.
2. Utworzenie środowisko testowego i zbioru przykładów uczących
3. Opracowanie algorytmów sterowania
4. Badania weryfikacyjne
5. Budowa platformy robota eksploracyjnego.
6. Analiza wyników badań i sformułowanie wniosków.

Promotor:	Podpis:	Kierownik jednostki:	Podpis:
dr inż. Marcin Januszka		**dr hab. inż. Marek Wyleżoł, prof. PŚ**	
Opiekun:	Podpis:	Data wydania tematu: 08.11. 2021 r.	Planowany termin zakończenia: 27.06. 2022 r.

Spis treści

Rozdział 1

Wstęp

Autonomia pojazdów przez wiele lat była bardzo ograniczonym pojęciem wykorzystywanym głównie przez wyspecjalizowane zespoły badawcze, lecz rozpatrywana w kontekście futurystki [77]. Jednak lata sześćdziesiąte XX wieku to czas wyścigu kosmicznego i autonomia przeniosła się na płaszczyznę pojazdów kosmicznych zwanych łazikami [78]. W rezultacie James Adams stworzył wózek Stanford [46] przedstawiony na rys 1.1. wyposażony w kamery i zaprogramowany do autonomicznego wykrywania i śledzenia linii na ziemi. Było to pierwsze zastosowanie kamer w pojazdach autonomicznych — kluczowego elementu w dzisiejszych pojazdach autonomicznych. Rozwój autonomii trwał głównie w laboratoriach akademickich poprzez realizację projektów m.in.: niemieckiego inżyniera Ernsta Dickamnna o nazwie „VaMoRs" [47] oraz projekt Navlab [48] w Carnegie Mellon University.

Rys. 1.1. Wózek laboratoryjny sztucznej inteligencji Stanford, 1964-71 [16]

Na początku XXI wieku przemysł pojazdów autonomicznych był w pełnym rozkwicie. Dział badawczy Departamentu Obrony Stanów Zjednoczonych - DARPA [49] sponsorował serię wyzwań mających na celu przyspieszenie autonomicznych samochodów. Pierwszy rocznik uczestników poniósł sromotną porażkę pokonując zaledwie kilka mil zanim się rozbił. Jednakże pięć zespołów z powodzeniem ukończyło kurs Grand Challenge 2005 [50], który odbył się zaledwie 18 miesięcy później. Najszybszy

zespół pokonał trasę kalifornijskiej pustyni Mojave w niecałe siedem godzin. Kolejne trzy najszybsze pojazdy ukończyły kurs w ciągu następnych 35 minut. Zwycięskie pojazdy zostały przedstawione na rys 1.2.

Rys. 1.2 Zwycięskie pojazdy DARPA Grand Challenge od lewej – zwycięzca Stanley Racing Team, zdobywcy trzeciego miejsca H1ghlander oraz drugiej pozycji Sandstorm® [50]

Obecnie badania autonomii samochodów bardzo rozwijają firmy motoryzacyjne tj. Ford, Mercedes-Bens, BMW, Tesla, a także firmy technologicznej jak Nvidia oraz Google z swoim samochodem Waymo. Od 2021 r. najbliższą firmą, która wprowadziła na rynek samochody autonomiczne jest Tesla z pakietem Full Self-Driving [51] umożliwiającym autonomiczne sterowanie bez użycia rąk podczas jazdy po autostradach.

Drogi publiczne to jednak skomplikowane miejsca nawigacji, gdyż występuje wiele probalistycznych ograniczeń, na które algorytmy nawigacji muszą reagować w czasie rzeczywistym, tj. przechodnie, światа uliczne oraz rozpraszające billboardy. Przeważnie w bliskiej odległości znajdują się inne pojazdy, budynki oraz drzewa. Także elementem nie do przewidzenia są zachowania innych kierowców w środowisku drogowym nie zawsze zgodne z przepisami drogowymi. Z tego powodu nawet wśród ww. przełomowych badań nie ma dostępnych na rynku w pełni autonomicznych pojazdów do jazdy w warunkach drogowych. Rozwijane są technologie wspomagania kierowcy przy parkowaniu, asystentów pasów ruchu oraz prowadzone są badania poczynają kroki do utworzenia samochodów osobowych bądź też busów, jednak jest to jeszcze odległa perspektywa, gdy te pojazdy staną się naszym stałym elementem codzienności [80].

Z racji tego środowiskiem bardziej przystawalnym do rozwoju systemów autonomicznych są tereny z ograniczony ruchem oraz obecnością człowieka. W branżach takich jak górnictwo działają już setki pojazdów autonomicznych [79]. W maju 2021 roku autonomiczna ciężarówka Caterpillar bezpiecznie przewiozła ponad miliard ton materiału w ciągu zaledwie 8 lat eksploatacji [52]. Rozwój i udane implementacje pojazdów autonomicznych następują w warunkach poza drogowych. Dobitnym przykładem jest rozwój badań oraz udane misje marsjańskich łaziki eksploracyjne tzw. łazików marsjańskich tj. Curiosity [54] oraz Perseverance [53].

Eksploracja Marsa przez człowieka w stosunkowo niedalekiej przyszłości staje się coraz bardziej prawdopodobna. Amerykańska Narodowa Agencja Aeronatuki i Przestrzeni Kosmicznej – NASA bada możliwe misje załogowe [56], a zainteresowanie Marsem jest szerokie ze względu na takie czynniki jak prowadzona debata na temat dowodów na istnienie życia na Marsie [55]. Kolonizacja Marsa to jednak szereg wyzwań, które muszą rozwiązać inżynierowie. Aby przyszłe misje były bezpieczne i efektywne, należy przewidzieć problemy i rozwinąć technologie, które będą potrzebne do ich rozwiązania. Zadania przewidziane dla autonomicznych robotów eksploracyjnych to m.in :

- przygotowanie drogi dla ludzi,
- zbadanie terenu,
- przewóz materiałów.

Misje te będą trwały przez długi czas a komunikacja z Ziemią będzie rzadka i występować będzie z dużym opóźnieniem. Ręczne sterowanie łazikami z Ziemi wiązałoby się z ogromnymi kosztami i osiągnięciem znacznie niższych zwrot z inwestycji ze względu na czas bezczynności łazika w oczekiwaniu na instrukcje. Roboty eksploracyjne generują znacznie więcej informacji niż mogą przekazać i z racji tego dużo bardziej prawdopodobne jest wystąpienie problemu w znacznie szybszym czasie, niż możliwe uzyskanie pomocy od operatora [81].

Z racji przedstawionych informacji występuje zapotrzebowanie na inżynierów zajmujących się autonomią. Zmusza to więc środowiska akademickie do opracowania metod szkoleniowych, które skutecznie sprostają wyzwaniom dydaktycznym w tej rozwijającej się dziedzinie. Nauczanie autonomii stanowi wyzwanie ze względu na czynniki związane z tym jaki występuje sposób nauczania oraz jakie zasoby są do tego potrzebne. Robotyka jest dziedziną interdyscyplinarną: uczeń podchodzący do

projektowania, konstruowania i obsługi robota będzie potrzebował teoretycznej i praktycznej wiedzy z zakresu matematyki, fizyki, teorii sterowania, wizji komputerowej, a także mechaniki, oprogramowania i elektrotechniki. Dodatkowe wyzwania wynikają z konieczności ścisłego połączenia sprzętu i oprogramowania, które jest niezbędne do działania robota. Silny eksperymentalny charakter robotyki unieważnia tradycyjne podejście do nauczania w którym uczniowie są bierni. Uczenie się jest bardziej efektywne, gdy uczniowie aktywnie angażują się w działania mające na celu praktyczne zdobycie umiejętności poprzez studiowanie tematyki, rozwijanie, eksperymentowanie oraz implementację algorytmów na fizycznych platformach. Z racji tego istnieje uzasadniona potrzeba utworzenia platformy będącej podstawą do nauki i rozwoju autonomicznych systemów sterowania w kontekście robotów eksploracyjnych z szczególnym uwzględnieniem marsjańskich robotów eksploracyjnych tzw. łazików marsjańskich.

1.1. Cel pracy

Celem pracy dyplomowej magisterskiej jest utworzenie systemu sterowania robotem eksploracyjnym bazującego na głębokich sieciach neuronowych. Oczekiwanym rezultatem pracy jest funkcjonalny system sterowania oparty na wytrenowanych architekturach AlexNet [26] oraz ResNet [28], zaimplementowany na platformie robota eksploracyjnego Orzeł 7 opracowanej w ramach pracy przejściowej [9].

1.2. Zakres pracy

Pierwszym zadaniem w realizacji pracy dyplomowej był szczegółowy przegląd literatury w zakresie dostępnych platform robotycznych do rozwoju systemów sterowania bazujących na głębokich sieciach neuronowych oraz konwolucyjnych głębokich sieci neuronowych w kontekście systemów sterowania poprzez wykorzystanie obrazu z kamer. W ramach pracy magisterskiej przeprowadzona została analiza zagadnień dotycząca uczenia głębokiego w przetwarzaniu obrazu. Następne zadanie obejmowało uszczegółowienie koncepcji robota eksploracyjnego. Kolejno utworzono tory testowe w warunkach wewnętrznych oraz zewnętrznych w celu weryfikacji poprawności konstrukcji robota eksploracyjnego oraz utworzenia zbioru danych uczących. Czwarte zadanie polegało na wytrenowaniu modelu sieci neuronowej oraz przeprowadzeniem badań nad optymalizacją hiperparametrów. Kolejnym zadaniem była implementacja wytrenowanego modelu na platformie robotycznej. Ostatnie zadanie obejmowało wykonanie badań weryfikacyjnych utworzonego systemu.

1.3. Założenia pracy

W toku realizacji pracy dyplomowej przyjęto następujące założenia:

- możliwość bezpośredniego połączenia z robotem eksploracyjnym niezależnie od zewnętrznych punktów dostępu,
- układ Jetson Nano powinien być wykorzystywany w pełnym zakresie parametrów aby pracować wydajnie podczas operacji klasyfikacji zajętości drogi,
- utworzenie torów testowych w celu wygenerowania danych uczących,
- tor testowy w warunkach zewnętrznych powinien być symulacją warunków marsjańskich,

- należy przeprowadzić badania nad hiperparametrami nauki wybranych architektur sieci neuronowych,
- możliwość prowadzenia eksploracji w warunkach dziennych i nocnych,
- wykorzystanie drukarki 3D w technologii FDM oraz SLA do wytworzenia elementów,
- założenia dotyczące kwestii systemowych zawarte zostały w podrozdziale 6.1.

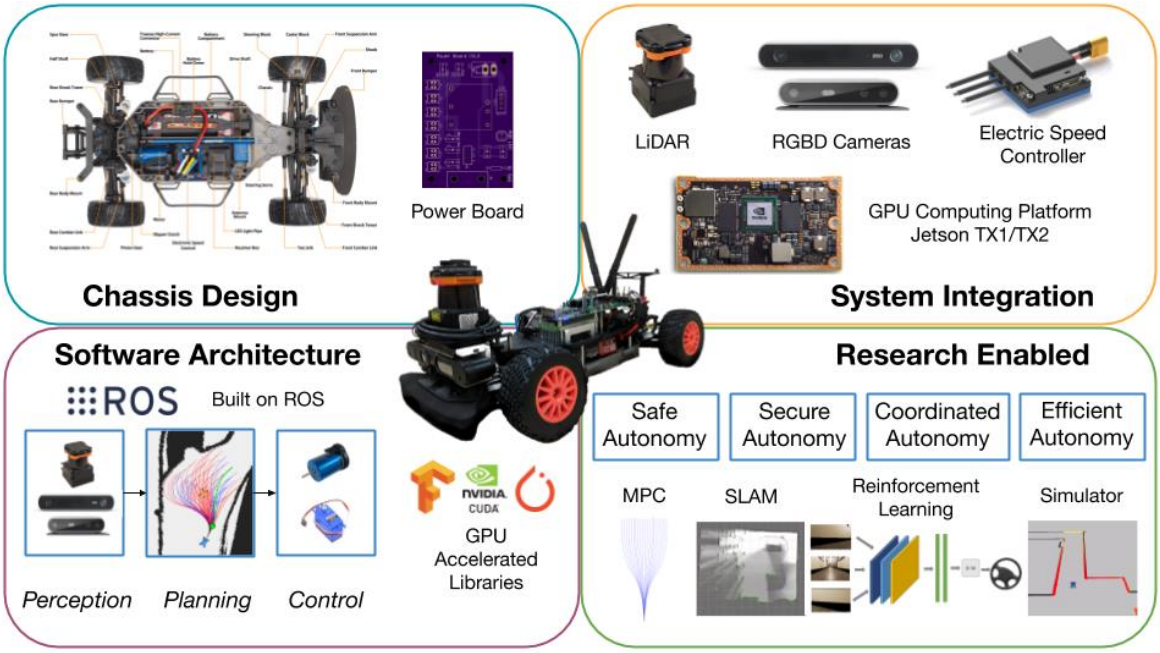
Rozdział 2

2. Przegląd istniejących rozwiązań

Badacze rynku przewidują, że do 2025 r. na drogach pojawi się ponad 20 milionów autonomicznych pojazdów [57], co z kolei oznacza, że istnieje potrzeba utworzenia łatwej i ogólnie dostępnych platform do nauki ww. systemów. Podyktowane jest to zdobyciem praktycznej wiedzy dla potencjalnych pracowników w zbliżających się potrzebach na rynku pracy. Małe autonomiczne platformy powstają w wielu różnych konfiguracjach przystosowanych do badania konkretnych problemów z dedykowanym oprzyrządowaniem. W środowiskach naukowych często do testowania wykorzystywane są modele w odpowiedniej skali np. 1/10 z względu na aspekty kosztów oraz bezpieczeństwa. Platformy do rozwoju systemów autonomicznej jazdy są więc szansą na naukę i eksperymentowanie z najnowocześniejszymi technologiami. Zgodnie z wykonanym przeglądem literatury oraz technologami występujący na rynku zauważono, że powstało wiele platform do nauki autonomii w kierunku samochodów autonomicznych jednak występuje problem w kwestii platform do rozwoju systemów autonomicznych w robotach eksploracyjnych. Wyniki przeprowadzonego przeglądu przedstawiono w poniższych podrozdziałach.

2.1. F1TENTH

F1TENTH [58] to platforma, która została pierwotnie założona na Uniwersytecie Pensylwanii w 2016 roku jednak obecnie rozwijania jest przez międzynarodową społeczność naukowców, inżynierów i entuzjastów systemów pojazdów autonomicznych. Pojazd oparty jest na podwoziu oferowanym przez Traxxas Slash 4x4 Premium Chassis natomiast jednostką obliczeniową jest Nvidia Jetson Xavier NX. Wyposażony jest w zestaw czujników tj. lidar, IMU, obwód VESC do kontroli i regulacji prędkości silnika elektrycznego i serwonapędów oraz opcjonalnie kamerę Stereloabs ZED z czujnikiem głębi i ruchu. Elementy składowe platformy F1TENTH zostały przedstawione na rys 2.1.



Rys. 2.1. Elementy składowe platformy F1TENTH [58]

Architektura oprogramowania dostępna jest w trybie open source w repozytorium GitHub. Opierając się na rozwiązaniach takich jak Linux, C++, Python™, TensorFlow, PyTorch oraz ROS Studio umożliwia tworzenie aplikacji wysokiego poziomu i rekonfigurację procesów niskiego poziomu, które są obsługiwane przez gotowe moduły i biblioteki. Platforma jest wydajna obliczeniowo i wszechstronna, dlatego można ją wykorzystywać do różnych badań, które obejmują między innymi wyścigi autonomiczne, uczenie się, robotykę, systemy komunikacji i wiele innych. Dużym atutem projektu jest także dostępność kursu F1TENTH CourseKit [59] uczącego podstaw autonomii kładącego nacisk na umiejętności analityczne w projektowaniu pojazdów autonomicznych.

2.2. AutoRally Robot

Platforma AutoRally (rys. 2.2) to wysokowydajne środowisko testowe przeznaczone do prowadzenia badań nad autonomią pojazdów. Projekt został opracowany w 2011 roku w Georgia Tech przez Briana Goldfaina, Paula Drewsa i Jamesa Regha [60] w celu badań nad systemami wizyjnymi wykorzystywanymi w pojazdach autonomicznych wspartych przez amerykański program DURIP (ang. Defense University Research

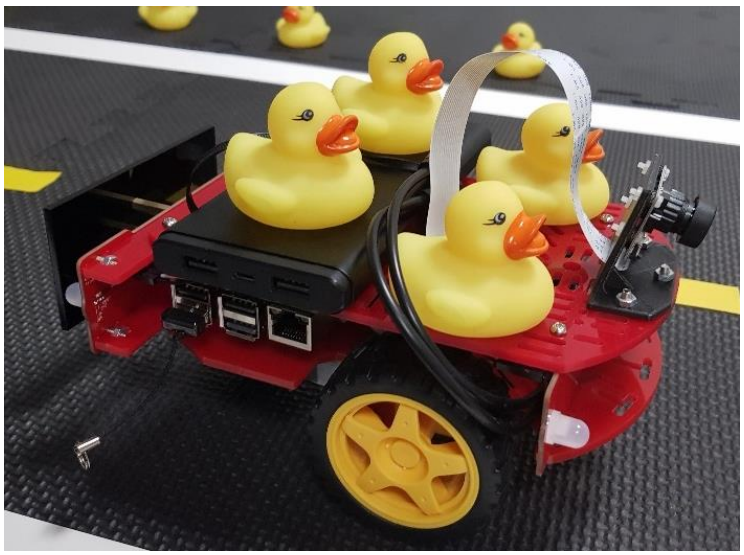
Instrumentation Program). Platforma została zaprojektowana jako samodzielny pojazd z uwzględnieniem solidności i trwałości konstrukcji a także łatwości użytkowania i montażu.

Rys. 2.2. Platforma AutoRally z widokiem od lewej - a) podwozia, b) złożonego pojazdu, c) jednostki obliczeniowej wraz z urządzeniami peryferyjnymi [60]

Platforma pojazdów autonomicznych AutoRally jest oparta na ciężarówce RC w skali 1:5. ma około 1 [m] długości, 0,6 [m] szerokości, 0,4 [m] wysokości, waży prawie 22 [kg] a jej prędkość maksymalna wynosi 90 [km/h]. Platforma jest zdolna do autonomicznej jazdy wykorzystując wyłącznie pokładowe czujniki, komputery i zasilanie. Platforma jest większa niż wiele innych skalowanych autonomicznych pojazdów naziemnych, ale stanowi efektywną kosztowo, wytrzymałą, wydajną i bezpieczną alternatywę dla pełnowymiarowych pojazdów autonomicznych pojazdów i zachowuje dużą ładowność w porównaniu z innymi skalowanymi platformami zbudowanymi z mniejszych samochodów RC. Możliwości AutoRally oferują znaczną poprawę osiągów w stosunku do tradycyjnych skalowanych pojazdów autonomicznych bez konieczności dużych inwestycji w infrastrukturę i względów bezpieczeństwa wymaganych w przypadku pełnowymiarowych pojazdów autonomicznych. Kompletna instrukcja budowy i konfiguracji platformy AutoRally jest publicznie dostępna oraz zawiera kompletną listę części oraz schematy połączeń, a także wszystkie wymagane pliki CAD (ang. computer-aided design) do produkcji części na zamówienie. Ponadto procedury operacyjne, środowisko symulacyjne, oprogramowanie podstawowe i sterowniki referencyjne napisane są w językach C++ oraz Python wraz z kolekcją danych dotyczących jazdy autonomicznej udostępnionych w repozytorium GitHub [61]. Platforma robotyczna jest solidną, ekonomiczną i bezpieczną platformą, która otwiera przed naukowcami i hobbystami przestrzeń agresywnej autonomicznej jazdy terenowej.

2.3. Duckiebot

Duckiebot [62] jest atrakcyjną i przystępną cenowo platformą badawczą służącą do rozwoju i nauki systemów autonomicznych oraz robotyki. Projekt zaprezentowano w 2016 roku w Massachusetts Institute of Technology. Elementami składowymi są komponenty ogólnodostępne na rynku. Obliczenia są wykonywane przy wykorzystaniu minikomputera Raspberry PI w wersji 2 (RPi2) wyposażonym w 4 rdzenie ARM 900 [MHz] i 1 [GB] pamięci RAM. Napęd realizowany jest poprzez dwa koła napędzane silnikami prądu stałego. Podwozie jest wymienną częścią robota i może zostać zastąpione przez dowolną konfigurację, która wykorzystuje dwa silniki prądu stałego. Duckieboty (rys. 2.3) wyczuwają świat za pomocą tylko jednej kamery dedykowanej do Raspberry. Konfiguracja, komunikacja, debugowanie i opcjonalne sterowanie ręczne Duckiebotem odbywa się przez WiFi. Dla dużych wdrożeń rozwiązaniem występującego problemu nasycenia sieci jest wbudowany punkt dostępowy w każdym Duckiebocie. Te mobilne hotspoty tworzą sieć o częstotliwości 5 [GHz], są zasilane niezależnie i łączą się z Raspberry PI 2 przez Ethernet. Ponadto można użyć bezprzewodowego joysticka dzięki czemu sterowanie ręczne staje się wygodniejsze. Duckieboty mogą być także wyposażone w diody LED RGB, aby umożliwić globalne zachowania, które wymagają komunikacji między pojazdami w celu orientacji i przekazywania informacji.



Rys. 2.3. Platforma badawcza Duckiebot DB18 [62]

Integralnym elementem projektu jest symulowane środowisko miejskie DuckieTowns [63] służące jako miejsca nauki i operowania robotów. Drogi zbudowane są z mat oraz oznakowane taśmami, po których poruszają się roboty. DuckieTowns (rys. 2.4) mogą zostać przekształcone w inteligentne miasta poprzez dodanie sygnalizacji świetlnej, znaków, przeszkód, skrzyżowań oraz mieszkańców Duckietown (kaczuszek). Projekt ten jest użytecznym narzędziem, ponieważ uczniowie oraz badacze mogą zaoszczędzić pieniądze i czas nie musząc rozwijać całej niezbędnej infrastruktury. Wszystkie materiały są dostępne w trybie otwarto źródłowym.

Rys. 2.4. Środowisko miejskie DuckieTowns [63]

2.4. Jackal UGV

Konstrukcją w kontekście robotów eksploracyjnych dostępnych na rynku jest przemysłowe rozwiązanie w postaci komercyjnej platformy JACKAL UNMANNED GROUND VEHICLE produkowany przez firmę Clearpath Robotics [64]. Projekt powstał jako kontrakt z Laboratorium Badawczym Armii Stanów Zjednoczonych - ARL (ang, US Army Research Labs) na stworzenie niedrogiej, przenośnej platformy badawczej do użytku terenowego. Koszt platformy w zależności od konfiguracji wynosi około 16 800,00 [$]. Jackal UGV (rys. 2.5) to mały, szybki i wytrzymały bezzałogowy pojazd naziemny z w pełni zintegrowanym GPS, IMU i wyposażony w konfigurowalną płytę górną służącą do instalacji m.in ramienia robotycznego. Robot działa na systemie ROS (ang. Robot Operating System). Wyposażony jest w dwie kamery: Axis P5514-E PTZ oraz kamerę sieciową Point Grey Flea3 FL3-U3-13E4C-C. Urządzenie posiada konstrukcję

wodoodporną i napęd typu skid-steer, który pozwala na poruszanie się po każdym terenie. Jackal ma wymiary 43 [cm] szerokości, 51 [cm] długości, 25 [cm] wysokości i około 17 [kg] masy oraz posiada 4 koła i generuje 500 watów mocy. Robot po odpowiedniej konfiguracji przeznaczony jest do pracy w wielu trudnych warunkach tj. place budowy, rafinerie ropy naftowej oraz tunele kolejowe.

Rys. 2.5. Jackal UGV produkcji Clearpath Robotics z zabudowaną kamerą StereoLabs ZED [64]

2.5. Podsumowanie

Tradycyjnie skalowane platformy do jazdy autonomicznej były budowane specjalnie na potrzeby jednego eksperymentu, ale pojawia się nowa fala platform otwarto źródłowych. Platformy powinny być łatwe w budowie, umiarkowanie tanie i oferować pokładowe możliwości detekcji i wykonywania obliczeń. Rozwój platform ma miejsce głównie w obszarze pojazdów do jazdy w warunkach zbliżonych do drogowych tj. F1TENTH, Dockiebot lub JetBot opisany szerzej w pracy przejściowej [9]. Występują także próby dostosowania pojazdów do jazdy w warunków nieutwardzonej drogi np. platforma AutoRally. Jednak żaden z otwarto-źródłowych projektów nie posiada wystarczających parametrów pod względem jazdy eksploracyjnej w warunkach

zewnętrznych. Pojazdami o takich parametrach jest przedstawiony komercyjny projekt Jackal UGV. Zważając na powyżej przedstawione informacje, istnieje potrzeba utworzenia platformy będącej podstawą do nauki i rozwoju autonomicznych systemów sterowania w kontekście robotów eksploracyjnych ze szczególnym uwzględnieniem marsjańskich robotów eksploracyjnych, tzw. łazików marsjańskich. Dostępne obecnie platformy umożliwiają przede wszystkim rozbudowę w kierunku samochodów wyścigowych opartych na platformach 4-kołowych lub prostych 2-kołowych konstrukcjach. Natomiast w realizowanej pracy magisterskiej zdecydowano się wykorzystać niewystępującą na rynku 6-ścio kołową platformę robotyczną z zawieszeniem typu rocker-bogie z napędem na każde koło wzorowaną na ostatnich konstrukcjach Amerykańskiej Agencji Kosmicznej NASA, tj. łazikach Curiosity oraz Perseverance. Platformę tą opracowano w ramach pracy przejściowej i przedstawiono na rys 2.6.

Rys. 2.6. Widok platformy robotycznej do nauki i rozwoju autonomicznych systemów sterowania w kontekście robotów eksploracyjnych [zdjęcie: Marcin Januszka]

Rozdział 3

3. Uczenie głębokie w przetwarzaniu obrazu

W niniejszym rozdziale przedstawiono opis podstawowych pojęć związanych z szeroko pojętą sztuczną inteligencją w kontekście analizy obrazu przy wykorzystaniu sztucznych sieci neuronowych oraz głębokich sieci neuronowych. Typem głębokich sieci neuronowych wykorzystywanych do analizy obrazów są sieci konwolucyjne zwane również splotowymi będących nieodzowną częścią składową systemów sterowania bazujących na informacjach pozyskiwanych z kamer.

3.1. Rozpoznawanie obrazu oraz występujące ograniczenia

Próbkowanie i kwantowanie sygnału wizyjnego powoduje powstanie obrazu cyfrowego z najmniejszym jednolitym elementem składowym w postaci piksela. Posiada on określoną jasność różnych kolorów, a jego barwa określana jest poprzez modele przestrzeni barw jak np. RGB, RGBA, CMYK, HSV itp. [3]. Przeprowadzanie operacji analizy obrazu poprzez zastosowanie filtrów oraz operacji arytmetycznych i geometrycznych na pikselach pozwala uwidocznić charakterystyczne cechy oraz uzyskać informacje o obrazie. Człowiek posiada naturalną zdolność do rozpoznawania i analizy obrazu poprzez poszerzanie swojej wiedzy przez lata doświadczeń identyfikacji różnych obiektów. Bardzo rozbudowane zdolności adaptacyjne ludzkiego zmysłu wzroku są bardzo trudne do otworzenia w warunkach inżynierskich. Z tego powodu występuje problem jak przetwarzać i reprezentować ogromną ilość ludzkiej wiedzy w komputerze w taki sposób, aby było to przejrzyste i intuicyjne [2]. W przypadku maszyn następuje porównanie obrazu wejściowego do wcześniej dostarczonych wzorców z przygotowanych zestawów odpowiednio opisanych danych [4]. Rozpoznawanie obrazu można rozróżnić na procesy segmentacji, klasyfikacji, kategoryzowania oraz lokalizacji obiektów. W zadaniach autonomicznego przejazdu robotów występuje problem w dziedzinie zarówno sprzętowej jak i oprogramowaniu w kwestii przeprowadzenia ogromnej ilości obliczeń, które są wymagane aby zadanie np. rozpoznania bądź klasyfikacji było wykonano poprawnie w czasie rzeczywistym [2].

3.2. Sztuczne sieci neuronowe

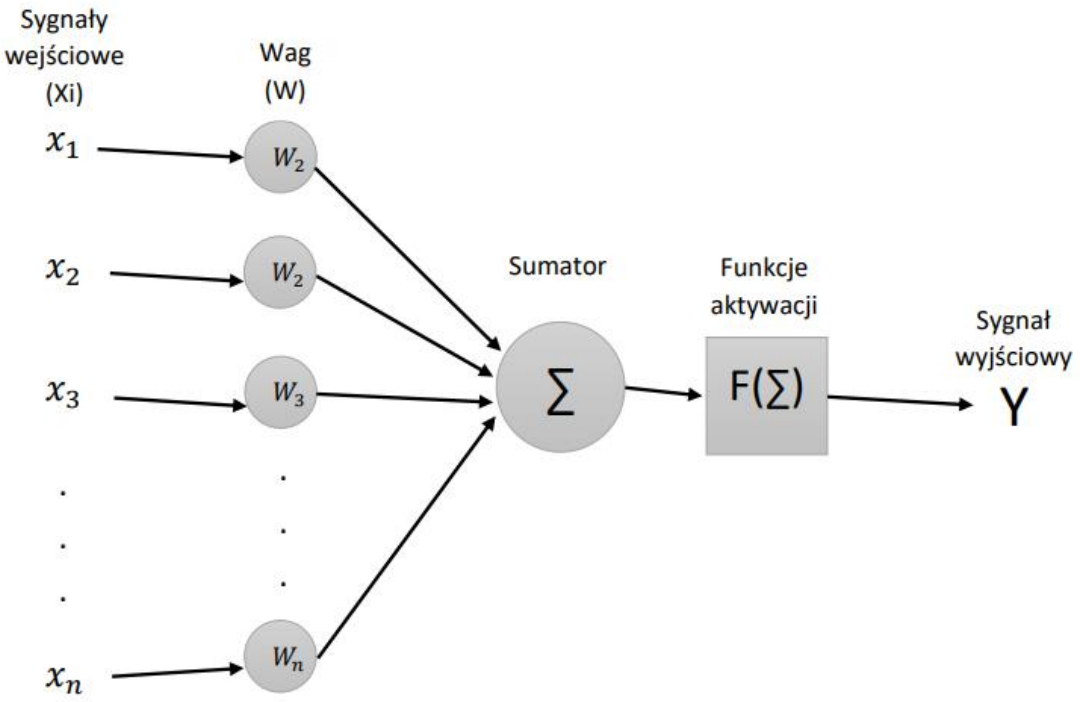
Sztuczne sieci neuronowe (ang. ANN – Artifical Neural Network) zostały utworzone jako rozszerzenie zdobytej wiedzy naukowców w kontekście badań na temat działania ludzkiego umysłu. Z tego powodu sieć neuronów opiera się na symulowaniu aktywności elektrycznej mózgu oraz układu nerwowego. Elementem przetwarzającym jest neuron. W książce autorstwa Ryszarda Tadesuewicza pt. „Odkrywanie właściwości sieci neuronowych przy użyciu programów w języku C#" [5] koncepcję biologicznego neurony charakteryzuje zdobywca nagrody Nobla - Ramon y Cajla w słowach „wprowadzono koncepcję neuronów jako wyspecjalizowanych komórek przetwarzających informację, odbierających i analizujących wrażenia zmysłowe, a także wypracowujących i wysyłających sygnały sterujące do wszystkich elementów, które w ciele człowieka znajdują się pod kontrolą mózgu, mięśni sterujących ruchami ciała, gruczołów oraz innych narządów wewnętrznych" [5]. Inspiracje działaniem struktur ludzkiego mózgu widoczne były w początkowej etapie rozwoju badań sztucznej inteligencji.

W 1943 roku powstała przełomowa koncepcja modelu sztucznego neuronu McCullocha-Pitsa [39] przedstawionego na rys 3.1., będącego podstawą dla wykorzystywanych obecnie złożonych struktur sztucznych sieci neuronowych. Model sztucznego neuronu przekształca wartości wejściowe (cechy wejściowe) na pojedynczą wartość wyjściową. Neuron McCullocha-Pitsa zbudowany jest z następujących elementów:

- Sygnał wejściowy (cecha wejściowa $x_1 - x_n$), któremu przypisywana jest waga ($w_1 - w_n$) – analogia do biologicznych połączeń synaptycznych,
- Sumator ($\sum$) – oblicza sumę iloczynów wartości podanych na wejściu x_i oraz wag w_i - odpowiednik potencjału komórkowego,

$$\Sigma = \sum_{i}^{n} x_i w_i$$

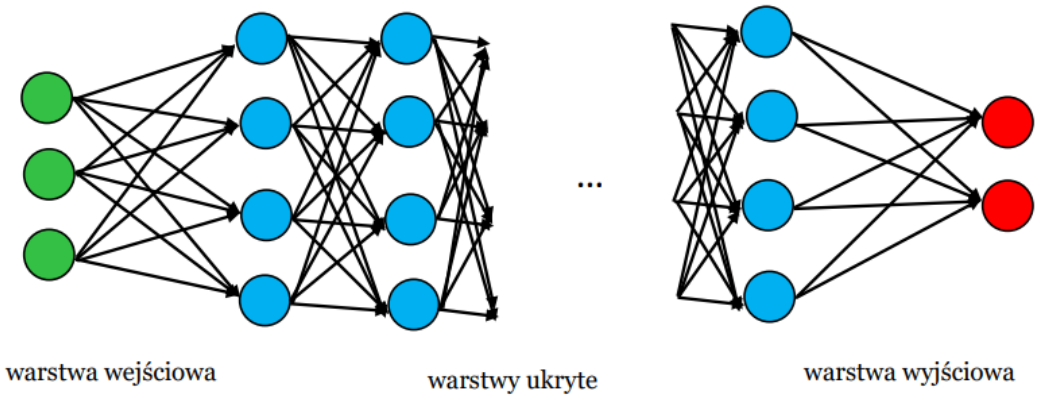
- Funkcja aktywacji $F(\sum)$ – stosowana do obliczonej sumy określająca pojedynczą wartość odpowiedzi neuronu dla sygnałów wejściowych,
- Sygnał wyjściowy (Y) – pojedyncza wartość wyjściowa uzyskana z przeprowadzonych operacji.



Rys. 3.1. Schemat budowy neuronu McCullocha-Pittsa [39]

Na podstawie tej koncepcji w 1957 roku Frank Rosenblatt poprzez połączenie kilku niezależnych neuronów McCullocha-Pitsa utworzył algorytm uczenia nazdzorowanego – perceptron [40]. Idea ta wykorzystana została do stworzenia sprzętowego systemu rozpoznawania prostych kształtów na obrazach o rozmiarze 20 × 20 pikseli. W 1969 roku M.Minsky oraz S.Paper w szeroko komentowanej książce *„Perceptrons”* [41] wykazali ograniczenia perceptronów. W zbudowanej z pojedynczej warstwy neuronów model rozwiązuje wyłącznie zadania klasyfikacji problemów liniowo separowalnych i z tego powodu nie jest możliwe nauczenie tego typu modeli np. odwzorowania funkcji logicznej XOR [27]. Spowodowało to długotrwały impas w rozwoju prac i prowadzenia badań nad sztucznymi sieciami neuronowymi. Dopiero przeprowadzone w 1988 roku badania Rumenthala [42] nad zastosowaniem uczenia z wykorzystaniem propagacji wstecznej błędu (ang. backpropagation) rozpoczęły popularność sieci neuronowych i stały się fundamentalnym sposobem uczenia sztucznych sieci neuronowych wykorzystywanym do dziś. Obecnie występującym celem sztucznych sieci neuronowych nie jest wierne symulowanie działania struktur ludzkiego mózgu, lecz utworzenie możliwie efektywnych matematycznych metod rozwiązywania złożonych problemów poprzez wykorzystanie pewnych upraszczających analogii [27]. Przeciwskazaniem występującym w

odwzorowaniu złożoności działań biologicznych sieci neuronowych są ograniczenia sprzętowe komputerów w kontekście mocy obliczeniowych. Dlatego występują odwzorowania w sposób uproszczony poprzez prostą budowę i podstawowe właściwości [6]. Zalety sieci neuronowych nie ograniczają się jedynie do tego, że umożliwiają one łatwe oraz swobodne tworzenie modeli nieliniowych nie wymagających konieczności samodzielnego formułowania przez użytkownika skomplikowanych hipotez, ale także charakteryzują się szybkością przetwarzania danych. Sieci umożliwiają ponadto kontrolę nad złożonym problemem wielowymiarowości. Sztuczne sieci neuronowe cechują się architekturą posiadającą kolejno połączone warstwy neuronów (rys 3.2).



Rys. 3.2 Architektura sieci neuronowych [7]

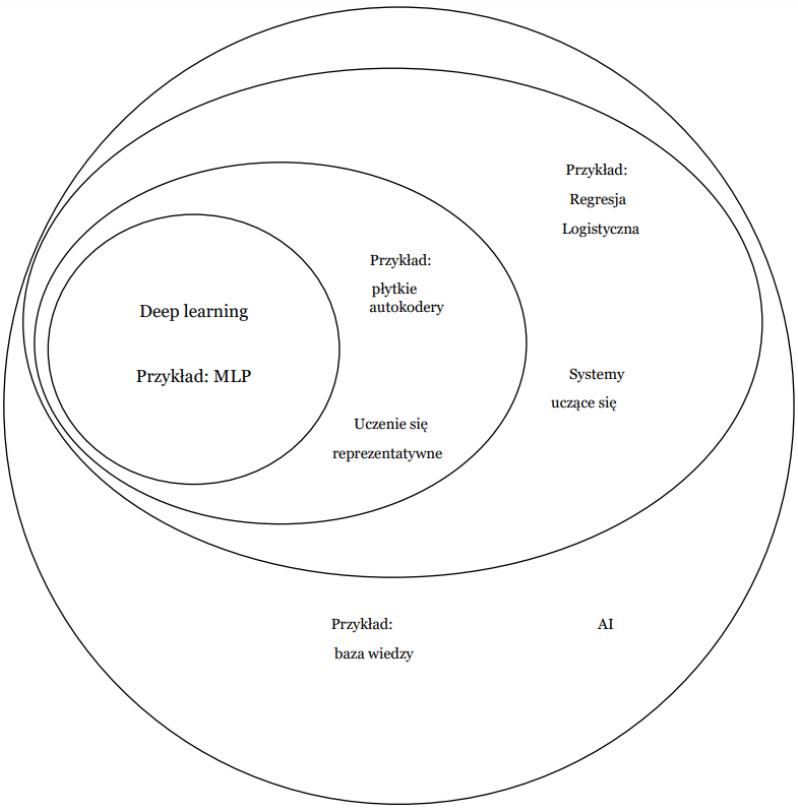
Tak utworzony model można traktować jak złożoną funkcję przyjmującą na wejściu – warstwie wejściowej (ang. input layer) parametry - wagi pozwalającą dopasować model by optymalnie opisywał dane w warstwie wyjściowej (ang. output layer) propagowanej poprzez przejściowej warstwy ukryte (ang. hidden layer). Często w przypadku zadań klasyfikacji zajętości drogi w sterowaniu autonomicznych pojazdów z wykorzystaniem sieci neuronowych zwracane jest prawdopodobieństwo zablokowania możliwości przejazdu. Celem uczenia modelu z zakresu sieci neuronowych jest uzyskanie dla każdej wagi globalnej wartości, dla której wartość straty jest minimalna. W tym celu stosuje się algorytm wstecznej propagacji (ang.backpropagation), który pozwala w wydajny sposób obliczyć pochodne cząstkowe funkcji kosztu względem optymalizowanych parametrów sieci. W trakcie implementacji najczęściej wykorzystuje się uczenie nadzorowane (ang. supervised learning) oznaczające posiadanie zbioru danych

wraz z przypisanymi etykietami odnoszącym się do poprawnie oznaczonych klas tzw. ground truth. Odmiennym podejściem jest uczenie nienadzorowane (ang. unsupervised learning), nie narzucające algorytmowi podczas procesu uczenia żadnych obserwacji (etykiet) lecz pozwalające na samodzielne znalezienie interesujących wzorców czy zależności. Zważając na powyższe informacje można dostrzec wadę sieci neuronowych związaną z koniecznością zgromadzenia dużej ilości danych uczących oraz trudności w dopasowaniu odpowiednich parametrów dla optymalizacji wyników nauki.

3.3. Głębokie sieci neuronowe

Odmianą sieci neuronowych służących do rozpoznania i klasyfikacji obiektów są głębokie sieci neuronowe DNN (ang. Deep Neural Network). Posiadają one wiele warstw ukrytych. W dziedzinie wizji komputerowej klasyczne sieci neuronowej okazały się nieefektywne. Obraz podawany na wejście może charakteryzować się liczbą tysięcy pikseli oraz do 3 kanałów kolorów co z kolei wymaga ogromnej liczby połączeń i parametrów klasycznej sieci neuronowej [22].

Uczenie głębokie stanowi odmienne podejście do tematyki sztucznej inteligencji - AI (ang.artifical intelligence), co przedstawiono na rys 3.2. Ściśle mówiąc, uczenie głębokie to rodzaj systemów uczących się, technika umożliwiająca systemom komputerowym ulepszenie oparte na doświadczeniu i danych. Uczenie głębokie jest tym rodzajem systemów uczących się, który osiąga wielką moc i elastyczność reprezentując świat jako zagnieżdżoną hierarchię pojęć przy czym każde pojęcie jest zdefiniowane w powiązaniu z pojęciami prostszymi i bardziej abstrakcyjnymi reprezentacjami wyznaczonymi za pomocą pojęć mniej abstrakcyjnych [1]. Typowym przykładem modelu deep learningu jest głęboka sieć jednokierunkowa lub wielowarstwowy perceptron - MLP (ang. multi layer perceptron). MLP to funkcja matematyczna dążąca do odwzorowania partii danych wejściowych na wartości wynikowe. Budowa funkcji opiera się na szeregu mniejszych funkcji składowych. Każdorazowe wykorzystanie innej funkcji matematycznej rozumiane jest jako nowa reprezentację danych wejściowych. Jedną z perspektyw deep learningu ma na celu właśnie poznanie właściwej reprezentacji danych.



Rys. 3.2 Diagram Venna wizualizujący technologie AI [1]

Każda warstwa reprezentacji może być rozumiana jako stan pamięci komputera po równoległym wykonaniu kolejnego zbioru instrukcji. Większa głębokość sieci wpływa na większą ilość instrukcji wykonywanych sekwencyjnie. Działanie to powoduje duże możliwości ponieważ kolejno instrukcje posiadają sposobność odwoływania się do uzyskanych wyników wcześniejszych instrukcji. W wyniku takiego podejścia do deep learningu nie wszystkie warstwy informacji muszą koniecznie kodować czynnik zmian, które objaśniają dane wejściowe [1]. Utworzenie programu, który potrafi zrozumieć dane jest możliwe dzięki przechowywaniu informacji o stanie przez reprezentacje. Informacja o stanie jest odpowiednikiem licznika bądź wskaźnika wykorzystywanych w tradycyjnych programach komputerowych lecz nie posiada nic wspólnego z zawartością konkretnych danych. Pomaga natomiast modelowi w organizacji przetwarzania [1].

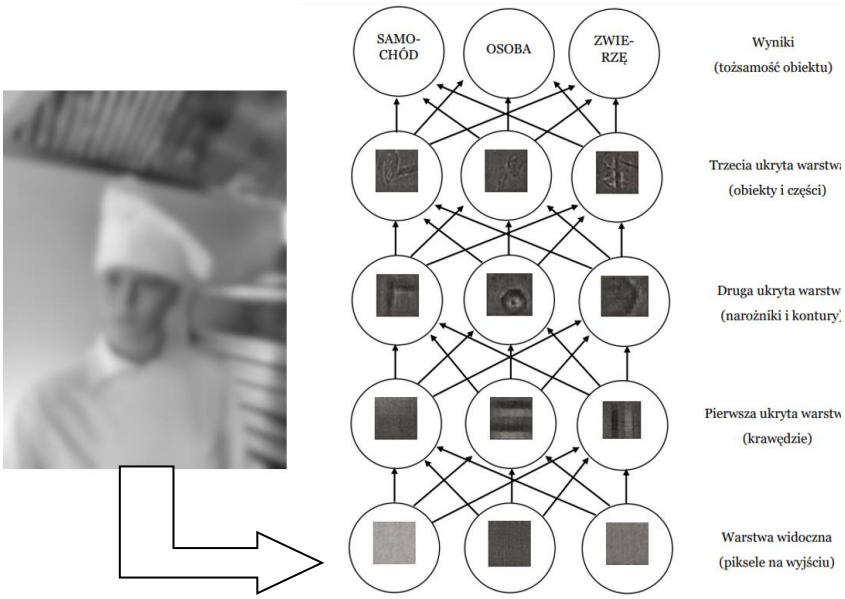
Często podkreślane jest podobieństwo uczenia głębokiego do pracy mózgu. Niezaprzeczalną kwestią jest, że badacze pracujący nad uczeniem głębokim podkreślają wpływ sposobu działania mózgu bardziej niż badacze w innych dziedzinach systemów uczących się, jak w przypadku jądra maszyn lub statystyk bayesowskich, lecz nie należy

widzieć deep learningu jako próby symulacji mózgu [1]. Nowoczesny deep learning czerpie inspirację z wielu dziedzin jak matematyka stosowana w tym algebra liniowa, prawdopodobieństwo, a także teoria informacji i optymalizacja numeryczna. W zależności od konieczności przeznaczenia sieci do konkretnych zadań rozróżnia się m.in. następujące rodzaje:

- Rekurencyjne sieci neuronowe (RNN – recurrent neural networks),
- Konwolucyjne sieci neuronowe (CNN – convolution neural networks),
- Sieci generatywne nienadzorowane np. Restrykcyjna Maszyna Boltzmanna (RBM – Restricted Boltzmann Machine).

3.4. Jak działa deep learning

Czynności będące proste do wykonania przez człowieka dla komputera stanowią trudne do przetworzenia zadanie. Przykładem może być zrozumienie znaczenia surowych danych pochodzących ze zmysłów jak np. obraz reprezentowany przez zbiór pikseli ponieważ funkcja odwzorowująca zbiór pikseli na tożsamość obiektów jest bardzo skomplikowana. Trudność tą rozwiązuje deep learning poprzez podział skomplikowanego odwzorowania na serię zagnieżdżonych prostszych odwzorowań. Każde z nich jest opisywane w oddzielnej warstwie modelu co przedstawiono na rys 3.3.



Rys. 3.3 Ilustracja modelu deep learningu [1]

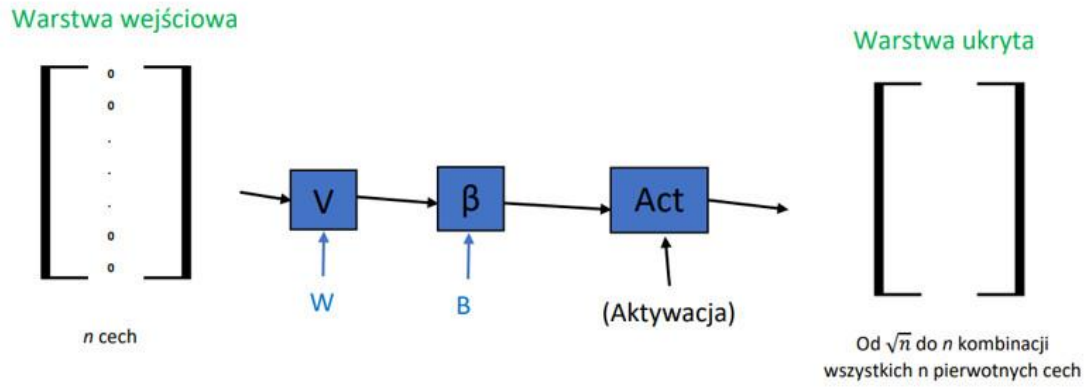
Warstwa zawierająca dane wejściowe w postaci zmiennych, które możemy obserwować nosi nazwę warstwy widocznej. Coraz bardziej abstrakcyjne cechy są uzyskiwane z obrazu poprzez następny ciąg warstw ukrytych. Wartości nie są podawane w danych stąd nazwa ukryte. Model musi sam określić elementy użyteczne w określaniu związków obserwowanych w obrazach. Pierwsza warstwa poprzez znajomość pikseli łatwo identyfikuje krawędzie w wyniku porównania jasności sąsiednich pikseli. Mając opis brzegów zawartych w pierwszej warstwie, druga warstwa może łatwo odnaleźć narożniki i rozszerzone kontury. Mając opis obrazu z drugiej ukrytej warstwy w postaci narożników i konturów, trzecia ukryta warstwa może wykryć całe fragmenty określonych obiektów, dzięki znalezieniu specjalnego zbioru konturów i narożników. Wreszcie ten opis obrazu w kategorii części obiektu, które się na niego składają, może być wykorzystany do rozpoznania obiektów znajdujących się na obrazie [1]. Na podstawie przedstawionych informacji można utworzyć bardziej szczegółowy wysokopoziomowy opis głębokiego uczenia modelu [7] :

I. przekazywanie danych wejściowych do funkcji modelu (krok w przód) w celu uzyskania predykcji,
II. obliczanie liczby reprezentującej wartość straty,
III. obliczanie gradientu (reprezentacji pochodnej cząstkowej funkcji straty względem każdego elementu z danych wejściowych bądź wyjściowych) dla wartości straty względem parametrów z wykorzystaniem reguły łańcuchowej i wartości obliczonych w kroku w przód,
IV. modyfikacje parametrów na podstawie gradientów.

3.5. Konwolucyjne sieci neuronowe

Typem głębokich sieci neuronowych wykorzystywanych do analizy obrazów są sieci konwolucyjne CNN (ang.convolutional neural network) zwane również splotowymi. Sieci CNN są standardową architekturą sieci neuronowych używaną do generowania predykcji, gdy obserwacjami wejściowymi są obrazy [7]. Konwolucyjna sieć neuronowa wykorzystuje różniczkowalną funkcję przekształcając zadany zbiór danych w następny. Splotowe sieci neuronowe biorą pod uwagę fakt, że zdjęcie składa się z mniejszych cech lub detali przez co tworzy mechanizm do analizy każdej z cech osobno wpływając ostatecznie na całościową decyzję [23]. Sieci te ukierunkowane są na przetwarzanie

danych o znanej topologii siatki. Przykładami mogą być dane geograficzne, które można sobie zwizualizować jako dwuwymiarową siatkę pikseli lub dane szeregów czasowych rozumianych jako jednowymiarowa siatka przyjmująca próbki w regularnych odstępach czasu. Splotowe sieci neuronowe znalazły bardzo dobre przeznaczenie w zastosowaniach praktycznych. Zgodnie z nazwą podstawowym elementem wykorzystywanym do działania jest operacja matematyczna zwana splotem. Splot to specjalistyczny rodzaj działania liniowego. Sieci splotowe to sieci neuronowe, które w przynajmniej jednej z warstw zamiast ogólnego mnożenia macierzy wykorzystują splot [1]. Dużą zaletą sieci konwolucyjnych jest umiejętność automatycznego wykrywania ważnych kombinacji pierwotnych danych wejściowych w procesie uczenia. Proces ten rozpoczyna się od utworzenia losowych kombinacji pierwotnych cech w wyniku mnożenia danych przez losową macierz wag. W procesie szkolenia sieć neuronowa uczy się dopracowywać pomocne kombinacje i odrzucać nieprzydane. Taki proces uczenia, w którym istotne są kombinacje cech nazywamy uczeniem reprezentacji przedstawionym graficznie na rys 3.4 [7].



Gdzie:

B – wyraz wolny

β – macierz wyrazów wolnych

W – waga

Act – funkcja aktywacji

V - macierz wag

0 – dane wejściowe

Rys3.4. Przedstawienie procesu uczenia reprezentacji za pomocą sieci neuronowych, które początkowo mają *n* cech, a następnie uczą się od $\sqrt{n}$ do *n* kombinacji tych cech na potrzeby generowania predykcji [7]

W analizie obrazów najczęściej występującą kombinacją jest zbiór cech (pikseli) obejmującą piksele znajdujące się w bliskiej odległości od siebie. Oznacza to, że prawdopodobieństwo wystąpienia wartościowej cechy jest dużo wyższe w grupie przyległych do siebie pikseli o wymiarach 3x3 w porównaniu do grupy 9 losowo wybranych pikselach. Celem jest wykorzystanie tej własności danych graficznych ponieważ kolejność cech ma znaczenie, gdyż informują, które piksele znajdują się blisko siebie w przestrzeni. Natomiast kolejność nie jest już tak istotna w przypadku innej postaci danych np. ceny domów.

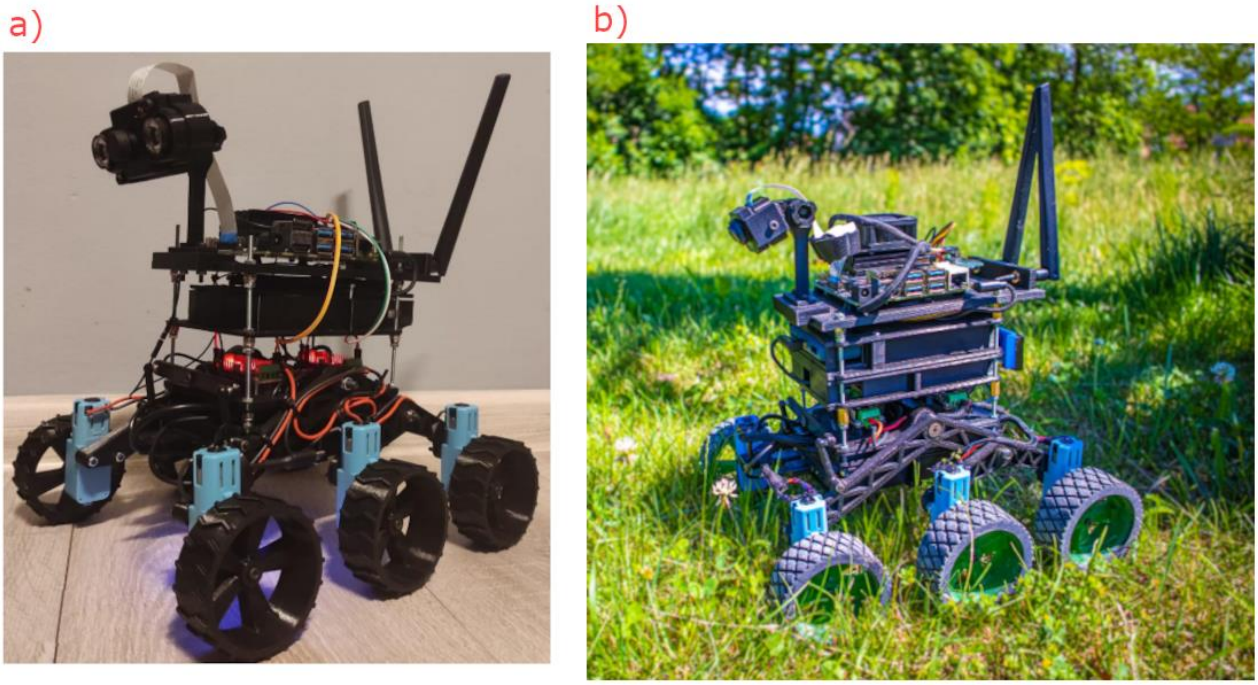
Rozdział 4

4. Uszczegółowienie koncepcji

W rozdziale przedstawiono konstrukcję platformy robotycznej do nauki i rozwoju autonomicznych systemów sterowania w kontekście robotów eksploracyjnych oraz wprowadzone modyfikacje zwiększające komfort pracy z jednostką oraz dostosowujące ją do założeń realizowanej pracy magisterskiej.

4.1. Modyfikacje

W trakcie prowadzenia badań dokonywano modyfikacji konstrukcji robota opracowanej w ramach pracy przejściowej [9] zgodnie z przyjętymi założeniami projektu. Kolejno prowadzone przejazdy testowe na torach testowych oraz implementacje nowych algorytmów sterowania uwidaczniały elementy wymagające przebudowy. Po wyciągnięciu wniosków z poprzedniej wersji konstrukcji używano ich do ulepszenia kolejnej. Końcowa wersja projektu jest efektem wykonania szeregu iteracyjnych modyfikacji prototypu do wersji ostatecznej, która została przedstawiona na rys 4.1.



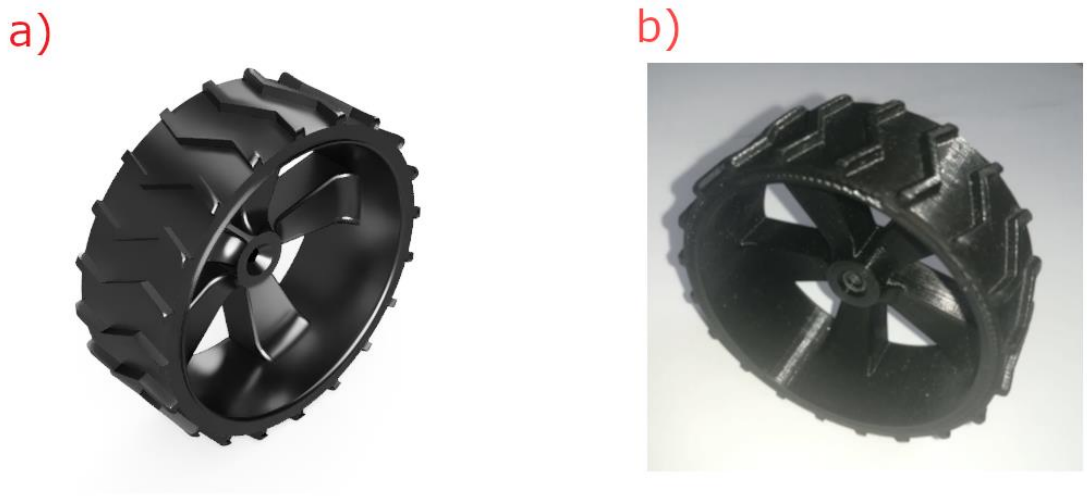
Rys. 4.1. Widok kolejnych prototypów platformy robotycznej od lewej - a) praca przejściowa b) praca magisterska

Modularna konstrukcja platformy robotycznej oraz zastosowanie druku 3D umożliwiły szybkie prototypowanie poszczególnych elementów oraz ich prostą implementację do konstrukcji. W celu zaimplementowania zmodyfikowanego elementu nie był konieczny demontaż całej konstrukcji tylko poszczególnych podzespołów. Modyfikacje w jednej dziedzinie nie wprowadzały potrzeby ponownego wytwarzania pozostałych elementów. Ułatwiło to proces szybkiego prototypowania ponieważ każdy z zaprojektowanych podzespołów spełnia określoną funkcję, odmienną od innych. Dodatkowo w przypadku łączenia części zastosowano zunifikowaną metodą łączeniową, która pozwoliła na dostosowanie elementów do przyjętej postaci.

4.2. Koła

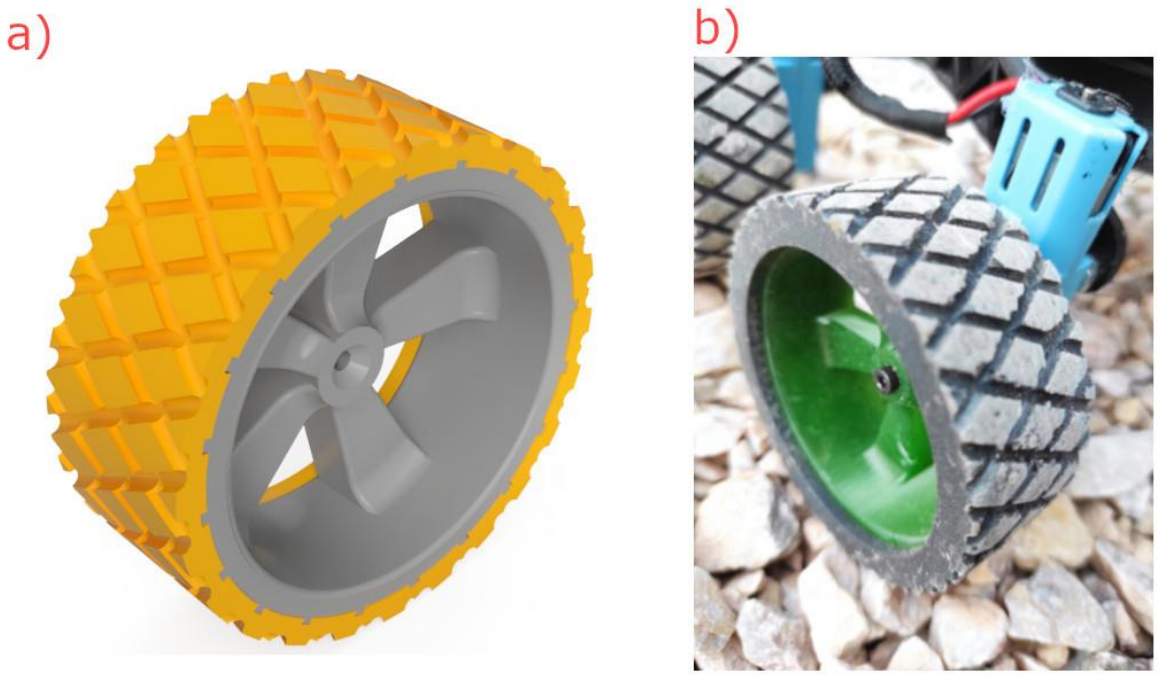
W trakcie przeprowadzania badań na torach testowych uwidoczniły się wady stosowania przyjętych w pracy przejściowej [9] kół platformy robotycznej. Koła przedstawione na rys 4.2. wzorowane były na konstrukcji zastosowanej w marsjańskim łaziku eksploracyjnym Curiosity [54]. Posiadają one wzór w kształcie odwróconej litery V tzw. chevron pattern. W projekcie przyjęto wyposażenie kół w 24 rowki. Koła zostały wydrukowane w całości co wyeliminowano punkty naprężeń, w których części łączyłyby się ze sobą w postaci śrub lub punktów zatrzasku. Wzór ten przyjęto kierując się z informacjami zamieszczonymi na stronie amerykańskiej Narodowej Agencji Aeronatuki i Przestrzeni Kosmicznej – NASA (ang. National Aeronautics and Space Administration) informującymi, że wzór chevron pattern zapewnia odpowiednią stabilność kół podczas pokonywania wzniesień [54]. Dodatkowo ostre krawędzie mogą być wykorzystane do pomocy łazikom podczas pokonywania przeszkód lub wzniesień. Koła wykonano w technologii FDM (ang. Fused Deposition Modeling) na urządzeniu Prusa i3 MK3S+. Wykorzystane tworzywo to filament PLA.

W trakcie przeprowadzania badań na torze testowym w warunkach wewnętrznych charakterystyczna stała się głośna praca kół oraz mały współczynnik tarcia na utwardzonych płaskich powierzchniach. Powodowało to obracanie się kół w miejscu oraz problemy przy wykonywaniu skrętów.



Rys. 4.2. Widok wykonanego projektu koła w ramach pracy przejściowej od lewej - a) model b) wydrukowany element

Na torze testowym w warunkach zewnętrznych właściwości jezdne były znacząco lepsze jednak stwierdzono, że wykonane na całym obwodzie rowki charakteryzują się za małą wysokością, przez co utrudniały pokonywanie kamieni o różnej frakcji. Z tego powodu zdecydowano się na zamodelowanie nowych kół (rys 4.3.) z podziałem na felgę oraz oponę. Decyzję tą podjęto z względu na większą elastyczność. W tej konfiguracji istnieje możliwość zmiany kształtu bieżnika opon dostosowując ją do występujących warunków. Ponadto, w technologii druku 3D zdecydowano się wykorzystać materiały o właściwościach zbliżonych do gumowych (oparty na żywicy fotopolimerowej), przez co możliwe stało się uzyskanie odpowiedniej przyczepności. Kształt felgi uzyskano poprzez modyfikacje ww. kół dostosowując kształt rowków do wpustów jakie zamodelowano na oponie w celu uzyskania odpowiedniego spasowania i docisku w trakcie pracy. Opony wykonano w technologii druku żywicznego SLA na urządzeniu Formlabs Form3 [38]. Tworzywo wykorzystane do opon to żywica FormLabs Flexible [36]. Do wydruku felg zastosowano tą samą technologię, natomiast wykorzystano drukarkę Elgoo Saturn [84] oraz żywicę Elgoo ABS-like [37] o parametrach zbliżonych do tworzywa ABS charakteryzujących się wysoką wytrzymałością, sztywnością i odpornością na uderzenia. Wydruki elementów w technologii żywicznej zostały wykonane przez mgra inż. Andzeja Jałowieckiego przy wykorzystaniu sprzętu udostępnionego przez Katedrę Podstaw Konstrukcji Maszyn Wydziału Mechanicznego Technologicznego Politechniki Śląskiej.



Rys. 4.3. Widok zmodyfikowanego koła z podziałem na felgę i oponę od lewej - a) model b) wydrukowany element

Po zastosowaniu tej modyfikacji uzyskano bardzo dobrą kulturę pracy w warunkach wewnętrznych na płaskich powierzchniach. Z względu na twardość żywicy z której wykonana jest opona dobrze sprawdza się ona na skałach osadowych różnej frakcji na torze w warunkach zewnętrznych. W trakcie prowadzenia badań wykonano także projekt opony (rys 4.4.) o odmiennym kształcie bieżnika. Celem było uzyskanie lepszej przyczepności przy poruszaniu się w zagłębieniu w postaci krateru na torze testowym w warunkach zewnętrznych. Wyciągnięto wnioski z obserwacji poprzednich wersji opony i zastosowano kształt bieżnika lepiej sprawdzający się w przypadku skał osadowych większej frakcji. Modyfikacja ta wprowadziła wadę, jednak w trakcie wykonywania prac prototypowych autor był świadomy tego faktu. Mianowicie opona gorzej sprawdza się w warunkach wewnętrznych, generując większy hałas podczas pracy. Z tego powodu wykorzystywane są obecnie dwa typy opon w zależności od występującej potrzeby.

Rys. 4.4. Model opony z bieżnikiem przystosowanym do skał osadowych większej frakcji

4.3. Zasilanie

Wybrany w pracy przejściowej [9] zestaw programistyczny Jetson Nano będący jednostką obliczeniową projektu wykorzystuje podejście DVFS (ang. Dynamic Voltage and Frequency Scaling). Ta technologia zarządzania energią jest wykorzystywana w większości współczesnego sprzętu komputerowego w celu maksymalizacji oszczędności energii. Moduł Jetson Nano zasilany jest z pojedynczego źródła zasilania a wszystkie wewnętrzne napięcia modułów oraz wejść/wyjść są generowane z ww. wejścia. Umożliwia to pokładowemu kontrolerowi zarządzanie energią wdrażając wielowarstwowe struktury zasilania oraz odpowiednie taktowanie optymalizując w ten sposób zużycie energii w zależności od obciążenia.

Do zasilania Jetsona Nano wykorzystywany jest oddzielny powerbank INIU BI-B41 10000mAh [14] charakteryzujący się dwoma wyjściami typu USB-A o parametrach napięcia 5 [V] oraz natężenia prądu 3[A] na każdym wyjściu. W zależności od wydajności prądowej źródła zasilania Jetson Nano obsługuje dwa definiowane programowo tryby zasilania. Domyślny tryb zapewnia zapotrzebowanie energetyczne w wysokości 10 [W] i aktywowany jest komendą:

```
$ sudo nvpmodel -m 0
```

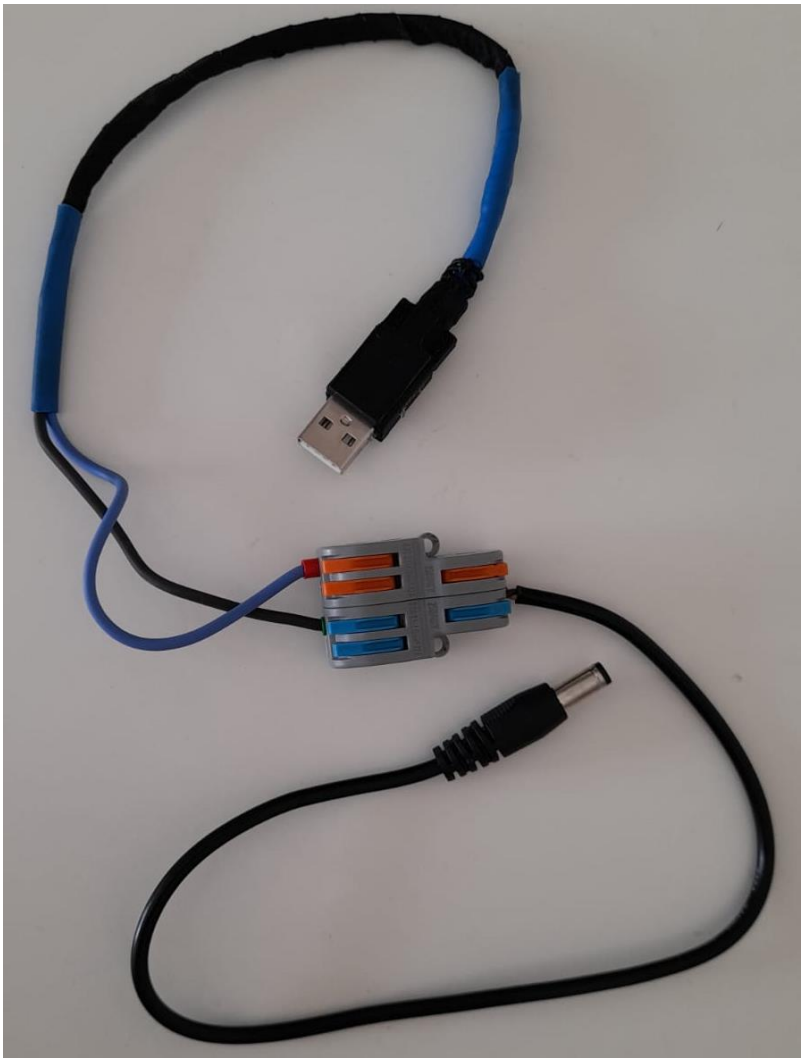
Natomiast drugim występującym trybem zasilania jest tryb o mocy 5 [W] włączany za pomocą komendy:

```
$ sudo nvpmodel -m 1
```

Różnica ta wynika z ograniczenia częstotliwości jednostki graficznej - GPU i procesora - CPU oraz liczby rdzeni CPU. Tryb 5 [W] powoduje ograniczenie pracy jednostki Jetson Nano do dwóch rdzeni procesora. Z tego powodu w realizacji pracy wykorzystywano tryb 10 [W]. Jetson Nano Developer Kit jest skonfigurowany do przyjmowania zasilania przez złącze Micro USB. W złączu tym występuje maksymalne ograniczenie natężenia prądu w wysokości 2 [A]. Po przekroczeniu tej wartości następuje automatyczne zakończenie działania pracy Jetsona Nano. W przypadku dużej liczby zastosowanych urządzeń peryferyjnych dołączonych do płyty nośnej wykorzystywanych w projekcie tj. kamera Raspberry Pi HD v2 8MPx, dwa sterowniki silników układ PCA9685 + TB6612, wyświetlacz PiOLED, karta sieciowa Intel Dual Band Wireless-AC oraz wentylator w momencie uruchamiania kamery oraz algorytmu sterowania czyli wysoko chłonnych zadań obliczeniowych następowało automatyczne wyłączenie zestawu programistycznego. W przypadku zastosowania trybu 5 [W] do ww. zadań występowała bardzo powolna praca platformy robotycznej. Z tego powodu zdecydowano się zasilać Jetsona Nano poprzez złącze baryłkowe (ang. barrel jack) posiadające te same parametry napięcia 5 [V] lecz dwukrotnie wyższe natężenie prądu wynoszące 4 [A]. W tym celu konieczne stało się założenie zworki na gniazdo J48 [13]. Wyżej wymienione problemy w trybie 5 [W] wykorzystywanym początkowo przy zasilaniu poprzez złącze barrel jack stały się również widoczne w przypadku pracy z kamerą. W momencie gdy następowało przesłanie obrazu z kamery z jednoczesnym włączeniem wyuczonej już sieci, wystąpiły opóźnienia w wyświetlaniu obrazu oraz przerwy. Skutkowało to gubieniem trasy przez robota oraz kolizjami z przeszkodami. Natomiast w przypadku zastosowania trybu 10 [W] dla ww. czynności następowało wyłączenie pracy zestawu programistyczny Jetson Nano w momencie uruchamiania wyuczonego modelu. W trakcie realizacji pracy stało się to znaczącym problemem koniecznym do rozwiązania, aby możliwe stało się przeprowadzenie dalszych badań. W tym celu wykonywano szereg testów przy użycia stabilizowanego zasilacza laboratoryjnego Korad KD3005P 0-30V 5A [12]. Uzyskano zgodnie z informacjami producenta płynną regulację napięcia w zakresie od 0 do 30 [V] oraz prądu w zakresie od 0 do 5 [A]. W wyniku analizy literatury znaleziono informację w dokumentacji firmy NVIDIA [13], że moduł NVIDIA Jetson Nano wymaga do działania

napięcia minimum 4,75 [V]. Z tego powodu natężenie prądu przyjęto o wartości 2.5 [A] natomiast napięcie ustalono na 5.10 [V] będące większe od nominalnego ponieważ uwzględniano straty napięcia w przewodzie zasilającym. Jednak problem wyłączania się jednostki Jetson Nano pozostał. W trakcie przeprowadzonych badań z zwiększanymi stopniowo parametrami prądowymi stwierdzono, że w momencie rozpoczęcia działania wyuczonego modelu następuje znaczy spadek napięcia zmierzony na ścieżkach zasilających płytkę z złącza baryłkowego. Parametry napięcia jaki przed wyłączeniem zarejestrował miernik Fluke 87V [15] wynosiły w granicach 4.83 [V] – 4.91 [V]. Pomiary te wskazały, że problem występuje w przewodach zasilających. Z tego powodu zdecydowano się przylutować przewody silikonowe o przekroju 0.5 [mm^2] i długości 500 [mm] bezpośrednio do złącza baryłkowego na płytce Jetson Nano. Przekrój ten wybrano, gdyż kierując się dokumentacją producenta deklarowana obciążalność prądową przewodu wynosi do 7 [A]. Po wykonaniu tych czynności zestaw programistyczny Jetson Nano zaczął działać poprawnie i ww. problemy zostały wykluczone. Przeprowadzono badania zmieniając parametry prądowe na zasilaczu stopniowo je zmniejszając, aż do momentu ponownego wyłącznie się Jetsona Nano. W trakcie przeprowadzonych badań stwierdzono, że w trybie zasilania 10 [W] przy zastosowanych urządzeniach peryferyjnych i wykonywanych zadaniach obliczeniowych czyli po uruchomieniu algorytmu minimalne wartości prądowe jakie trzeba dostarczyć do Jetson Nano przy wykorzystywanych przewodach i zasilaczu to napięcie 5.09 [V] oraz natężenie 2.48 [A]. Po uzyskaniu wymaganej wiedzy dokonano pomiarów wartości wyjściowych parametrów zastosowanego powerbanku. Wartość napięcia jakie zmierzono to 5.14 [V]. Jest to bardzo korzystna wartość z względu na straty napięcia występujące w przewodzie. Natomiast maksymalne natężenie prądu jakie wystawia powerbank zgodnie z dokumentacją producenta wynosi 3 [A] na każdym wyjściu co zgodnie z przeprowadzonymi badaniami jest wartością wystarczającą. Zważając na ww. informacje zdecydowano się wykorzystać przewód zasilający Akyga AK-DC-01 [16] o deklarowanym napięciu 30 [V] , natężeniu 4 [A] oraz długości 800 [mm]. Jednak w połączeniu z powerbankiem i Jetsonem Nano ponownie pojawiły się ww. problemy. Podobnie jak w przypadku pierwotnie używanych przewodów następowały znaczne spadki napięcia uniemożliwiające prawidłową pracę Jetsona Nano. Z tego powodu postanowiono na samodzielnie wytworzenie złącza typu USB-A wraz z przylutowaniem przewodów silikonowych o przekroju 0.5 [mm^2] w połączeniu z końcówką żeńską złącza baryłkowego (rys 4.5). Zdecydowano się na taką układ, gdyż przez zastosowaną szybko złączkę można dowolnie wykorzystać przewód do

różnego typu końcówek zasilających. W takiej konfiguracji następuje poprawna praca platformy robotycznej zasilanej przy pomocy powerbanku INIU BI-B41 10000mAh [14] w trybie pracy 10 [W].



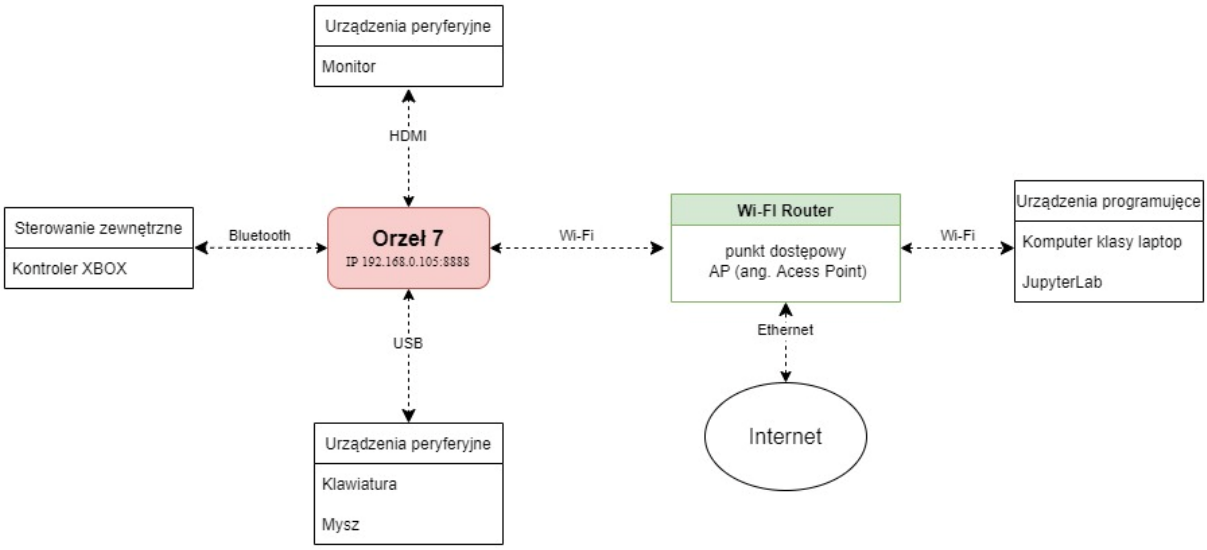
Rys. 4.5. Widok wykonanego przewodu zasilającego

4.4. Hotspot

Fabryczny zestaw Jetson Nano Developer Kit pozbawiony jest karty sieciowej umożliwiającej wymianę danych poprzez sieć Wi-Fi oraz bezprzewodowej komunikacji Bluetooth. Standardowo za łączność odpowiada gigabitowe złącze Ethernet do transmisji przewodowej znajdujące się na płytce rozwojowej. Zgodnie z informacjami zawartymi w pracy przejściowej, w projekcie dodano kartę sieciową Intel Dual Band Wireless-AC 8265

wraz z dwiema antenami. Pozwoliło to na korzystanie z sieci Wi-Fi w częstotliwości zarówno 2.4 [GHz] oraz 5 [GHz] w standardzie 802.11ac, a także ze standardu Bluetooth w wersji 4.2. Maksymalna prędkości transmisji danych wynosi 867 [Mbps].

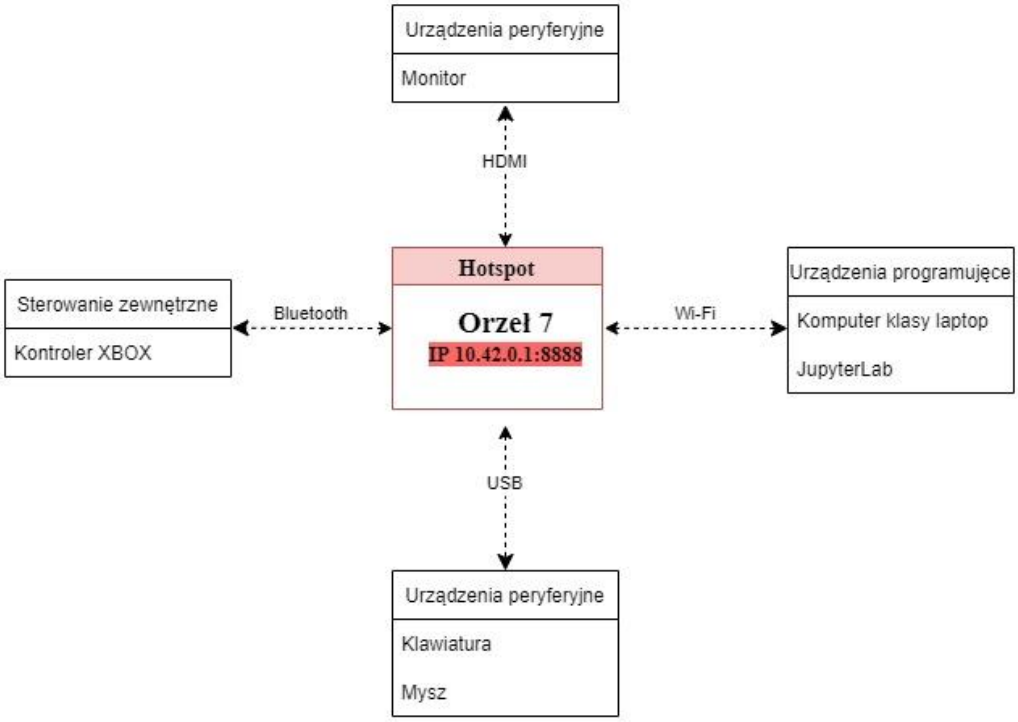
W pracy przejściowej [9] komunikacja z platformą robotyczną opierała się na połączeniu za pomocą bezprzewodowego standardu sieci komputerowej Wi-Fi wykorzystując standardową bezprzewodową sieć lokalną – WLAN (ang. Wireless Local Area Network). Platforma robotyczna łączyła się do punktu dostępowego - AP (ang. Acess Point) ze zintegrowanym routerem tak samo jak urządzenie programujące klasy laptop, co zostało przedstawione na rys.4.6. W związku z włączonym protokołem DHCP (ang. Dynamic Host Configuration Protocol) automatycznie przydzielającym adresy IP – adres platformy robotycznej zmieniał się początkowo z przedziału od 192.168.0.102 do 192.168.0.108. Jednak został on skonfigurowany na statyczny adres IP o numerze 192.168.0.105 w celu uniknięcia konfliktu adresów w sieci WLAN oraz uzyskania bardziej stabilnego połączenia.



Rys. 4.6. Schemat pierwotnego połączenia sieciowego

Rozwiązanie to sprawdziło się w kwestii pracy na torze testowym w warunkach wewnętrznych, gdzie występował dobry zasięg spowodowany bliskością punktu AP. Jednak przeznaczenie platformy robotycznej to zadania eksploracyjne.

Zważając na ten fakt zdecydowano się skonfigurować ww. kartę sieciową używaną w Jetsonie Nano w funkcji Hotspotu. W takim rozwiązaniu platforma robotyczna występuje w formie punktu dostępu i to do niej wykonywane są połączenia poprzez sieć Wi-Fi z urządzenia programującego klasy laptop bądź smartfonu (rys 4.7.). Modyfikacja ta znacząco wpłynęła na mobilność platformy np. w przypadku prowadzenia badań na torze testowym w warunkach zewnętrznych, gdyż do prawidłowej pracy wymagana jest tylko obecność platformy robotycznej oraz urządzenia programującego. Potencjalną wadą tego rozwiązana jest natomiast zwiększone zużycie energii w trybie Hotspot. W trakcie badań przeprowadzono testy w kontekście zasięgu zastosowanego rozwiązania. W przypadku otwartej przestrzeni na osiedlu mieszkaniowym uzyskano możliwość sterowania robotem platformą robotyczna z odległości 93 metrów. W warunkach pracy wewnętrznej poprawna praca robota eksploracyjnego odbywała się w granicy różnicy jednego piętra domu jednorodzinnego, jednak występowały czasowe opóźnienia przesyłania obrazu oraz wykonywania poleceń sterowniczych. W trakcie pracy z platformą robotyczną używano trybu pozbawionego interfejsu graficznego użytkownika - GUI (ang. Graphical User Interface) w celu zmniejszenia zużycia pamięci a także odciążenia procesora graficznego. Do komunikacji z zestawem programistycznym Jetson Nano wykorzystuje się protokół komunikacyjny SSH (ang. Secure Shell), który jest domyślnie włączony. W tym celu używane jest oprogramowanie PuTTY [17] zapewniającą funkcję programowego emulatora terminalu. Umożliwia to zdalną sesję na komputerze po wpisaniu odpowiedniego adresu IP urządzenia z którym chcemy się łączyć. W przypadku projektu magisterskiego była to platforma robotyczna o adresie IP 10.42.0.1 wraz z numerem portu 8888.



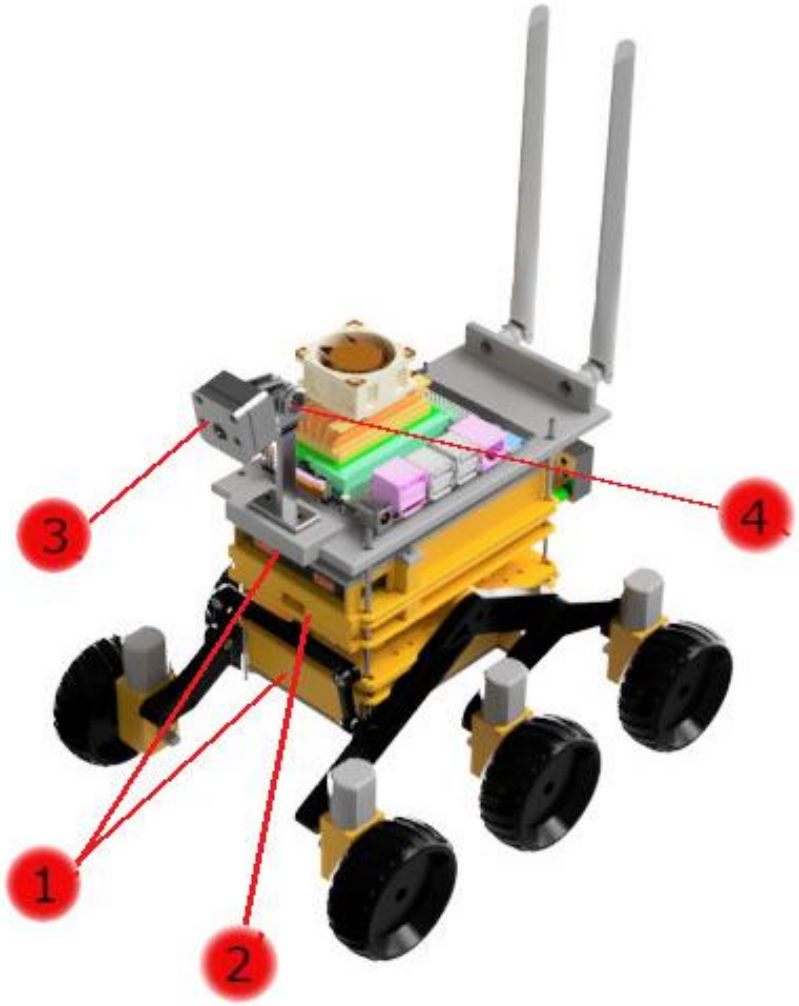
Rys. 4.7. Schemat połączenia sieciowego z rolą robota eksploracyjnego jako hotspotu

4.5. Modernizacja konstrukcji

Projekt konstrukcji robota wykonany został przy użyciu oprogramowania Autodesk Fusion 360 [19]. Jest to środowisko obejmujące moduły m.in.: projektowania CAD oraz renderowania 3D. Technikę wykonania robota, jaką przyjęto w założeniach, oparto na druku przestrzennym 3D w technologii FDM (ang. Fused Deposition Modeling) kierując się szybkością prototypowania oraz wykonywania podzespołów, a także wiedzą autora w tym zakresie. Zaprojektowane elementy w środowisku Autodesk Fusion 360 zostały wyeksportowane do formatu STL (ang. stereolithography), a następnie zaimportowane do programu typu slicer. W niniejszym projekcie wykorzystywany jest PrusaSlicer [20]. Pliki STL zostały umieszczone w dodatku B załączonym do niniejszej pracy.

W trakcie prowadzonych badań zdecydowano się nie wprowadzać zmian w wykonanej w ramach pracy przejściowej [9] konstrukcji zawieszenia typu rocker boogie. W wyniku testów przeprowadzanych na torze testowym w warunkach zewnętrznych uzyskano bardzo dobre własności terenowe w pokonywaniu założonych przeszkód, a także zadowalającą stabilność konstrukcji. Natomiast zdecydowano się przebudować centralną część platformy robotycznej w kontekście wykorzystania dwóch źródeł zasilania w postaci

powerbanków. Zważając na opisane w poprzednich podrozdziałach problemy w kwestii poprawnego zasilania jednostki Jetson Nano postawiono dedykować oddzielny powerbank do zasilania zestawu Jetson Nano natomiast drugi powerbank wykorzystany jest do zasilenia dwóch sterowników silników. Takie rozwiązanie jest korzystne w kwestii pracy Jetsona Nano przy wykonywaniu wysoko chłonnych zadań obliczeniowych, gdyż dedykowany powerbank pracuje w zakresie maksymalnych wartości deklarowanych parametrów. Analogiczna sytuacja występuje w kontekście wykorzystania drugiego powerbanku służącego do zasilania silników oraz sterowników w przypadku maksymalnego obciążenia przy pokonywaniu przeszkód. Rozwiązanie to wpłynęło pozytywnie na stabilność pracy platformy robotycznej przyczyniając się do eliminacji wymienionych w podrozdziale 4.3 problemów. W tym celu zamodelowano i wykonano dwie obudowy powerbanków oznaczone na poniższym modelu cyfrą 1. Dodatkowo wykonano kieszeń na przewody (rys 4.8., poz 2) zasilające jednostkę obliczeniową Jetson Nano w celu uporządkowania instalacji oraz możliwości dokonywania szybkich zmian zgodnie z prowadzonymi badaniami opisanymi w podrozdziale 4.3. W odniesieniu do pracy przejściowej zdecydowano się także wprowadzić zmianę w kontekście wykorzystywanej kamery. Zastąpiono kamerę IMX219-160 8MP IR-CUT dedykowaną dla zestawu Jetson kamerą Raspberry Pi Camera HD v2 8MPx (rys 4.8., poz 3). Czynność ta podyktowana została występującymi problemami z integracją kamery IMX219 pomimo instalacji sterowników oraz dedykowanych bibliotek. W trakcie prowadzonych badań stwierdzono, że kamera została uszkodzona przed wykorzystaniem jej przez autora projektu. Z racji występujących problemów z integracją zdecydowano się na wykorzystanie najbardziej popularnej i wspieranej ww. kamery.



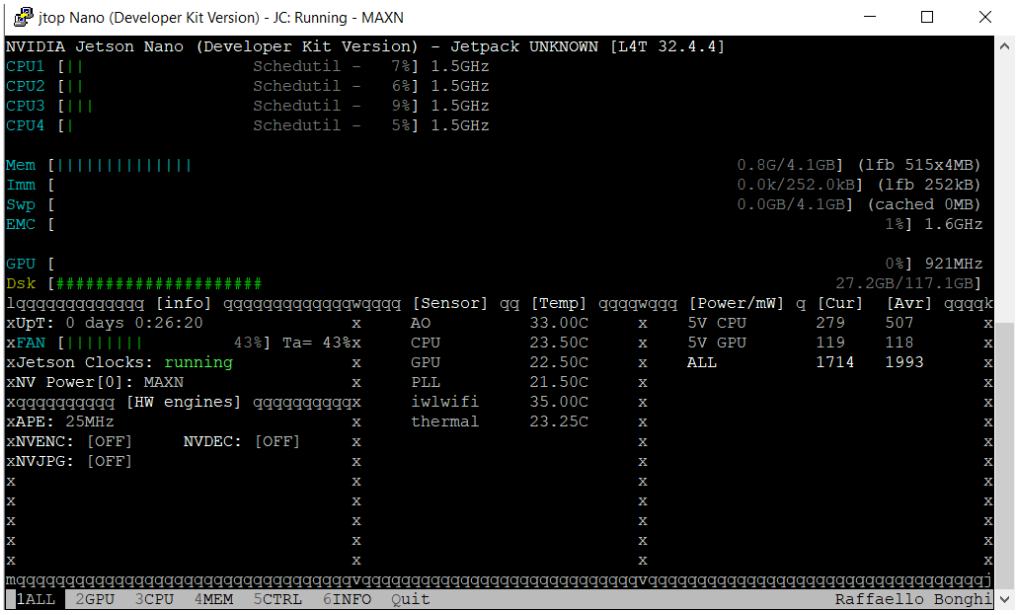
Rys. 4.8. Złożenie modelu robota eksploracyjnego

Zgodnie z założeniami zadania eksploracyjne robota powinny odbywać się także w warunkach nocnych. Poprzednio wykorzystana kamera wyposażona była w diody LED na podczerwień oraz wbudowany filtr podczerwieni przez co możliwa była jazda w warunkach nocnych opierając się na obrazie z kamery. Jednak w przypadku kamery Raspberry Pi Camera HD v2 8MPx czynność ta stała się niemożliwa z powodu występującego braku widoczności w warunkach nocnych. Z racji tego wykorzystano wbudowane w powerbank diody LED aktywowane za pomocą przycisków bistabilnych gwarantujących widoczność w warunkach nocnych. Warto zaznaczyć, że jest to rozwiązanie tymczasowe. Robot docelowo ma charakteryzować się całkowitą autonomią, więc planowane jest utworzenie możliwości samodzielnego włączenia oświetlenia np. poprzez zastosowanie tranzystora. Przeznaczeniem robota eksploracyjnego jest autonomiczna jazda w oparciu o dane obrazowe z kamery zamocowanej na odpowiednio

dobranej wysokości oraz pod odpowiednim kątem względem podłoża. Zważając na to wykonano mocowanie (rys. 4.8., poz. 4) pozwalające na swobodną konfigurację wysokości oraz nachylenia w zależności od potrzeb i warunków zewnętrznych.

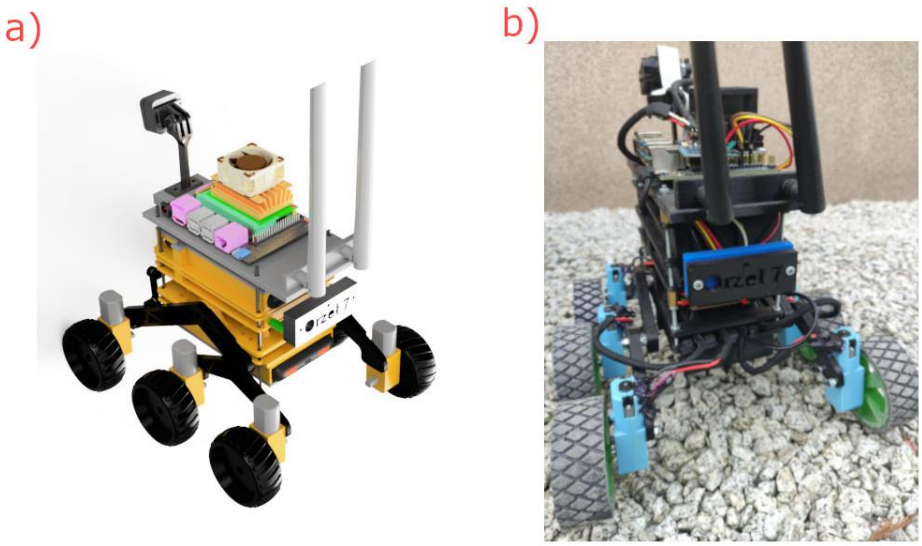
W trakcie prowadzonych badań często występującym problem było zerwanie przewodów w miejscu lutów bądź też przetarcie przewodu w związku z licznymi kolizjami na kamiennym terenie. Z tego powodu zdecydowano się rozplanować przeprowadzenie przewodów w oplocie z plecionki w celu osłony przed uderzeniami mechanicznymi. Złącza lutowane zostały natomiast wzmocnione przed ukręceniem poprzez otoczenie całego elementu klejem na gorąco. Spowodowało to usztywnienie złącz lutowanych szczególnie przy wyprowadzeniach z silników, które podatne były na złamania w związku z licznymi uderzeniami o przeszkody w trakcie prowadzenia badań.

NVIDIA Jetson Nano Developer Kit wyposażony jest w pasywny radiator jednak w momencie dokonywania wysoko chłonnych zadań obliczeniowych np. szkolenia modelu temperatura jednostki graficznej oraz procesora przekraczała 45 [°C]. Parametry temperaturowe pracy Jetsona Nano badano przy wykorzystania narzędzia jetson-stats [18]. Jest to narzędzie służące do monitorowania systemu, które działa w terminalu. Przy jego wykorzystaniu można zobaczyć oraz kontrolować w czasie rzeczywistym stan Jetson Nano, parametry pracy CPU, RAM oraz GPU. Przykładowy widok zakładki parametrów pracy jednostki Nvidia Jetson Nano został przedstawiony na rys 4.9. Z racji występującej wysokiej temperatury pracy układu zastosowano wentylator sterowany automatycznie sygnałem PWM w zależności od obciążenia pracy jednostki w celu maksymalizacji skuteczności chłodzenia wpływając na bardziej wydajną pracę jednostki.



Rys. 4.9. Widok zakładki parametrów pracy jednostki obliczeniowej platformy robota eksploracyjnego Orzeł 7 w narzędziu jetson-stats

W celu identyfikacji robota nadano mu w tylnej części emblemat z przyjętą skróconą nazwą robota w postaci – Orzeł 7 (rys 4.10). Nazwa ta wykorzystywana jest z kwestii praktycznych w materiałach promocyjnych, podczas referowania przeprowadzonych badań oraz w codziennej pracy z robotem wykluczając posługiwanie się nazwą platformy robotycznej do nauki i rozwoju autonomicznych systemów sterowania w kontekście robotów eksploracyjnych.



Rys. 4.10. Widok tylnej części robota eksploracyjnego z emblematem Orzeł 7 od lewej - a) model b) rzeczywiste wykonanie.

Rozdział 5

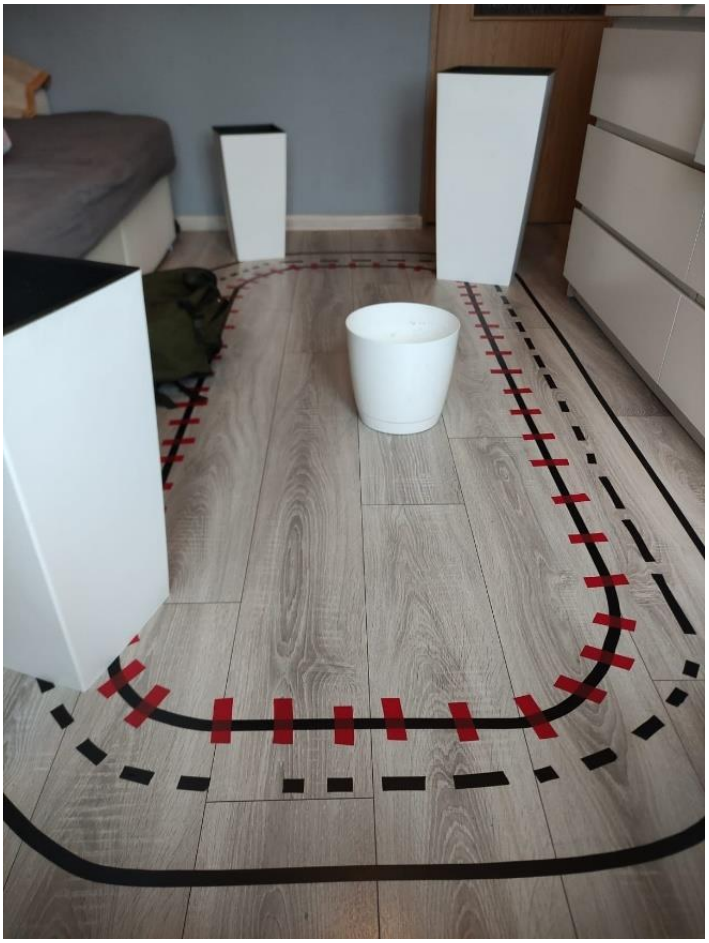
5. Środowiska testowe do zbioru danych uczących

W celu zrealizowania projektu wymagane jest utworzenie zbioru danych uczących służących do wytrenowania modeli umożliwiających odpowiednią klasyfikację zajętości drogi. Poniżej opisano wykonane dwa tory testowe w warunkach wewnętrznych oraz zewnętrznych dające możliwość utworzenia zbioru danych zgodnie z przyjętymi założeniami.

5.1. Tor testowy w warunkach wewnętrzny

W ramach pracy magisterskiej utworzono eksperymentalne środowiska testowe będące istotnym elementem prowadzonych badań w zakresie systemu sterowania z wykorzystaniem robota eksploracyjnego. Środowisko testowe w warunkach wewnętrznych ma kształt elipsy o wymiarach 3000 x 1500 [mm]. Składa się ona z ciągłych czarnych pasów inscenizujących pas drogowy z przerywanymi liniami pośrodku. W tym przypadku celem jest jedynie testowanie algorytmów autonomicznej jazdy z pominięciem specyficznych własności mechanicznych łazika. Umożliwia ono przeprowadzenie badań w świecie rzeczywistym oraz zdobycie doświadczenia w kontekście stawianych założeń. Taki rodzaj weryfikacji utworzonej konstrukcji pozwala przeprowadzić testy w sposób optymalny w kontekście praktycznej implementacji. Możliwe wtedy staje się zbieranie przykładów i utworzenie zbioru danych uczących będące najbardziej czasochłonną oraz problematyczną częścią każdego projektu wykorzystującego sieci neuronowe w tym, także realizowanej pracy magisterskiej. Działania te spowodowały dużą użyteczność w przypadku przeprowadzania pierwszych testów autonomicznej jazdy np. w przypadku implementacji algorytmu podążania drogą. W tym celu wykorzystano właśnie przygotowywaną inscenizację drogi. Stanowisko badawcze do jazdy w warunkach wewnętrznych zostało zaprojektowane tak aby umożliwić stopniowanie trudności w zadaniach percepcji, wnioskowania i kontroli. W tym celu dodano kontrastujące czerwone pręgi na wewnętrznym obrzeżu utworzonej drogi silnie ograniczając obszar pracy robota eksploracyjnego. Kolejna modyfikacja toru dotyczyła dodania elementów o kształcie wielościanów o różnych wymiarach (rys. 5.1.). Konfiguracja ta jest zasadniczą przyjętą wersją toru testowego w warunkach wewnętrznych do realizacji pracy magisterskiej.

Obejmuje ona zadanie klasyfikacji zajętości drogi oraz dokonanie operacji ominięcia przeszkody.



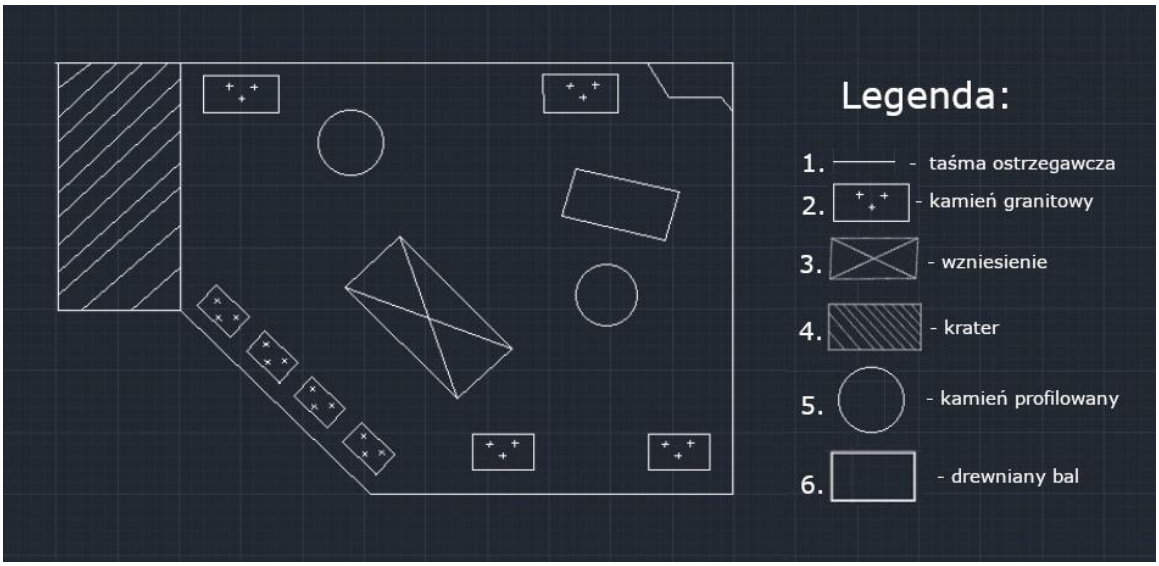
Rys. 5.1. Widok toru testowego w warunkach wewnętrznych

5.2. Tor testowy w warunkach zewnętrznych

W celu przetestowania właściwości zawieszenia oraz zdolności poruszania się utworzonego w ramach pracy przejściowej [9] robota eksploracyjnego po terenach nieutwardzonych symulujących warunki marsjańskie utworzono środowisko badawcze do jazdy w warunkach zewnętrznych. Utworzenie środowiska jest procesem czasochłonnym gdyż w fazie koncepcyjnej należy określić potrzeby oraz przewidzieć scenariusze występujące w przypadku prowadzenia badań. Tor testowy (rys. 5.2.) ma nieregularny kształt o granicznych wymiarach 3500 x 4000 [mm].

Rys. 5.2. Widok toru testowego w warunkach zewnętrznych

Inspiracją tworzenia toru był unikalny tor Marsian Yard [10] będący areną zmagań międzynarodowych zawodów Europen Rover Challenge (ERC) [10] odbywających się każdego roku w Kielcach na terenie Politechniki Świętokrzyskiej. Teren został odgrodzony taśmą ostrzegawczą (rys. 5.3, pkt.1) oraz wzdłuż niekontrastujących krawędzi toru zastosowano kamień murowy granitowy (rys. 5.3, pkt.2). Wykonano to w celu ułatwienia zbierania danych jasno określając granicę toru. Podłoże zostało pokryte skałami osadowymi o niejednolitej frakcji w celu przetestowania właściwości rożnych typów opon. Przygotowano wzniesienie (rys. 5.3, pkt.3) będące sprawdzianem dla zawieszenia typu rocker-bogie. Dodatkowo wykonano podłużne zagłębienie (rys. 5.3, pkt.4) symulując formy marsjańskie w postaci krateru. Właściwości jezdne zostały sprawdzone przy pokonywaniu odpowiednio profilowanych kamieni (rys. 5.3, pkt.5). Elementem odznaczającym się odmiennymi wymiarami oraz fakturą jest drewniany bal (rys. 5.3, pkt.6) dodany w celu przetestowania poprawności klasyfikacji przeszkody przez algorytm oraz przetestowania manewru ominięcia.



Rys. 5.3. Schemat toru testowego w warunkach zewnętrznych z wyszczególnionymi elementami składowymi

Rozdział 6

6. Projekt algorytmu sterowania

W tym rozdziale zostaną opisane założenia utworzonego systemu. W dalszej części przedstawiono także strukturę systemu sterowania robotem eksploracyjnym oraz wprowadzone niezbędne modyfikacje umożliwiające poprawną pracę platformy robotycznej.

6.1. Założenia systemu

W ramach pracy przejściowej [9] zaprojektowano prototyp platformy robotycznej, dokonano przeglądu literatury oraz oprogramowania, a także zapoznano się z metodyką tworzenia systemów wizyjnych w sterowaniu robotów bazujących na technikach uczenia głębokiego. Na podstawie zdobytych informacji zdefiniowano założenia systemu:

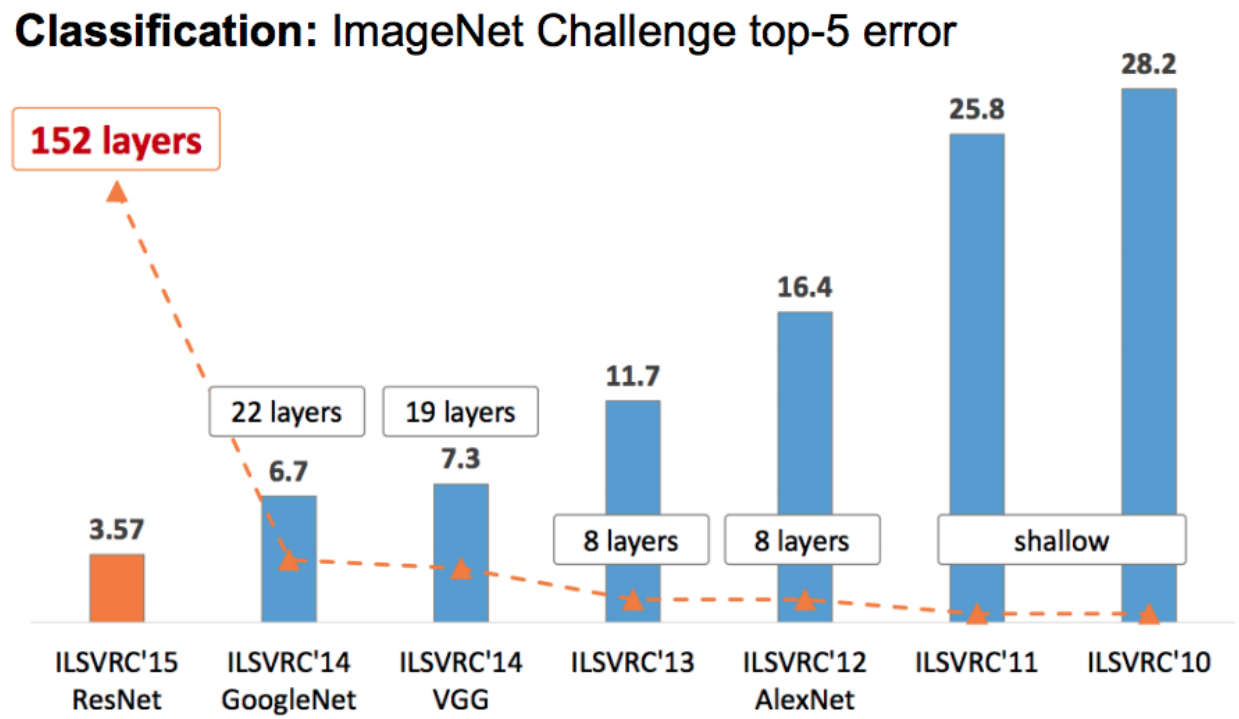
- wykorzystanie zróżnicowanych architektur w klasyfikacji obrazów (AlexNet o 8 warstwach ukrytych oraz ResNet - głęboka sieć szczątkowa o 152 warstwach ukrytych),
- użycie wstępnie nauczonej sieci neuronowej (ang. transfer learning) z pakietu Torchvision biblioteki PyTorch [29]
- nauka sieci z wykorzystaniem przygotowanych danych pochodzących z torów testowych,
- proces uczenia wykonywany na komputerze klasy laptop (CPU Intel Core i7-11800H, GPU RTX 3060, 64 GB RAM, 1512 GB SSD, system Windows 10),
- zbiór uczący składać się będzie z zdjęć o wymiarach 224 x 224 px poprzez zmniejszanie programowo rozmiaru wykonywanych zdjęć,
- praca systemu będzie się odbywać w czasie rzeczywistym,
- sterowanie robotem powinno być możliwe za pomocą dwóch niezależnych źródeł
- uczenie modelu będzie wykorzystywać rdzenie CUDA (ang. Compute Unified Device Architecture) [44] w celu poprawy wydajności obliczeń.

Przeprowadzano badania w celu wykorzystania do nauki modelu platformy Jetson Nano. Posiadane parametry techniczne umożliwiają przeprowadzenie trenowania w niedużej liczbie epok jednak jednostka pracuje na swoich maksymalnych parametrach

skutkujących wysoką temperaturą. Z tego powodu wykorzystano zewnętrzną jednostkę wyposażaną w GPU dedykowany procesom obliczeniowym. Uzyskano przyspieszenie procesu uczenia oraz łatwość prowadzenia badań nad zmiennością parametrów. Rozmiar zdjęć został dostosowany do danych przyjmowanych przez sieć neuronową w celu minimalizacji rozmiaru pliku zbioru danych. Wykorzystano transfer learningu aby osiągnąć lepsze wyniki nauki sieci oraz skrócenie czasu szkolenia.

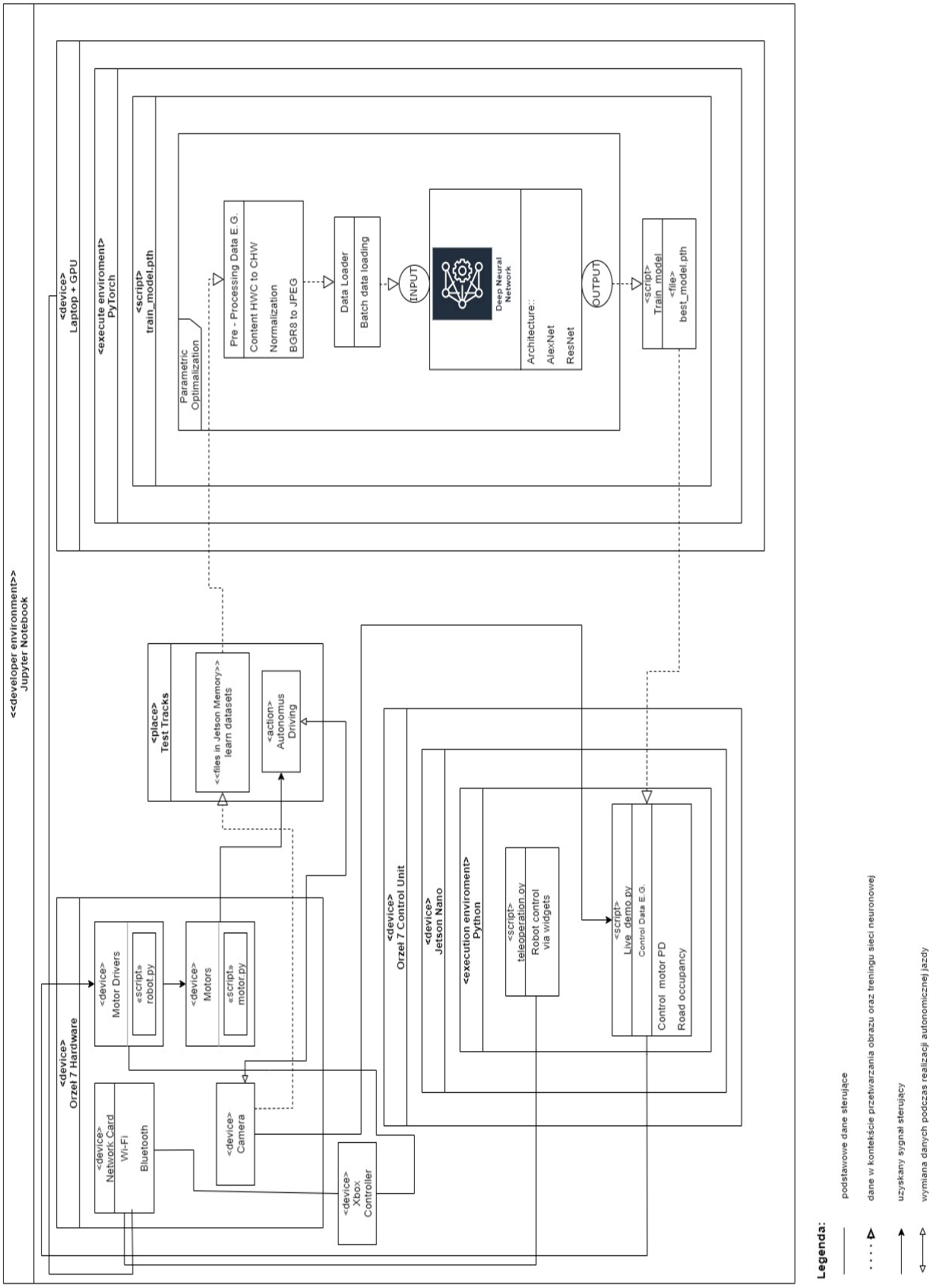
6.2. Struktura systemu sterowania

Zgodnie z założeniami oraz przeglądem literatury do implementacji systemu zdecydowano się wykorzystać język Python (wersja 3.9) [65]. Część pracy związaną z algorytmami bazującymi na głębokich sieciach neuronowych została zaimplementowana przy pomocy biblioteki stosowanej w aplikacjach wizji komputerowej – PyTorch (wersja 1.12.0), która zoptymalizowana jest pod kątem obliczeń tensorowych z akceleracją GPU. W wyniku przeglądu literatury oraz zgodnie z dokonanymi założeniami zdecydowano się wykorzystać w zadaniach klasyfikacji obrazów dwie architektury zróżnicowane pod względem budowy AlexNet oraz ResNet, które zostały bardziej szczegółowo opisane w dalszej części pracy. Na wykresie (rys. 6.1.) przedstawiona została różnica pomiędzy liczbą zastosowanych warstw w zwycięskich architektach sieci biorących udział w danym roku w prestiżowy konkursie rozpoznawania obrazu ImagNet ILSVRC (ang. ImageNet Large Scale Vision Recognition Challenge)



Rys. 6.1. Wykres głębokości oraz poziom błędu klasyfikacji obrazu w konkursie ILSVRC w latach 2010 – 2015 [21].

Projekt systemu sterowania robotem eksploracyjnym bazujący na głębokich sieciach neuronowych przygotowano w oparciu o podaną poniżej specyfikację. Diagram przedstawiony poniżej został utworzony przy użyciu oprogramowania Diagrams.net [39]. Na rys. 6.2. przedstawiono architekturę utworzonego systemu sterowania robotem eksploracyjnym bazującego na głębokich sieciach neuronowych. Można wyróżnić dwa urządzenia składowe tj. komputer klasy laptop wyposażony w jednostkę graficzną dedykowaną do zadań obliczeniowych oraz robota eksploracyjnego – Orzeł 7. Wszystkie czynności służące do implementacji systemu sterowania oraz komunikacji z robotem eksploracyjnym wykonywane są za pomocą internetowej interaktywnej platformy komputerowej JupyterLab [66] służącej jako zintegrowane środowisko programistyczne IDE (ang, Integrated Development Environment) oraz repozytorium z dokumentacją przykładów projektu JetBot AI [67].



Rys. 6.2. Struktura systemu sterowania robotem eksploracyjnym

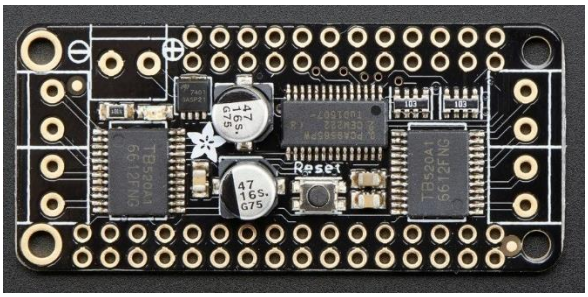
Orzeł 7 rozdzielony jest na część sprzętową oraz jednostkę sterującą czyli układ Jetson Nano. Część sprzętowa obejmuje m.in. sterowniki silników oraz silniki ze zmodyfikowanymi skryptami *robot.py* oraz *motor.py*, a także kamerę będącą kluczową częścią systemu do generowania danych oraz stanowiącą jedyny sensor, za pomocą którego robot zbiera informację z otoczenia. Karta sieciowa służy do komunikacji z robotem przy wykorzystaniu urządzeń programujących oraz sterowania poprzez zewnętrzny kontroler poprzez zastosowanie technologii Bluetooth. Niezbędnym elementem w systemie opartym na przetwarzaniu obrazu jest zbiór danych, w skład którego wchodzą zdjęcia wolnej oraz zablokowanej drogi przejazdu na wybranym torze testowym. Z tego powodu na diagramie zaznaczono tory testowe jako składowa systemu sterowania robotem eksploracyjnym. Następnie przy użyciu laptopa z jednostką graficzną oraz poprzez wykorzystanie biblioteki PyTorch w skrypcie języka Python [załącznik A] dane są przygotowywane na wejście sieci neuronowej tzw. *preprocecssing danych* poprzez normalizację, zmianę rozmiaru oraz formatu. W kolejnym kroku przygotowane dane są dzielone na próbki oraz wgrywane do wstępnie przetrenowanych wybranych architektur sieci neuronowych tj. AlexNet oraz ResNet. Proces ten podlega optymalizacji poprzez dostrajanie parametrów, co zostało opisane w dalszej części pracy. Na wyjściu sieci otrzymuje się przetrenowany model sieci neuronowej. Ostatni element to wgranie modelu z komputera klasy laptop na układ Jetson Nano.

Wyposażony jest on w system operacyjny Linux Ubuntu 20.4 [68] przez co konieczne jest zapewnienie języka Python w wersji 3.9. Obraz otrzymywany jest w czasie rzeczywistym przy wykorzystaniu kamery Raspberry Pi Camera HD V2 8 Mpx, przez co Orzeł 7 może reagować przy pomocy wytrenowanego modelu na zmieniające się warunki toru testowego. Jetson może być kontrolowany przy zastosowaniu zewnętrznego kontrolera z poziomu komputera klasy laptop poprzez połączenie SSH lub zdalny pulpit, a także przy wykorzystaniu widżetów sterowniczych w narzędziu Jupyter Notebook. Alternatywne można połączyć się z układem Jetson poprzez użycie monitora HDMI oraz klawiatury i myszki.

6.3. Modyfikacja systemu sterującego silnikami

Projekt platformy robotycznej udostępniony przez firmę Nvidia o nazwie JetBot AI, na którym bazuje niniejsza praca magisterska pozwala sterować dwoma silnikami

prądu stałego za pośrednictwem magistrali I^2C oraz kontrolera DC Motor + Stepper FeatherWing (rys 6.3).



Rys. 6.3. Sterownik silników Adafruit DC Motor + Stepper FeatherWing [69]

Sterownik posiada możliwość integracji wielu urządzeń peryferyjnych z względu na rozbudowane 40-stykowe złącze rozszerzające wyposażone w interfejsy: I^2C, *UART*, *PWM, GPIO*. Jednak w kontekście sterowania silnikami posiada ograniczone możliwości pozwalając użyć dwóch dwubiegunowych silników krokowych lub 4 szczotkowych silników prądu stałego. Przyjęty w ramach pracy przejściowej typ zawieszenia rocker bogie zastosowany w sześciokołowym robocie eksploracyjnym z oddzielnym napędem na każde koło stosunkuje konieczność wykorzystania dwóch kontrolerów. W odróżnieniu od projektu JetBot AI zdecydowano się wykorzystać dwa sterowniki silników oparte o chipsety TB6612 oraz kontroler PCA9685 (rys. 6.4). Układ wyposażony jest w cztery mostki H. Chipset TB6612 zapewnia maksymalnie natężenie o wartości 1,2 [A] na mostek z zabezpieczeniem termicznym wyłączenia oraz wyposażony jest w wewnętrzne diody ochronne zabezpieczającymi przed zjawiskiem tzw. kickbacku.

Rys. 6.4. Sterownik silników oparty o chipsety TB6612 oraz kontroler PCA9685 [74]

Komunikacja z sterownikiem opiera się o magistralę I^2C z tego powodu konieczne stało się wykorzystanie drugiej magistrali, w którą wyposażony jest moduł Jetson Nano A02. W celu sterowania sześcioma silnikami konieczne stało się przebudowanie w katalogu *jetbot/jetbot* plików *motor.py* oraz *robot.py* służącymi do sterowania poszczególnymi częściami robota. Plik *motor.py* jest plikiem konfiguracyjnym sterowników silników robota. W celu wykorzystywania wszystkich 4 kanałów sterownika konieczna stała się ich definicja w klasie *Motor* w sposób jak pokazano poniżej.

```
    # configure state
    alpha = traitlets.Float(default_value=1.0).tag(config=True)
    beta = traitlets.Float(default_value=0.0).tag(config=True)

    def __init__(self, driver, channel, *args, **kwargs):
        super(Motor, self).__init__(*args, **kwargs)  # initializes
traitlets

        self._driver = driver
        #channel=4 #TB no right 3 left
        # definition of controller channels
        self._motor = self._driver.getMotor(channel)
        if (channel == 1):
            self._ina = 1
            self._inb = 0
        elif (channel == 2):
            self._ina = 2
            self._inb = 3
        elif (channel == 3):
            self._ina = 3
            self._inb = 0
        elif (channel == 4):
            self._ina = 4
            self._inb = 3
        atexit.register(self._release)
```

Listing 6.1. Przykładowa zawartość pliku *motor.py*

Plik *robot.py* zawiera natomiast funkcje do sterowania silnikami robota. Do pracy z silnikami prądu stałego za pomocą ww. sterowników wykorzystywano zaprojektowaną

przez firmę Adafruit bibliotekę dla języka Python - *Adafruit Motor Hat*. Wykorzystywany jest framework *Traitlets* pozwalający klasom Pythona posiadać atrybuty ze sprawdzaniem typu, dynamicznie obliczonymi wartościami domyślnymi i wywołaniami zwrotnymi przy zmianie wartości. Zmodyfikowano klasę *Robot* w której zdefiniowane są wszystkie silniki oraz dwie magistrale I^2C. Utworzony kod został przedstawiony poniżej wraz z komentarzami opisującymi przeznaczenie fragmentu kodu. Zdeklarowany znak wartości *speed* będący odpowiednio zdefiniowaną w trakcie działania programu liczbą dodatnia bądź ujemną został ustalony heurustycznie w trakcie prowadzenia testów w zależności od kierunku obrotów silnika.

```
import time
import traitlets
from traitlets.config.configurable import SingletonConfigurable
from Adafruit_MotorHAT import Adafruit_MotorHAT
from .motor import Motor
#TB_7

class Robot(SingletonConfigurable):

    left_motor = traitlets.Instance(Motor)
    left_motor2 = traitlets.Instance(Motor)
    left_motor3 = traitlets.Instance(Motor)
    right_motor = traitlets.Instance(Motor)
    right_motor2 = traitlets.Instance(Motor)
    right_motor3 = traitlets.Instance(Motor)


    # config 2 I2C_bus
    i2c_bus = traitlets.Integer(default_value=1).tag(config=True)
    i2c_bus2 = traitlets.Integer(default_value=0).tag(config=True)


    # definition right motors
    right_motor_channel =
traitlets.Integer(default_value=1).tag(config=True)
    right_motor_alpha = traitlets.Float(default_value=1.0).tag(config=True)
    right_motor_channel2 =
traitlets.Integer(default_value=2).tag(config=True)
    right_motor_alpha2 = traitlets.Float(default_value=-
1.0).tag(config=True)
    right_motor_channel3 =
traitlets.Integer(default_value=1).tag(config=True)
    right_motor_alpha3 = traitlets.Float(default_value=1.0).tag(config=True)
```

```
    #definition left motors
    left_motor_channel = traitlets.Integer(default_value=2).tag(config=True)
    left_motor_alpha = traitlets.Float(default_value=1.0).tag(config=True)

    left_motor_channel2 =
traitlets.Integer(default_value=4).tag(config=True)
    left_motor_alpha2 = traitlets.Float(default_value=-1.0).tag(config=True)
    left_motor_channel3 =
traitlets.Integer(default_value=3).tag(config=True)
    left_motor_alpha3 = traitlets.Float(default_value=1.0).tag(config=True)


    #initialization of motors with the help of controllers
    def __init__(self, *args, **kwargs):
        super(Robot, self).__init__(*args, **kwargs)
        self.motor_driver = Adafruit_MotorHAT(i2c_bus=self.i2c_bus)
        self.motor_driver2 = Adafruit_MotorHAT(i2c_bus=self.i2c_bus2)

        self.left_motor = Motor(self.motor_driver2,
channel=self.left_motor_channel, alpha=self.left_motor_alpha)
        self.left_motor2 = Motor(self.motor_driver,
channel=self.left_motor_channel2, alpha=self.left_motor_alpha2)
        self.left_motor3 = Motor(self.motor_driver2,
channel=self.left_motor_channel3, alpha=self.left_motor_alpha3)
        self.right_motor = Motor(self.motor_driver2,
channel=self.right_motor_channel, alpha=self.right_motor_alpha)
        self.right_motor2 = Motor(self.motor_driver,
channel=self.right_motor_channel2, alpha=self.right_motor_alpha2)
        self.right_motor3 = Motor(self.motor_driver,
channel=self.right_motor_channel3, alpha=self.right_motor_alpha3)


    def set_motors(self, left_speed, right_speed):
        #left motors parameters
        self.left_motor.value = left_speed
        self.left_motor2.value = left_speed
        self.left_motor3.value = left_speed

        self.all_left_motors = self.left_motor.value and
self.left_motor2.value and self.left_motor3.value


        #right motors parameters
        self.right_motor.value = right_speed
        self.right_motor2.value = right_speed
        self.right_motor3.value = right_speed
```

```
        self.all_right_motors = self.right_motor.value and
self.right_motor2.value and self.right_motor3.value
        self.all_right_motors = right_speed


        self.left_motor2.value = 0
        self.left_motor3.value = 0

        #right motors parameters
        self.right_motor.value = 0
        self.right_motor2.value = 0
        self.right_motor3.value = 0
```

Listing 6.2. Przykładowa zawartość pliku *robot.py*

Zdefiniowano funkcję *forward*, *backward*, *left*, *right* oraz *stop* służące do wykonywania ruchów robota w określonych kierunkach. Funkcje te wykorzystywane są w nawigowaniu robota po dokonaniu klasyfikacji zajętości drogi przez wyuczony model. Dodatkowo funkcje te umożliwiają sterowanie robotem zgodnie z założeniami za pomocą dwóch niezależnych źródeł (interaktywnych przycisków w przeglądarce internetowej oraz zewnętrznego kontrolera).

```
    # function for forward move robot
    def forward(self, speed=1.0, duration=None):

        #left motors speed value
        self.left_motor.value = speed
        self.left_motor2.value = speed
        self.left_motor3.value = -speed

        #right motors speed value
        self.right_motor.value = speed
        self.right_motor2.value = -speed
        self.right_motor3.value = speed

    # function for backward move robot
    def backward(self, speed=1.0):

        #left motors speed value
        self.left_motor.value = -speed
        self.left_motor2.value = -speed
        self.left_motor3.value = speed
```

```
        #right motors speed value
        self.right_motor.value = -speed
        self.right_motor2.value = speed
        self.right_motor3.value = -speed

    # function for left move robot
    def left(self, speed=1.0):

        #left motors speed value
        self.left_motor.value = -speed
        self.left_motor2.value = -speed
        self.left_motor3.value = speed

        #right motors speed value
        self.right_motor.value = speed
        self.right_motor2.value = -speed
        self.right_motor3.value = speed

    # function for right move robot
    def right(self, speed=1.0):

        #left motors speed value
        self.left_motor.value = speed
        self.left_motor2.value = speed
        self.left_motor3.value = -speed

        #right motors speed value
        self.right_motor.value = -speed
        self.right_motor2.value = speed
        self.right_motor3.value = -speed

    # function for stop move robot
    def stop(self):
        #left motors parameters
        self.left_motor.value = 0
        self.left_motor2.value = 0
        self.left_motor3.value = 0

        #right motors parameters
        self.right_motor.value = 0
        self.right_motor2.value = 0
        self.right_motor3.value = 0
```

Listing 6.3. Przykładowa definicja funkcji *forward, backward, left, right i stop*

Projektu JetBot AI wykonany jest w technologii Docker [70] przez co wszystkie katalogi to hermetyzowane środowiska, które nie umożliwiają prostej komunikacji z wejściowymi lub wyjściowymi urządzeniami peryferyjnymi. Z tego powodu wszystkie wprowadzane zmiany w plikach wymagały przebudowywania kontenera przy użyciu polecenia *sudo.*

6.4. Zastosowane technologie

W związku z gwałtownym rozwojem technologii tworzone są liczne narzędzia wykorzystywane m.in.: w projektach o zagadnieniach sztucznej inteligencji oparte o otwarty kod źródłowy. Najczęściej wykorzystywanymi narzędziami są języki wysokiego poziomu tj. Python lub R. Alternatywą dla nich są natomiast programy komercyjne m.in. LabVIEW lub MATLAB wymagające jednak posiadania licencji. W celu implementacji algorytmu klasyfikacji zajętości otoczenia zdecydowano się wykorzystać język Python. Decyzję tą podjęto m.in. z względu na znajomość języka przez twórcę pracy magisterskiej. Tworzony kod programu jest hostowanymi skryptami języka Python, a dostęp do plików oraz terminala realizowany jest poprzez internetową interaktywną platformę komputerową JupyterLab. Zaletą języka Python w obszarze sztucznej inteligencji jest dostępność wielu bibliotek wspomagających głębokie uczenie znacznie skracając czas rozwoju projektu np. PyTorch i TensorFlow, które wspierają także obliczenia na GPU. Dodatkowo występuje szerokie wsparcie na formach internetowych i blogach programistycznych ze względu na popularność tego języka w dziedzinie sztucznej inteligencji. W celu implementacji algorytmu klasyfikacji zajętości niezbędna jest konfiguracja środowisk oraz sprzętu. Wykorzystywane w ramach pracy magisterskiej narzędzia można podzielić na część sprzętową oraz oprogramowanie. Użyto otwarto źródłowe programy oraz biblioteki tj.:

- Anaconda (wersja 3) [71] z interpreterem Python 3.9,
- JupyterLab (wersja 3.4.5).

W części sprzętowej wykorzystano następujące urządzenia:

- komputer klasy laptop (CPU Intel Core i7-11800H, GPU RTX 3060, 64 GB RAM, 1512 GB SSD, system Windows 10),

- układ Jetson Nano A02 (CPU ARM Cortex A57 MPCore, GPU Nvidia Maxwell, 128 rdzeni CUDA, 4 GB RAM, system Linux w wersji Debian R32.3.1),
- kamera Raspberry Pi Camera HD V2 8 MPx (rozdzielczość 8MPx, HD 1080p/30 fps, czunik Sony IMX219),
- kontroler Xbox (częstotliwość Bluetooth 2.4 GHz),
- sterownik silników oparty o dwa chipsety TB6612 oraz kontroler PCA9685,
- karta sieciowa Intel Dual Band Wireless-AC 8265 (częstotliwość Bluetooth 2.4 lub 5 GHz, max. prędkość transmisji 867 Mbps).

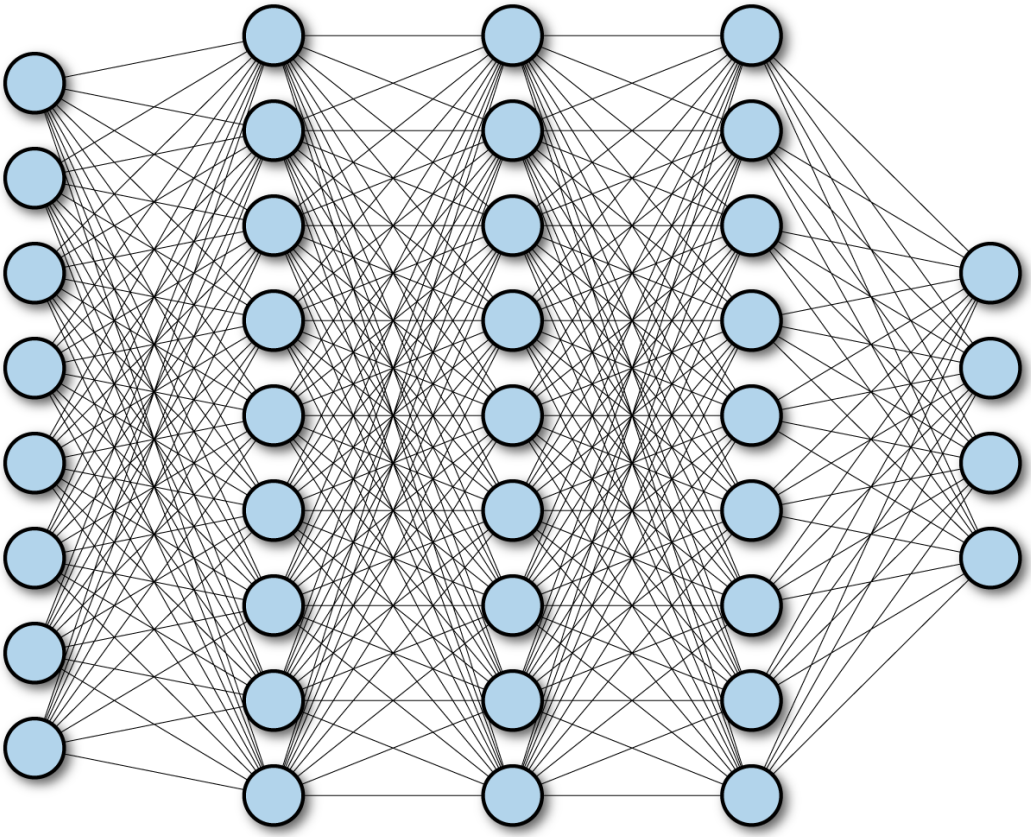
Rozdział 7

7. Optymalizacja sieci neuronowych

Rozdział opisuje wykorzystane architektury sieci neuronowych. Przedstawiono wyniki trenowania modeli przy zastosowaniu różnych hiperparametrów na utworzonych zbiorach danych odmiennej wielkości. Omówiono występujące ograniczenia oraz zastosowane techniki w celu osiągnięcia lepszych wyników nauki.

7.1. Sieci neuronowe w pełni połączone a sieci konwolucyjne

W uczeniu głębokim w detekcji obrazu dominują dwa rodzaje sieci - w pełni połączone sieci neuronowe - FCNN (ang. Fully Connected Neural Netowk) oraz opisane w powyższych rozdziałach splotowe sieci neuronowe - CNN. Są to podstawowe architektury głębokiego uczenia i prawie wszystkie inne głębokie sieci neuronowe są ich pochodnymi [8]. Warstwa gęsta to regularna głęboko połączona warstwa sieci. W pełni połączona sieć neuronowa składa się z serii w pełni połączonych warstw, które łączą każdy neuron w jednej warstwie z każdym neuronem w innej warstwie. Główną zaletą sieci w pełni połączonych jest to, że są one "agnostyczne strukturalnie", tzn. nie trzeba przyjmować żadnych specjalnych założeń dotyczących danych wejściowych. Mimo, że niezależność od struktury sprawia, że sieci w pełni połączone mają bardzo szerokie zastosowanie, to jednak mają one słabszą wydajność niż sieci specjalnego przeznaczenia dostrojone do struktury przestrzeni problemowej.



Rys. 7.1. Ilustracja w pełni połączone sieci neuronowe [8]

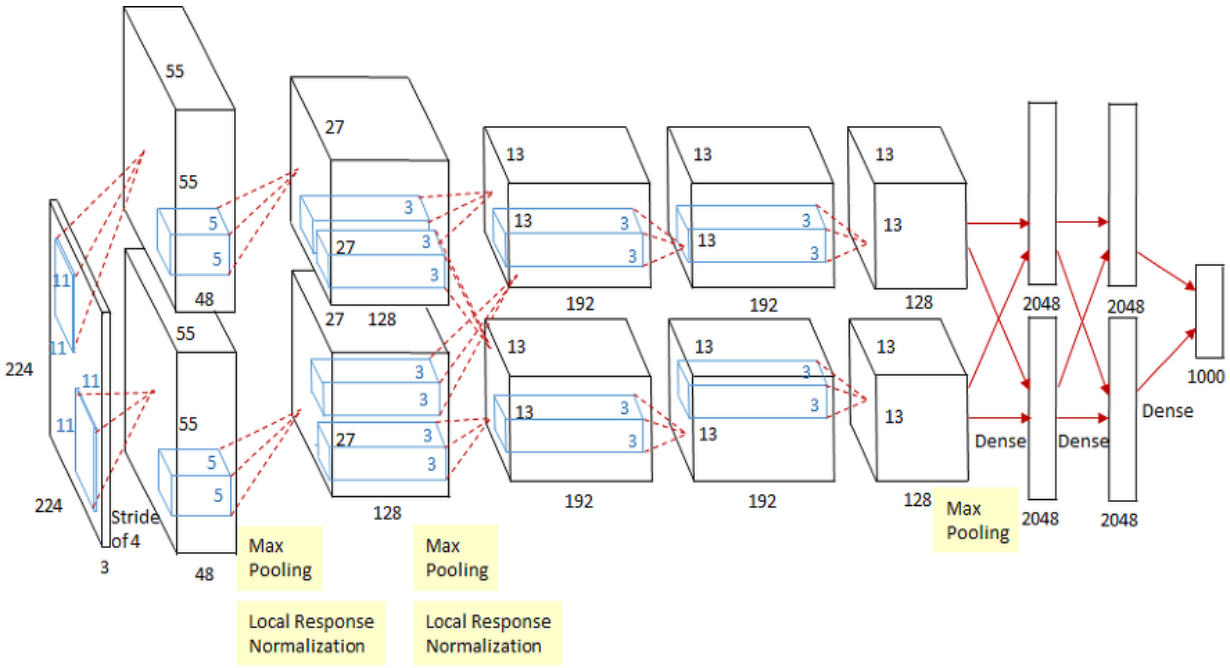
Wyżej wymienione architektury sieci głębokich przedstawiają kilka zasadniczych różnic :

- Warstwy konwolucyjne tworzą o około rząd wielkości więcej neuronów niż warstwy gęste. Każdy neuron jest kombinacją tylko i wyłącznie kilku cech z poprzedniej warstwy a nie wszystkich cech z poprzedniej warstwy jak to w przypadku warstw gęstych.
- Warstwy konwolucyjne starają się uczyć odtworzenia lokalnych wzorców natomiast warstwy gęste uczą się ich parametrów. Dodatkowo wzorce rozpoznawane przez sieć konwolucyjną są niezależne od przesunięcia.
- Neurony są grupowane na mapy cech. Każda mapa określa, czy konkretna cecha wizualna lub kombinacja cech wizualnych, jak w przypadku głębokich sieci CNN znajduje się w danej lokalizacji obrazu [7].

7.2. Architektura AlexNet

AlexNet to rozwinięta splotowa siec neuronowa CNN. Uważana jest za jedną z najbardziej dominujących głębokich sieci neuronowych w dziedzinie rozpoznawania obrazów z wykorzystaniem obliczeń prowadzonych na wielu procesorach graficznych GPU (ang. Graphics Processing Unit) [24]. Architektura (rys 7.2.) ta wprowadziła wiele nowoczesnych rozwiązań, które obecnie stały się standardem sieci konwolucyjnych. Wykorzystywana jest nienasycona funkcja aktywacji ReLU (rectified linear unit), która zapewnia lepszą wydajność oraz dużą szybszą zbieżność procesu uczenia niż funkcja sigmoidalna (sigmoid) lub funkcja tangesu hiperbolicznego (tanh) [21]. W celu zmniejszenia niedopasowania w warstwach w pełni połączonych jednocześnie unikając przetrenowania pomimo wykorzystywania bardzo dużego zbioru uczącego zastosowano opracowaną niewiele wcześniej metodę regularyzacji zwaną „dropout". Dodatkowo dzięki wykorzystaniu nieliniowego korygowania wag nastąpiło sześciokrotne skrócenie czasu uczenia. Jednak czas uczenia sieci jest wydłużony poprzez wykorzystanie w swoim algorytmie wyłączania neuronów o określonym prawdopodobieństwie skutkując wykorzystywaniem innej puli neuronów w każdej iteracji treningu sieci [26]. Dodatkowo w swojej pracy pt. *ImageNet Classification with Deep Convolutional Neuarl Network* [26] autorzy architektury przedstawili informacje o zasadności stosowania augmentacji danych wpływając na poprawę wyników uczenia. Architektura zbudowana jest z ośmiu warstw. Pięć pierwszych warstw to warstwy konwolucyjne natomiast trzy ostanie to warstwy połączone ze sobą w pełni. Pomiędzy warstwami konwolucyjnymi występuje warstwa max-pooling. Z tego powodu AlexNet jest podobna do architektury LeNet [25] jednak różnica występuje w wielkości modelu, gdyż AlexNet posiada 60 milionów parametrów. Pierwsza warstwa splotowa odpowiada za przetwarzanie fragmentów obrazu o wymiarach 224 x 224 piksele z trzema kanałami RGB. Wykorzystywane do tego jest 96 filtrów o skoku równym 4 i rozmiarach 11x11x3. Operacja max-pooling wykonywana jest w oknach o wymiarach 3x3 nachodzących na siebie z skokiem o wartości 2. Uzyskana odpowiedź sieci jest lokalnie normalizowana. Największym wyzwaniem dla architektur w czasie ich powstawania są dostępne zasoby obliczeniowe w trakcie nauki. Zważając na ten fakt twórcy architektury AlexNet uwzględniając jej rozmiar dokonali podziału przetwarzania na dwa procesory graficzne GPU. Rozwiązanie to wpłynęło na architekturę. Filtry znajdujące się na tej samej jednostce graficznej są połączone z drugą, czwartą i piątą warstwą splotową. Następne warstwy wykorzystują malejące okno przetwarzania o

wymiarach 5x5 oraz 3x3. Następuje zwiększenie liczby wykorzystywanych filtrów do 256 i 384. Dwie warstwy gęste z których każda zawierają 4096 neuronów posiadają większą część występujących w sieci parametrów [27].



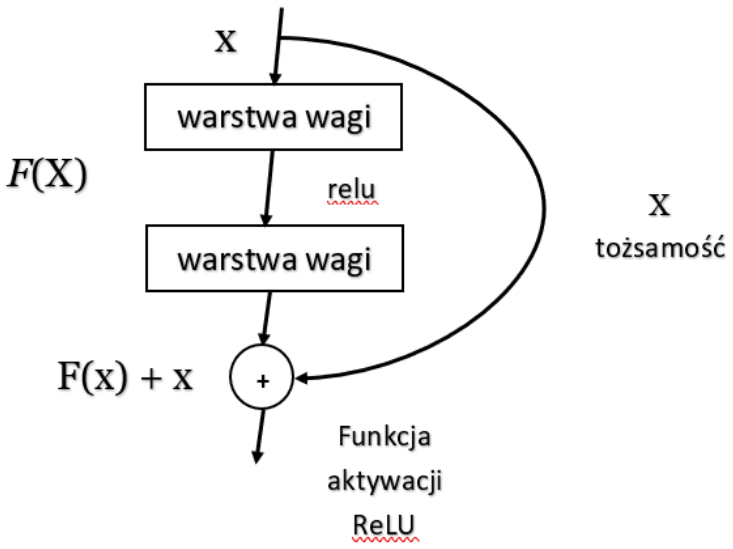
Rys. 7.2. Struktura sieci AlexNet [82]

7.3. Architektura ResNet

Architektura ResNet opiera się na sieciach rezydualnych. Sieci te w swojej strukturze wykorzystują fakt, że w wyniku zwiększania liczby stosowanych warstw (przyjmowanych jako udoskonalenie tworzonych modeli) następuje przekroczenie wartości granicznej i pojawia się wysoki spadek skuteczności modelu. Efekt ten jest pierwotnie podobny do zjawiska przeuczenia sieci ponieważ początkowo występuje poprawa dokładności modelu poprzez dodawanie kolejnych warstw. Jednak w granicznym punkcie kolejne iteracje zwiększania głębokości zaczynają determinować obniżanie efektywności modelu. Natomiast spadek ten nie dotyczy tylko zbioru walidacyjnego, lecz także zbioru trenującego. Teoretycznie modele głębokie powinny przedstawiać wyniki co najmniej nie gorsze niż płytsze odpowiedniki jednak na podstawie powyższych informacji modele takie stają się nadmiernie trudne w optymalizacji. Spostrzeżenia te zostały przedstawione w publikacji pt. *Deep Residual Leraning for Image Recognition* [28].

Autorzy przedstawili także rozwiązanie – biorąc za punkt wyjścia model płytszy, dodawane do niego kolejne warstwy w najgorszym wypadku (gdy nie jest możliwe znalezienie lepszej reprezentacji) powinny po prostu wprowadzić przekształcenie tożsamościowe. W praktyce jednak znalezienie tak teoretycznie trywialnego rozwiązania okazuje się zbyt trudne dla stosowanych optymalizatorów, dlatego autorzy wprowadzają tę koncepcję odgórnie do zaproponowanej architektury wymuszając uczenie funkcji rezydualnych (resztowych) [27].

Architektura ResNet została utworzona przez zespół Kaiming He [28]. Innowacyjność sieci Resnet polega na zastosowaniu normalizacji używanej do małego zbioru próbek uczących (wartości średniej zerowej oraz jednostkowej wariancji) nazywanych batch normalization [25]. Dodatkowo występują powiązania z dalszymi warstwami oraz pomijane są powiązania z warstwami sąsiednimi. Głównym elementem sieci ResNet jest moduł, który do typowego sekwencyjnego złożenia warstw dodaje połączenie zachowujące niezmienioną wartość wejściową X. Moduł ten przestawiono na rys 7.3. Uczy on się zamiast pewnego przekształcenia $f(x)$ funkcji $y = f(x) + x$. Powoduje to, że gdyby w skrajnym wypadku okazało się, że przekształcenie tożsamościowe jest optymalne, to łatwiej jest przy takim podejściu zrównać część rezydualną do zera niż wyuczyć ciąg standardowych warstw takiego przekształcenia od podstaw [27].



Rys. 7.3 Element podstawowy w sieci rezydualnej [28]

Autorzy publikacji *Deep Residual Leraning for Image Recognition* przedstawili na podstawie przeprowadzonych badań kilka ważnych wniosków:

- bardzo głębokie sieci rezydualne są łatwe w optymalizacji w przeciwieństwie do podobnych modeli pozbawionych połączeń rezydualnych (Autorzy analizowali modele z 18 i 34 warstwami ukazując, że większa liczba warstw pogarsza wyniki uczenia, natomiast nie występuje to w przypadku sieci rezydualnych),
- występuje łatwiejsze uzyskiwanie poprawy poprzez operację zwiększanie głębokości.

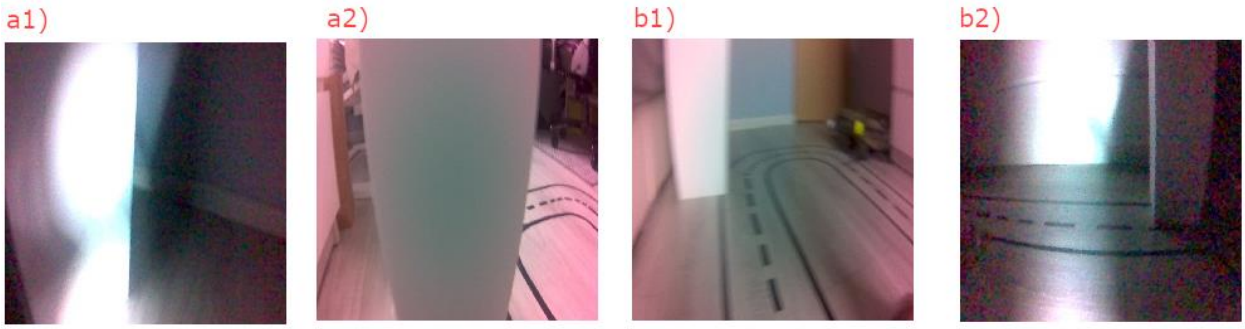
Kaiming He wraz z zespołem w 2016 roku w publikacji *Deep Residual Leraning for Image Recognition* [30] przeprowadzili badania w kontekście modeli ResNet zbudowanych z 50, 101 oraz 152 warstw. W wyniku badań uzyskali oni malejącą wartość błędu walidacji. Sieć ResNet 152 w porównaniu z architekturą VGGNet [31] charakteryzuje się mniejszą złożonością pomimo olbrzymiej różnicy w głębokości. Późniejsze publikacje zespołu Kaiming He [32] przedstawiają wyniki, że sieci rezydualne pozwalają tworzyć modele składające się nawet z 1000 warstw. Architektura ResNet to jedna z najgłębszych oraz najbardziej skutecznych sieci w zadaniach rozpoznawania obrazu jednak cechująca się dużą złożonością obliczeniową [27].

7.4. Transfer wiedzy

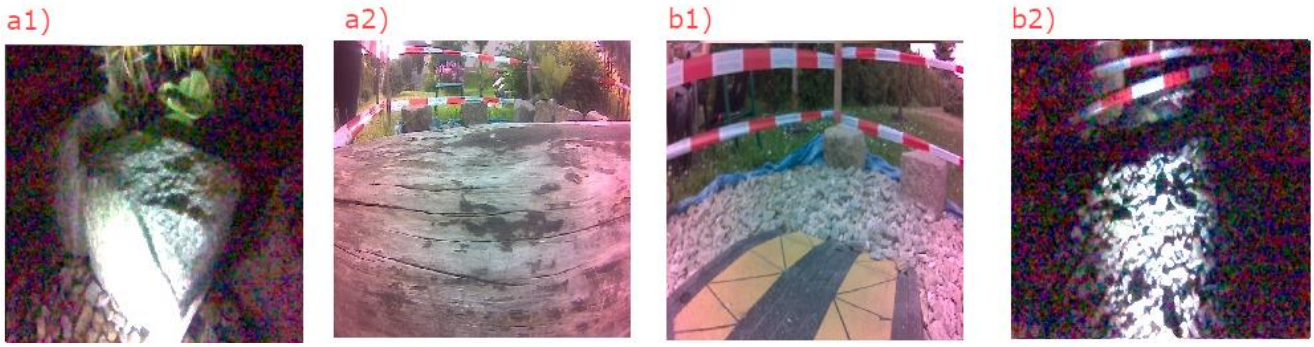
W celu osiągnięcia lepszych wartości nauki modelu, a także przyśpieszenia czasu szkolenia wykorzystano podejście uczenia poprzez transfer wiedzy (ang. transfer learning). W podejściu tym wykorzystane zostało spostrzeżenie dotyczące obserwacji, że tworzone modele głębokie przy wykorzystaniu odpowiednio licznych zbiorów danych tworzą reprezentacje na tyle ogólne, że istnieje możliwość ich zastosowana również w innych przypadkach, w których występuje ograniczony dostęp do danych uczących. Wstępnie wytrenowane na milionach obrazów zbioru ImageNet modele tj. AlexNet lub ResNet posiadają wyuczone w warstwach cechy wagi podczas pierwotnego szkolenia. Wykorzystywany w projekcie magisterskim zbiór danych posiada tylko dwie etykiety klas w porównaniu z 1000 etykiet pierwotnie wytrenowanych w modelach. Z tego powodu całkowicie zastąpiono wagi oraz cechy tylko w ostatniej nowej niewyszkolonej warstwie, która posiada tylko dwa wyjścia uzyskując istotną oszczędność czasową oraz poprawę zdolności generalizacji.

7.5. Zbiór danych

W procesie budowy systemu sterowania opartym na przetwarzaniu obrazu bazującym na głębokich sieciach neuronowych największym wyzywaniem jest utworzenie zbioru danych uczących. Działanie systemu oraz wykazywana skuteczność przejazdów współzależna jest z liczbą, jakością oraz różnorodnością posiadanych przykładów w zbiorach danych. Trudność ta związana jest z osobliwością ukształtowania miejsc w których operuje pojazd. Z tego powodu nie występują konkretne dane w zbiorach internetowych, których można użyć. Możliwe jest wykorzystanie ogólnodostępnego zbioru np. ImageNet składającego się z ponad 14 milionów obrazów wraz z adnotacjami przez co można wyczuć sieć identyfikacji elementów. Działanie to jednak nie jest przedmiotem tej pracy, w której celem jest poprawny przejazd platformy robotycznej po utworzonych torach testowych poprzez klasyfikowanie zajętości trasy. Z tego powodu zadanie utworzenia zbioru uczącego jest czynnością pracochłonną w kontekście zróżnicowania danych pod względem możliwości występujących przeszkód. Zgodnie z założeniami zadania eksploracyjne prowadzone są w warunkach dziennych i nocnych. W związku z tym utworzono zestawu danych dla obu przypadków. Przykłady utworzonych zdjęć ze zbiorów testowych przedstawiono poniżej na rys 7.4. oraz 7.5. Warto zaznaczyć, że zniekształcenia obrazu oraz charakterystyczne przesunięcie ku czerwieni przy krawędziach obrazu wynikają ze specyfiki stosowanej kamery. Ciekawe jest także to, że w warunkach nocnych kamera generuje znacznie większy szum.



Rys. 7.4. Przykładowe zdjęcia z zbioru toru testowego w warunkach wewnętrznych od lewej – a1) zajętość drogi w warunkach nocnych; a2) zajętość drogi w warunkach dziennych; b1) wolna droga w warunkach dziennych; b2) wolna droga w warunkach nocnych



Rys. 7.5. Przykładowe zdjęcia z zbioru toru testowego w warunkach zewnętrznych od lewej – a1) zajętość drogi w warunkach nocnych; a2) zajętość drogi w warunkach dziennych; b1) wolna droga w warunkach dziennych; b2) wolna droga w warunkach nocnych

Rozmiar zdjęć w zbiorze uczącym skalowany jest programowo do rozmiaru 224 x 224 [px] w celu dostosowania ich do pierwszej warstwy architektur AlexNet oraz ResNet, a także aby zmniejszyć rozmiar danych przekazywanych do sieci. W trakcie realizacji pracy przeprowadzono badania nad zależnością pomiędzy wielkością zbioru uczącego a wynikami nauki modelu sieci oraz poprawnością pracy jednostki robotycznej po jej zaimplementowaniu. Do realizacji tego zadania utworzono kilka zbiorów danych o różnych wielkościach, a ich rozkład przedstawiono w poniższej tabeli 2.1. Przyjęty podział na zbiory o wartościach 300, 500 i 800 próbek zostały ustalone niezależnie przez autora niniejszej pracy w związku z istniejącymi możliwościami czasowymi tworzenia zbiorów.

Tor wewnętrzny		**Tor zewnętrzny**	
Dzień	**Noc**	**Dzień**	**Noc**
Liczba próbek			
300		300	
500		500	
800		800	

Tab 2.1.: Przyjęte wielkości utworzonych zestawów danych

7.6. Optymalizacja parametrów sieci AlexNet oraz ResNet

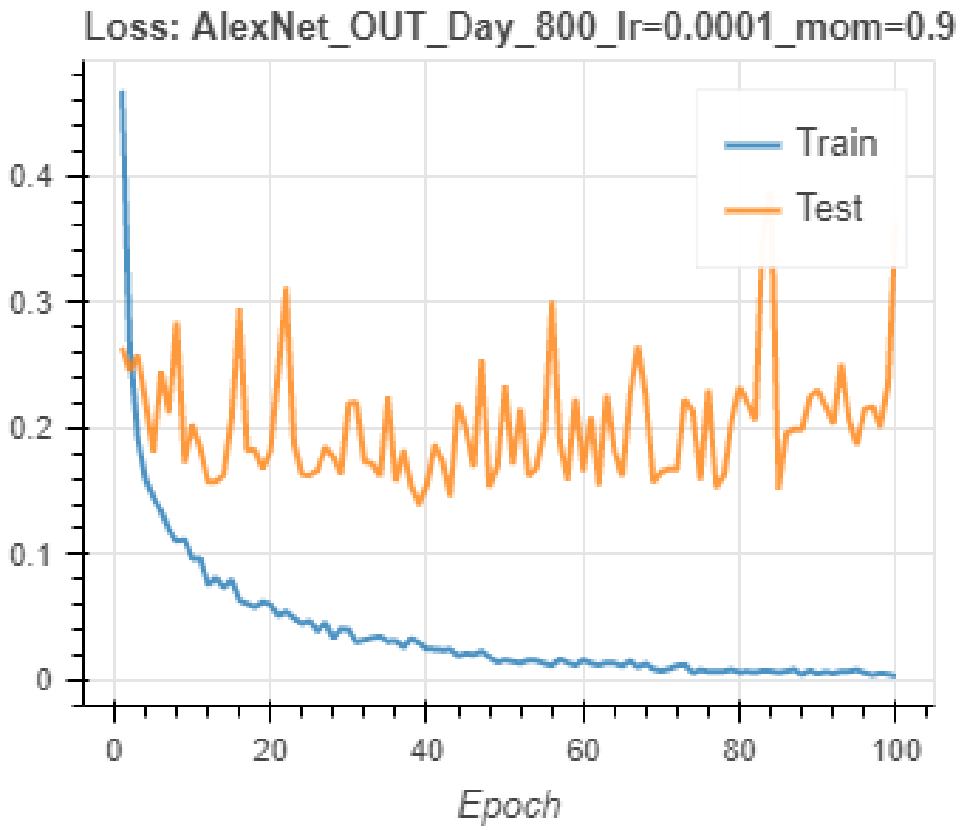
Projekt Jetbot AI posiada wstępnie określone parametry sieci neuronowych ustalone przez firmę Nvidia. Wykorzystywana jest biblioteka PyTorch, gdyż posiada m.in. funkcję obliczania tensorowego z silnym przyspieszeniem szybkości obliczeń na GPU oraz system tape-base autograd będący techniką wykorzystywaną do wydajnego obliczenia gradientów w operacji propagacji wstecznej podczas nauki modelu. W pracy dyplomowej przeprowadzono badania nad optymalizacją parametrów w celu osiągnięcia lepszych wyników uczenia modelu. Parametrami podlegającymi optymalizacji są:

- *transforms.normalize* – normalizacja obrazu poprzez przekazanie wartości średniej i odchylenia standardowego. W trakcie prowadzonych badań w obydwu architekturach zastosowano zgodnie z dobrą praktyką wartości wyuczone dla zbioru ImageNet obliczone na podstawie wielu iteracji na milionach zdjęć (wartość średniej - [0.485, 0.456, 0.406], odchylenie standardowe - [0.229, 0.224, 0.225]),
- *torch.utilsdata.random_split* – losowy podział danych na zbiór testowy oraz uczący na nienakładające się na siebie przedziały o zadanych długościach (
  - zbiór uczący – obrazy z tego zbioru używane są do trenowania modelu przez co powinien być reprezentatywną próbą pozwalając na poprawę generalizacji problemu,
  - *zbiór testowy* – wykorzystywany po zakończeniu procesu nauki, pozwala sprawdzić jak dobrze wyuczył się model),
- *batch.size* – określa liczbę załadowanych próbek do model w trakcie jednej iteracji,
- *num_workers* – określa ile warstw umieszczają dane w pamięci RAM dla procesów roboczych,
- *num_epochs* – określa liczbę epok,
- *learning_rate* – określa wielkość kroku w każdej iteracji podczas zbliżania się do minimum funkcji straty,
- *momentum* – wprowadza do procesu optymalizacji bezwładności, która pomaga w nie zatrzymywaniu się sieci w minimach lokalnych.

7.7. Trenowanie sieci neuronowej

W kolejnym etapie wykonano trenowanie sieci neuronowych dla utworzonych zbiorów danych. Wszystkie obliczenia zostały wykonane na komputerze klasy laptop laptop (CPU Intel Core i7-11800H, GPU RTX 3060, 64 GB RAM, 1512 GB SSD, system Windows 10). Obliczenia rozproszone są wykonywane na układzie graficznym ze względu na jego budowę i charakteryzującym się dużo większą szybkością aniżeli procesory z powodu możliwości przetwarzania równoległego operacji na wielu zestawach danych. Zważywszy na to wszystkie obliczenia wykonano na jednostce graficznej. Trenowanie wykonywane jest z wykorzystaniem rdzeni CUDA (ang. Compute Unified Device Architecture) [44] – technologii opracowanej przez firmę Nvidia będącą uniwersalną architekturą procesorów wielordzeniowych (głównie kart graficznych) umożliwiającą wykorzystanie ich mocy obliczeniowej do rozwiązywania ogólnych problemów numerycznych w sposób wydajniejszy, niż w tradycyjnych sekwencyjnych procesorach ogólnego zastosowania.

Wykonano 100 iteracji obliczeń dla każdego zbioru danych. W początkowej fazie trenowania wystąpił problem z niereprezentatywnością danych walidacyjnych wynikający ze złego dopasowania podziału na zbiór testowy (rys. 7.6). Wartość optymalna rozdzielenia procentowego zbioru danych nie jest znana *a piori*. Z tego powodu w trakcie przeprowadzonych badań dostosowywano podziału na zbiór testowy i treningowy w zależności od występujących potrzeb modelu. Standard który przyjęto aby uzyskać optymalny wynik to uzyskanie wielkości zbioru testowego kształtującego się w zakresie od 10 do 20 [%] zbioru danych.



Rys. 7.6. Przykład krzywej uczenia się trenowania i walidacji ukazujący zestaw danych testowych, który jest zbyt mały w stosunku do zestawu danych szkoleniowych

7.8. Trening architektury AlexNet

Jedną z najważniejszych części szkolenia modelu, aby działał dobrze jest wybór hiperparametrów. Hiperparametry służą do sterowania procesem uczenia. Są to m.in. współczynnik uczenia, wartość pędu, wielkość partii oraz liczba warstw procesów roboczych. Hiperparametry powinny być również profilaktycznie okresowo przewartościowywane, ponieważ w miarę zmiany ilości danych oraz użycia nowego GPU/CPU wybrane wcześniej wartości mogą już nie być optymalne. Jest to często określane jako proces starzenia się hiperparametrów [76]. Najlepiej dobrane wartości hiperparamterów zostały przedstawione w tabeli 2.2. Ogólnie przyjmuje się, że istnieje pewien najlepszy punkt dla wielkości partii zakresie od 1 do całego zestawu danych uczących, który zapewni najlepsze uogólnienie [83]. Wspomniana najlepsza wartość wielkości partii zależy od zestawu danych i danego modelu. Przeprowadzono badania nad różnymi wielkościami 32, 64, 128, 256 oraz 512 (rys.7.7).

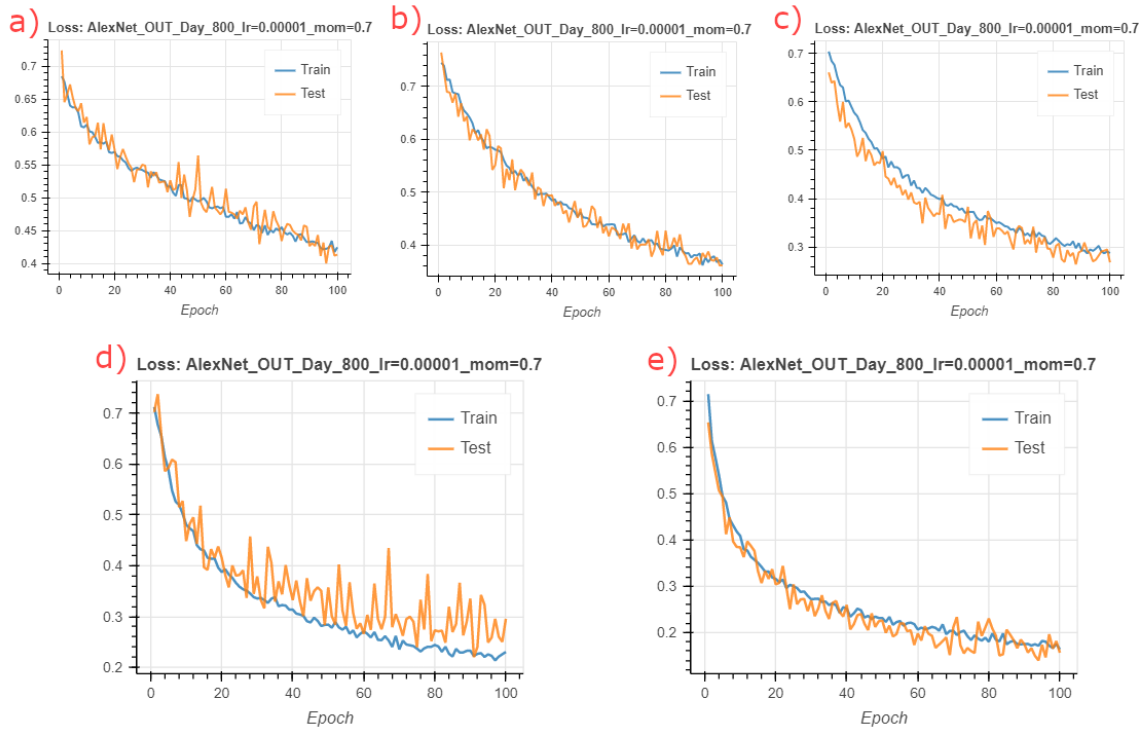
Parametr	**Architektura AlexNet**					
Tor testowy	wewnętrzny					
warunki	dzień			noc		
Zbiór danych	300	500	800	300	500	800
batch_size	32	32	32	32	32	32
num_workers	8	8	8	8	8	8
leraning_rate	1E-4	1E-5	1E-5	1E-6	1E-5	1E-5
momentum	0.7	0.7	0.7	0.7	0.9	0.9

Parametr	**Architektura AlexNet**					
Tor testowy	zewnętrzny					
warunki	dzień			dzień		
Zbiór danych	300	300	300	300	300	300
batch_size	32	32	32	32	32	32
num_workers	8	8	8	8	8	8
leraning_rate	1E-5	1E-5	1E-5	1E-4	1E-4	1E-4
momentum	0.9	0.9	0.7	0.9	0.9	0.7

Tab 2.2: Parametry wejściowe dla poszczególnych zestawów danych w architekturze AlexNet

Większe wielkości partii doprowadzały do złego uogólniania (literatura optymalizacyjna opisuje ten problem jednak obecnie nie istnieją wyjaśnienia tego procederu [42]). Na poniższym rysunku przedstawiono wartość straty modelu z odpowiednio zwiększaną wielkością partii zgodnie z kolejną potęgą liczby 2 ze względu na sposób przeprowadzania obliczeń przez komputer. Prace w literaturze optymalizacyjnej [42] wykazały natomiast, że zwiększenie szybkości uczenia się może zrekompensować większe rozmiary partii. Mając na uwadze mnogość działań wpływających na wynik trenowania stwierdzono, że optymalną wartością parametru będzie rozmiar 32. Liczba

warstw procesów roboczych zgodnie z dobrą praktyką została ustawiona na liczbę równą liczbie rdzeni procesora [43]. W przypadku wykorzystywanego sprzętu parametr ten ustawiono na wartość osiem. Warto dodać, że zdefiniowanie cechy *num_workers* powyżej liczby 16 generuje automatyczne ostrzeżenie o zbyt dużej liczbie elementów w zestawie ładującym dane – Data Loader, co skutkować może powolnym działaniem lub zatrzymaniem procesu szkolenia. Przeprowadzono badania nad parametrem szybkości uczenia, który jest jednym z najważniejszych, jeśli nie krytycznym hiperparametrem, ponieważ określa amplitudę skoku jaki ma wykonać technika optymalizacji w każdej iteracji. Jeśli wskaźnik jest bardzo niski to osiągnięcie zbieżności zajmie dużo czasu. A jeśli jest bardzo wysoki to może oscylować wokół minimum lub nawet się rozbiec [45]. Dlatego najlepszym sposobem na określenie prawidłowych współczynników uczenia jest przeprowadzenie eksperymentów z ręcznym dostrojeniem parametru. Sieć uczono w zakresie parametru *learning_rate* od 0.1 do 0.000001 oraz wartością *momentum* 0.7 lub 0.9. Gdy algorytm przedstawiał zbieżną oraz nie oscylującą wartość funkcji kosztu został wybrany do zaimplementowania na platformie robota eksploracyjnego. Wyniki dotyczące uzyskanych wyników zaprezentowano w tabeli 2.3.



Rys. 7.7 Przykład krzywych uczenia się trenowania i walidacji w zależności od wielkości partii od lewej – a) 512, b) 256, c) 128, d) 64, e) 32

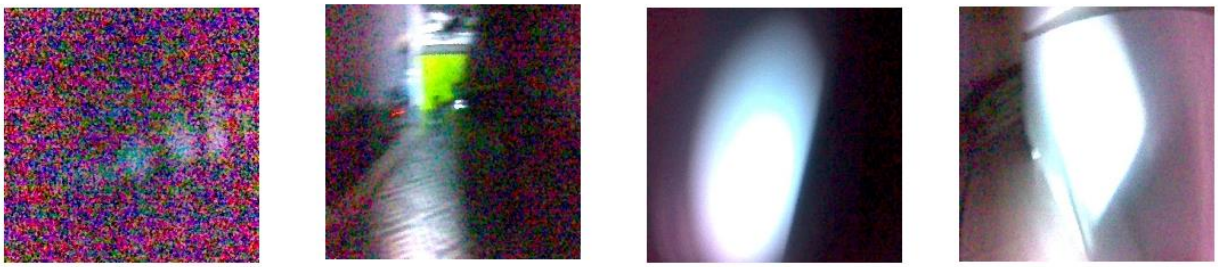
Parametr	**Architektura AlexNet**					
Tor testowy	wewnętrzny					
warunki	dzień			noc		
Zbiór danych	300	500	800	300	500	800
Legenda	Opis zastosowany do rozdzielenia wyników na zbiorach - Testowy (treningowy)					
Strata uczenia	0.11 (0.12)	0.28 (0.34)	0.20 (0.24)	0.57 (0.63)	0.28 (0.24)	0.2 (0.23)
Skuteczność klasyfikacji	0.95 (0.96)	0.9 (0.86)	0.92 (0.92)	0.71 (0.66)	0.86 (0.88)	0.90 (0.91)

Parametr	**Architektura AlexNet**					
Tor testowy	zewnętrzny					
warunki	dzień			noc		
Zbiór danych	300	500	800	300	500	800
Legenda	Opis zastosowany do rozdzielenia wyników na zbiorach - Testowy (treningowy)					
Strata uczenia	0.04 (0.04)	0.08 (0.08)	0.16 (0.16)	0.18 (0.19)	0.22 (0.2)	0.14 (0.08)
Skuteczność klasyfikacji	0.96 (0.98)	0.97 (0.97)	0.94 (0.92)	0.95 (0.92)	0.92 (0.9)	0.941 (0.97)

Tab 2.3. Wartość skuteczności oraz funkcji kosztu uzyskanych modeli w architekturze AlexNet

Na przedstawionych wynikach można zauważyć, że najmniejsze zestawy danych wynoszące około 300 zdjęć zajętej oraz wolnej drogi przedstawiły najlepsze wartości wyuczenia modelu. Zgodnie z oczekiwaniami dla mniejsze liczby próbek modele bardzo

szybko osiągnęły okolice 97 [%] skuteczności na danych uczących. Natomiast w zbiorze testowym osiągnięto około 95 [%] skuteczność co ukazuje, że testowany model bardzo dobrze potrafi wydobyć z zbioru obrazów cechy odpowiednio klasyfikując stan zajętości trasy. Wyjątkiem są tutaj wyniki uzyskane z zbioru danych zebranych w warunkach nocnych na torze wewnętrznym. Model przedstawia tutaj najgorszą wartość funkcji kosztu równą 0.63 na danych uczących oraz skuteczność na poziomie 66 [%]. Po przeanalizowaniu zbioru danych zauważono, że znajduje się tam wiele zdjęć nieostrych oraz z bardzo słabym oświetleniem przedstawiającym kluczowe przedmioty wielościenne wpływające na możliwość przejazdu (rys 7.8). Wynika to z czasu tworzenia zbioru odbywającego się na początkowym etapie projektu. Autor pracy nie posiadał wtedy doświadczenia w zbieraniu danych, dostosowaniu kamery oraz stylu jazdy. Znaczenia nabiera tutaj jednak używane w informatyce wyrażenie dotyczące danych *"garbage in, garbage out*" określające, że jakość danych wyjściowych jest określona przez jakość danych wejściowych. Oznacza to, że wadliwe, słabe lub niekompletne dane powodują niedokładne przewidywania nawet przy najlepszym doborze parametrów oraz architektur. Na podstawie tych doświadczeń zwrócono uwagę na poprawność danych zebranych na torze testowym w warunkach zewnętrznych co przedstawiają uzyskane wyniki nauki.



Rys. 7.8 Przykład niepoprawnie zebranych danych na torze testowym w warunkach wewnętrznych w trybie nocnym.

7.9. Trening architektury ResNet 18

W treningu sieci rezydualnych również wykorzystano hiperparametry zastosowane w architekturze AlexNet. Wykonano 100 iteracji obliczeń dla każdego zbioru danych. Sieć uczono w zakresie parametru *learning_rate* od 0.1 do 0.000001 oraz wartością *momentum* 0.7 lub 0.9. Natomiast wielkość partii przyjęto równą liczbie 8. Wynika to z

pojawiającego się błędu *RuntimeError: CUDA out of memory* odnoszącego się do braku miejsca na zestawy mini partii danych w pamięci GPU w momencie zastosowania większych rozmiarów partii danych. Najlepiej dobrane wartości hiperparamterów zostały przedstawione w tabeli 2.4.

Parametr	**Architektura ResNet18**					
Tor testowy	wewnętrzny					
warunki	dzień			noc		
Zbiór danych	300	500	800	300	500	800
batch_size	8	8	8	8	8	8
num_workers	8	8	8	8	8	8
leraning_rate	1E-4	1E-4	1E-4	1E-4	1E-4	1E-4
momentum	0.7	0.7	0.9	0.9	0.7	0.9

Parametr	**Architektura ResNet18**					
Tor testowy	zewnętrzny					
warunki	dzień			noc		
Zbiór danych	300	500	800	300	500	800
batch_size	8	8	8	8	8	8
num_workers	8	8	8	8	8	8
leraning_rate	1E-4	1E-4	1E-5	1E-3	1E-4	1E-4
momentum	0.9	0.9	0.9	0.7	0.9	0.7

Tab 2.4: Parametry wejściowe dla poszczególnych zestawów danych w architekturze ResNet18

Model ResNet 18 osiągnął najlepszy wynik na danych nocnych utworzonych z toru testowego w warunkach zewnętrznych równych 99 [%] skuteczności na zbiorze uczącym oraz 98 [%] na zbiorze treningowym. Wyniki dotyczące uzyskanych wyników zaprezentowano w tabeli 2.5. Utrata modelu na zestawie danych uczących jest niewiele mniejsza, niż na danych walidacyjnych. Niewielka różnica pomiędzy dwiema końcowymi

stratami nosi nazwę *generalization gap*. Minimalna wartość tej różnicy wskazuje, że model został poprawnie wytrenowany i dobrze uogólnia otrzymane dane. Widoczna po ostatniej epoce tendencja minimalizacji wartość *generalization gap* wskazuje że powinno się zwiększyć liczbę epok i ponownie przetrenować model. Jednak dobre praktyki wskazują [1], że dalsze szkolenie w celu lepszego dopasowania prawdopodobnie doprowadzi do nadmiernego dopasowania modelu (ang. overfitting) co zostało przedstawione w podrozdziale 7.10.

Parametr	**Architektura ResNet18**					
Tor testowy	wewnętrzny					
warunki	dzień			noc		
Zbiór danych	300	500	800	300	500	800
Legenda	Opis zastosowany do rozdzielenia wyników na zbiorach - Testowy (treningowy)					
Strata uczenia	0.1 (0.11)	0.13 (0.11)	0.12 (0.08)	0.25 (0.16)	0.36 (0.28)	0.25 (0.21)
Skuteczność klasyfikacji	0.94 (0.97)	0.97 (0.98)	0.94 (0.97)	0.92 (0.95)	0.86 (0.91)	0.92 (0.95)

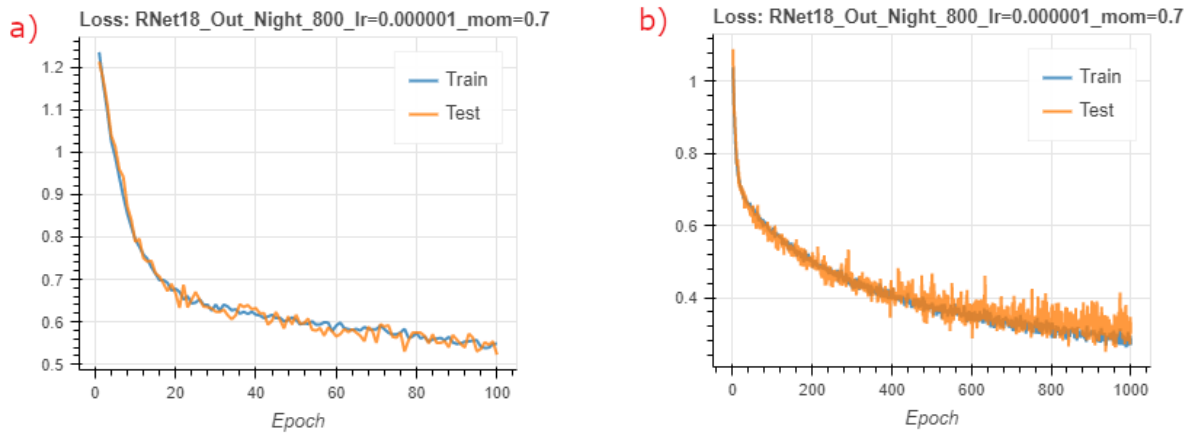
Parametr	**Architektura ResNet18**					
Tor testowy	zewnętrzny					
warunki	dzień			noc		
Zbiór danych	300	500	800	300	500	800
Legenda	Opis zastosowany do rozdzielenia wyników na zbiorach - Testowy (treningowy)					
Strata uczenia	0.2 (0.06)	0.18 (0.09)	0.14 (0.2)	0.03 (0.02)	0.1 (0.26)	0.12 (0.24)
Skuteczność klasyfikacji	0.9 (0.98)	0.94 (0.98)	0.95 (0.97)	0.98 (0.99)	0.86 (0.95)	0.91 (0.97)

Tab 2.5. Wartość skuteczności oraz funkcji kosztu uzyskanych modeli w architekturze ResNet18

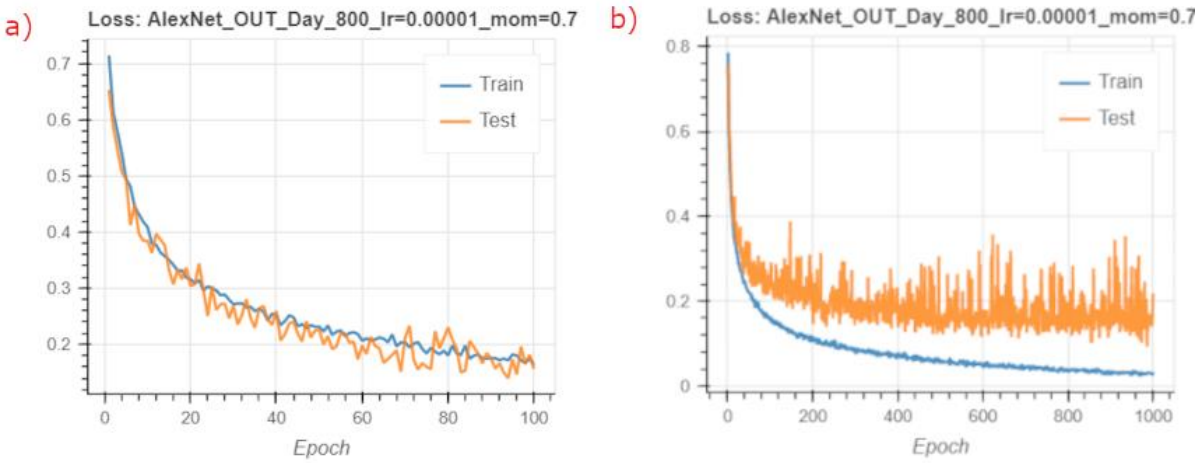
W rezultacie porównania wyników nauki architektur uzyskano lepsze wyniki w sieci ResNet 18 w kontekście danych z torów testowych w warunkach wewnętrznych oraz w warunkach nocnych na torze zewnętrznym. Natomiast architektura AlexNet osiągnęła nieznacznie lepsze wyniki nauki na danych w warunkach dziennych na torze zewnętrznym różniące się maksymalnie o 0.1 wartości straty. Lepszy wynik architektury ResNet18 charakteryzującą się większą liczbą warstw splotowych jest zgodny z główną zaletą tego modelu opisaną przez twórców tego modelu [75].

7.10. Trenowanie modelu z użyciem większej liczby epok

Na podstawie analizy otrzymanych wykresów wybrano po jednym zestawie parametrów z architektury AlexNet oraz ResNet 18, których wyniki wskazywały na polepszenie efektów nauki poprzez zwiększenie liczby epok. Modele swoje najlepsze rezultaty osiągnęły w ostatniej epoce, co sugeruje że model mógłby osiągnąć jeszcze nieznacznie wyższą skuteczność, gdyby zwiększyć liczbę epok. Z tego powodu dokonano nauki modelu na 1000 iteracjach obliczeń w celu sprawdzenia czy taki zabieg wpłynie na poprawę wyników. W przypadku wyników uzyskanych dla architektury AlexNet (rys 7.10) oraz ResNet (rys. 7.9) możemy zauważyć, że nie nastąpiła zakładana znaczna poprawa wyników nauki, tylko minimalne zmniejszenie wartości straty w przypadku sieci ResNet. Natomiast w architekturze AlexNet nastąpiło pogorszenie otrzymanego wyniku. Modele zbyt dobrze dopasowały się do uczącego zestawu danych i doszło do zjawiska tzw. *overfittingu*. Problem z nadmiernym dopasowaniem polega na tym, że im bardziej model staje się wyspecjalizowany w uczeniu danych, tym gorzej jest w stanie uogólnić nowe dane, co skutkuje wzrostem błędu uogólnienia.



Rys. 7.9. Wyniki krzywych uczenia się trenowania i walidacji architekturze ResNet 18 po zastosowania od lewej a) 100 iteracji b) 1000 iteracji



Rys. 7.10. Wyniki krzywych uczenia się trenowania i walidacji architekturze AlexNet po zastosowania od lewej a) 100 iteracji b) 1000 iteracji

W otrzymanych wynikach nauki modeli treningowych zauważono zależność, że podczas trenowania modeli głębokiego uczenia z dokładnie tymi samymi hiperparametrami otrzymano różne wyniki. Literatura definiuje ten problem poprzez następującą każdorazową inicjalizację wag, jak także również tasowanie danych treningowych [1]. W związku z tym prowadzono archiwizację wszystkich wyuczonych modeli, gdyż przyjęty najlepszy rozkład parametrów dla danego zestawu danych nie gwarantuje uzyskania ponownie tak dobrze wytrenowanego modelu.

Rozdział 8

8. Badania weryfikacyjne

Rozdział zawiera szczegółowy opis szeregu badań przeprowadzonych w ramach pracy magisterskiej na utworzonych torach testowych wraz z przestawieniem implementacji systemu sterowania na platformie robotycznej Orzeł 7.

8.1. Implementacja systemu

Jednostka obliczeniowa robota eksploracyjnego - układ Jeton Nano A02 4GB to niezależny minikomputer przeznaczony do zadań związanych z sztuczną inteligencją. Z tego powodu może być obsługiwany jak standardowy komputer PC z użyciem urządzeń peryferyjnych tj. mysz, klawiatura i monitor (wyłącznie HDMI). Alternatywnym sposobem połączenia jest wykorzystanie zdalnego pulpitu za pomocą protokołu RDP (ang. Remote Desktop Protocol) lub łączenie terminalowe z wykorzystaniem protokołu SSH, który jest stosowany w realizowanej pracy magisterskiej. Implementację systemu wykonano na utworzonej platformie dydaktycznej robota eksploracyjnego Orzeł 7 przedstawionego na rys 8.1.

Rys. 8.1. Widok platformy dydaktycznej robota eksploracyjnego Orzeł 7 [zdjęcie: Marcin Januszka]

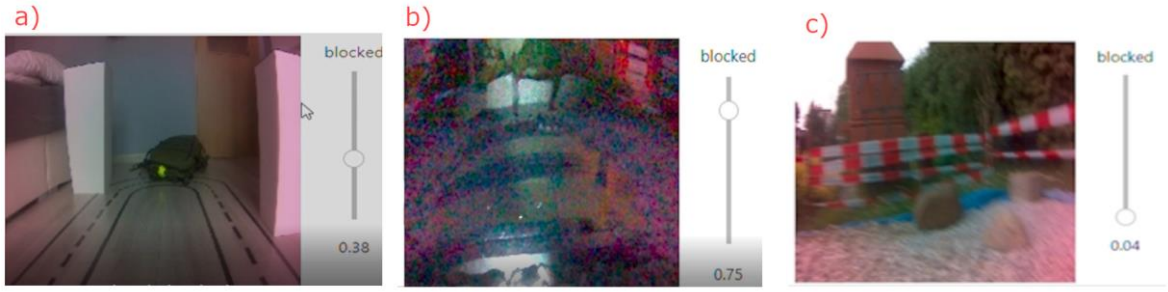
Przed rozpoczęciem implementacji systemu wykonano konfigurację z wykorzystaniem karty microSD o pojemności 128 GB charakteryzującą się prędkością odczytu do 130 [MB/s] oraz prędkością zapisu do 90 [MB/s] w celu płynnej pracy systemu. Wykonano następujące operacje:

- instalacja oprogramowania SD Card Formatter [33],
- formatowanie karty pamięci microSD przy użyciu oprogramowania SD Card Formatter,
- pobranie obrazu systemu udostępnionego w projekcie Jetbot AI,
- instalacja oprogramowania balenaEthcer [34],
- zapisanie systemu na karcie pamięci z wykorzystaniem programu balenaEtcher.

Wykonanie ww. czynności spowodowało wyposażenie układu Jetson Nano w system operacyjny Linux Ubuntu 20.04. Następnie zainstalowano biblioteki:

- Nvidia cuDNN (wersja 8.3),
- Nvidia CUDA (wersja 11.2),
- Nvidia TensorRT (wersja 8.4),

Oraz zaktualizowano zmienne systemowe biblioteki CUDA. Po wykonaniu tych czynności oraz przebudowie kodu sterowania silnikami opisanymi w podrozdziale 6.3 zaimplementowano wytrenowane modele sieci neuronowych. Po wykonaniu tych czynności możliwe stało się przeprowadzenie badań weryfikacyjnych. Praca systemu w różnych warunkach została przedstawiona na rys. 8.2 wraz z wartością klasyfikacji zajętości drogi.



Rys. 8.2. Przykładowe zdjęcia z widoku pracy robota warunkach od lewej – a) wewnętrznych dziennych ; b) zewnętrznych nocnych; c) zewnętrznych dziennych

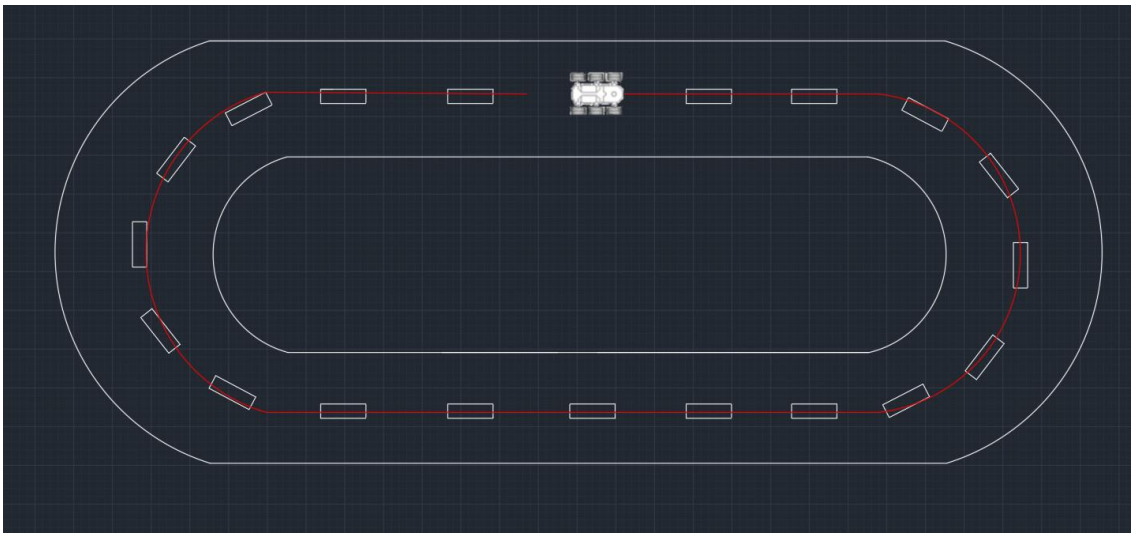
8.2. Plan badań

W ramach realizacji pracy dyplomowej magisterskiej przeprowadzono szereg badań weryfikacyjnych zaimplementowanego systemu. Celem zaplanowanych testów stało się określenie poprawności działania systemu po zaimplementowaniu na platformie robotycznej modeli charakteryzujących się najlepszymi wartościami wytrenowania. Zbadano zdolność operowania systemu w założonych warunkach dziennych oraz nocnych na utworzonych torach testowych tworząc cztery scenariusze (tryb nocny i dzienny na torze w warunkach wewnętrznych oraz zewnętrznych). Celem badań było sprawdzenie funkcjonalności systemu sterowania autonomicznego w warunkach utworzonych torów testowych. Badania weryfikacyjne podzielono na 3 etapy. Pierwsza część to badania cząstkowe dotyczące weryfikacji poprawności działania wszystkich podzespół utworzonej platformy robotycznej poprzez przejazd po wyznaczonym torze niezawierającym przeszkód. Badania zasadnicze obejmowały natomiast drugą oraz trzecią część prowadzonych badań. Drugi etap obejmował przejazd po za planowej trasie na torze testowym w warunkach wewnętrznych. Trzeci etap natomiast dotyczył pracy systemu na torze testowym w warunkach zewnętrznych.

8.3. Badania cząstkowe

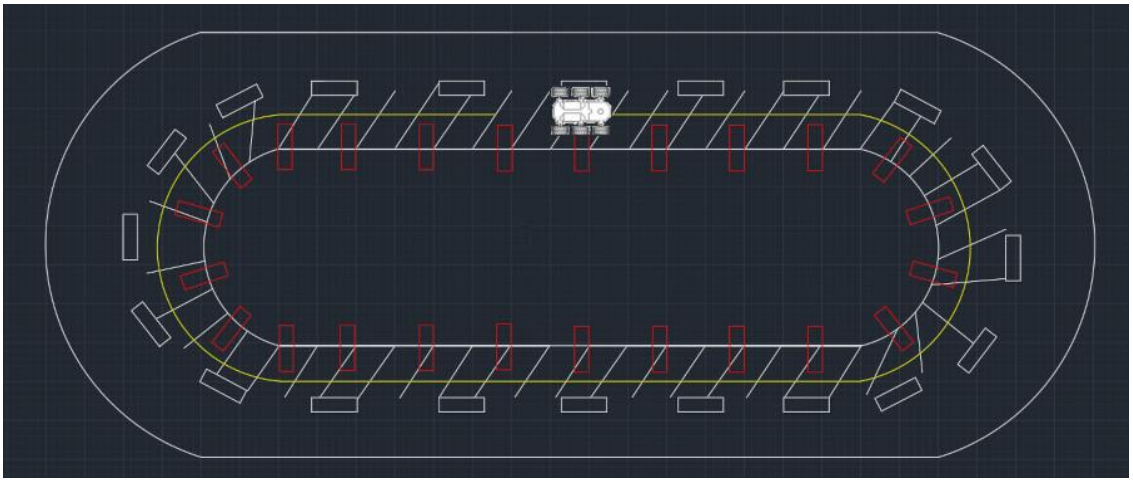
W ramach badań cząstkowych przeprowadzono próbę pokonania przygotowanego toru testowego w warunkach wewnętrznych w początkowej fazie pracy, co zostało opisane w podrozdziale 5.1. Schemat stanowiska badawczego został przedstawiony na rys 8.3. Badania rozpoczęto od przejazdu po torze przy użyciu sterowania poprzez zewnętrzny kontroler. Przetestowano poprawność działania systemu oraz wszystkich komponentów, np. kierunku pracy napędów w sześciu kołach oraz wszystkich urządzeń peryferyjnych.

W drugim scenariuszu zbadano przejazd przy wykorzystaniu zaimplementowanego przykładu w oprogramowaniu Nvidii o nazwie *road following* w celu sprawdzenia zachowania robota eksploracyjnego w pracy z modelem sieci neuronowej. Elementy brzegowe służyły w trakcie zbierania danych uczących jako obszar graniczny wychylenia robota. Środkowe pasy pośrodku przyjęto jako punkt odniesienie prostoliniowej trajektorii ruchu robota. Kolorem czerwonym zaznaczono planowaną trajektorię przejazdu robota.



Rys. 8.3. Schemat stanowiska badawczego utworzonego do badań cząstkowych dla pierwszego scenariusza

Następnie dodano kontrastujące czerwone pręgi na wewnętrznym obrzeżu utworzonego toru i zebrano nowe dane uczące. Przetestowano działanie algorytmu ograniczając punkt orientacyjny robota do jednej przerywanej linii a istniejące obrzeża zewnętrze i środkową przerywaną linię potraktowano jako strefę graniczną będącą zakłóceniem poprawnej jazdy, przez co ograniczono jazdę robotę do ściśle określonego obszaru zaznaczonego na rys 8.4. Kolorem żółtym zaznaczono planowaną trajektorię przejazdu robota.



Rys 8.4. Schemat stanowiska badawczego utworzonego do badań cząstkowych dla drugiego scenariusza

Do przeprowadzenia testów przyjęto określoną liczbę 20 przejazdów. Wartość tą wybrano ponieważ pozawala ona na obserwację trendu zachowania robota, a także jest optymalna czasowo. Parametr skuteczność określa procentową liczbę poprawnych przejazdów w porywaniu z przejazdami zakończonymi zagubieniem trasy. Badania cząstkowe ukazały poprawność pracy robota eksploracyjnego wraz z wszystkimi urządzeniami peryferyjnymi odpowiednio reagującego na zadane sygnały sterujące poprzez zewnętrzy kontroler.

Pierwszy scenariusz testu robot pokonywał bardzo dobrze z 90 % skutecznością, gdyż na 20 przeprowadzonych przejazdów dwukrotnie wyjechał poza tor tracąc bezpowrotnie drogę. W drugim scenariuszu robot zachowywał się poprawnie, gdyż jego skuteczność przejazdu wynosiła 65 %. Warto jednak zaznaczyć że obszar pracy był bardzo ograniczony będący równy szerokości robota. Wskazany kierunek nawigacyjny wzdłuż jednej założonej trasy był trudny do zidentyfikowania z powodu występującej przerywanej linii czerwonych pasków identycznej z dwóch stron (jedna poprawna, druga poza wyznaczonym obszarem pracy). Skutkowało to częstością zagubienia drogi w przypadku wykonania korekty trajektorii ruchu przez robota w kierunku wewnętrznym gdyż następował brak widocznych znaków w obrazie kamery robota będącego w środku toru testowego. Uzyskane wyniku testów cząstkowych zaprezentowano w tabeli poniżej.

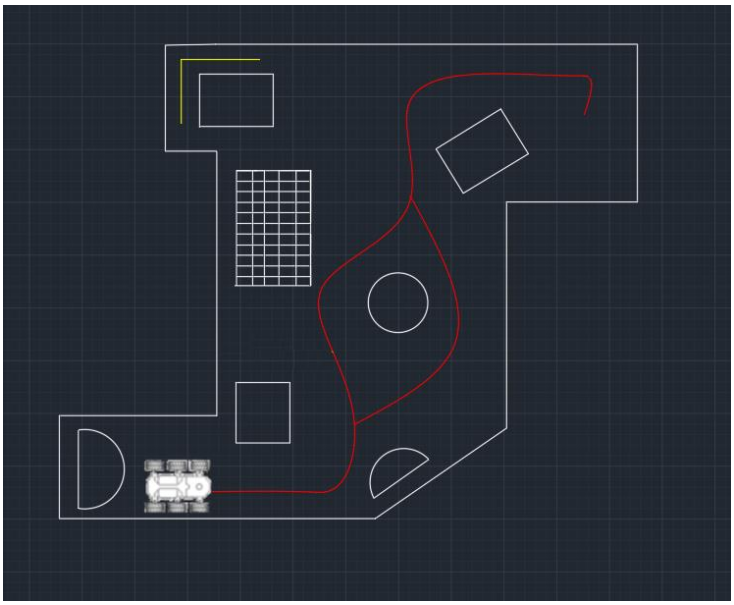
Numer scenariusza	**Liczba przejazdów [liczba]**	**Poprawne przejazdy [liczba]**	**Zagubienie trasy [liczba]**	**Skuteczność [%]**
I	20	18	2	90
II	20	11	9	65

Tab 3.1. Tabela pomiarowa testów przeprowadzonych w badaniach cząstkowych

8.4. Badania zasadnicze

W badaniach zasadniczych przeprowadzonych na torze testowym w warunkach wewnętrznych dodano elementy o kształcie wielościanów (tj. graniastosłupy i ostrosłupy) w kolorze białym co przedstawione zostało na rys 8.5 jako elementy bez wypełnienia. Kolor ten wybrano w związku z zjawiskiem odbijaniem światła z diod LED zamontowanych w robocie eksploracyjnym Orzeł 7, co znacząco poprawiało warunki

eksploracji nocnej. Wypełnienie siatką zastosowano do elementu o ciemnym kolorze specjalnie umieszczonego w celu przeprowadzenia badań nad rozpoznaniem takowego elementu w warunkach nocnych. Trajektorię przejazdu zaznaczono na rys 8.5. Kolorem czerwonym zaznaczono planowaną trajektorię przejazdu robota.



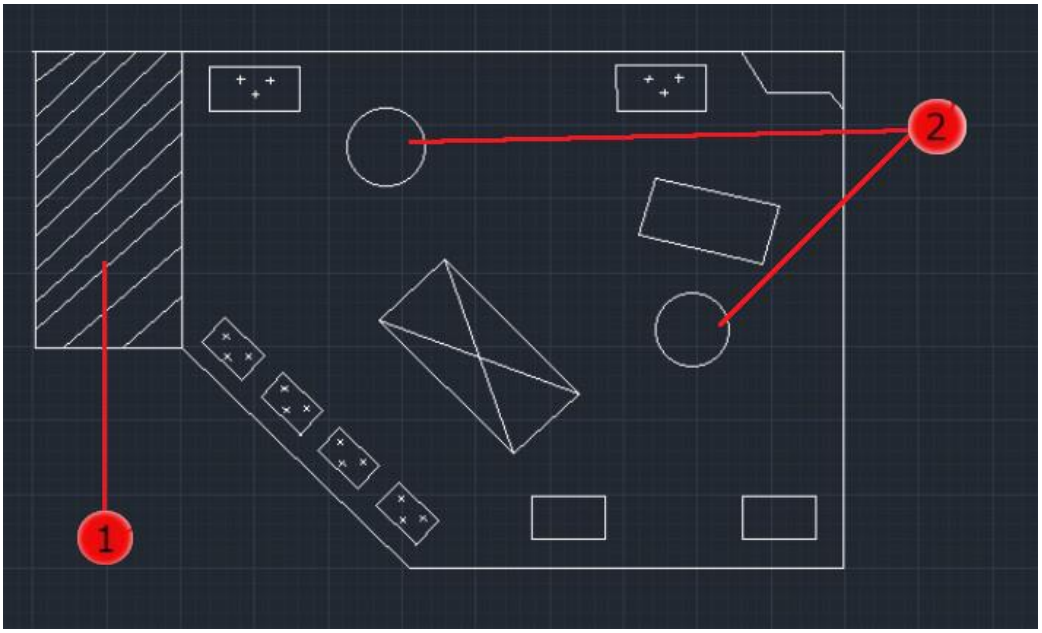
Rys 8.5. Schemat toru testowego w warunkach wewnętrznych

W trakcie przeprowadzonych badań w warunkach dziennych uzyskano skuteczność pokonania zaplanowanej trasy na poziomie 95 [%]. Trasa przejazdu została tak skonfigurowana, że robot mógł wybrać dowolny kierunek pokonywania przeszkody o kształcie cylindrycznym. Realizowane przejazdy w warunkach nocnych odznaczyły się mniejszą skutecznością. Robot Orzeł 7 trzykrotnie nie dotarł do końca trasy ponieważ algorytm sterowania spowodował zawrócenie w celu ominięcia przeszkody i nie powrócił do planowanej trajektorii. W jednym przypadku robot został unieruchomiony za przeszkodzą, a to miejsce zostało oznaczone na schemacie kolorem żółtym. Uzyskane wyniki testów na torze wewnętrznym w warunkach dziennych i nocnych przedstawiono w tabeli poniżej.

Warunki scenariusza	Liczba przejazdów [liczba]	Poprawne przejazdy [liczba]	Zagubienie trasy [liczba]	Skuteczność [%]
Dzień	20	19	1	95
Noc	20	16	4	80

Tab 3.1. Tabela pomiarowa testów na torze w warunkach wewnętnych przeprowadzonych w badaniach zasadniczych

W drugiej części badań zasadniczych przeprowadzono testy na torze w warunkach zewnętrznych (rys 8.6). Przyjęty scenariusz określał eksplorację robota po pełnym obszarze toru w najkrótszym czasie. Przetestowane zostały wszystkie aspekty platformy robotycznej zarówno pod względem konstrukcji, a także systemu sterowania. Newralgicznymi punktami stały się obszar zagłębienia w postaci krateru (rys. 8.6., poz.1). oraz profilowane kamienie (rys. 8.6., poz.2). Elementy te posiadały możliwość przejazdu tylko z określonego kierunku w celu ich pokonania. W przypadku kamieni zbiór danych uczących został dokładnie przygotowany w kontekście perspektywy pokonania tej przeszkody, więc sytuacja utknięcie na niej pojawiła się tylko jednokrotnie.



Rys 8.6. Schemat toru testowego w warunkach zewnętrznych

Przejazd w warunkach dziennych odbywał się średnio w czasie 6 min i 17 sekund. Nastąpiło jednokrotne utknięcie robota na profilowanym kamieniu oraz dwukrotne zatrzymanie robota w kraterze. W warunkach nocnych natomiast robot czterokrotnie nie potrafił pokonać przeszkody krateru. Dwukrotnie problemy wystąpiły w wyniku przejazdu przez profilowanych kamieni gdyż robot eksploracyjny Orzeł 7 zawisł na zawieszeniu nie potrafiąc wykonać żadnego ruchu. Uzyskane wyniku testów zasadniczych w warunkach zewnętrznych zaprezentowano w tabeli poniżej. Akceptacja czasu przejazdu robota eksploracyjnego po torze w warunkach zewnętrznych brana pod uwagę do obliczania średniego czasu jazdy następowała po spełnieniu kryterium tj. pokonanie wzniesienia, krateru oraz jednego z profilowanych kamieni.

Warunki scenariusza	**Liczba przejazdów [liczba]**	**Poprawne przejazdy [liczba]**	**Zagubienie trasy [liczba]**	**Skuteczność [%]**	**Średni czas przejazdu [min]**
Dzień	20	17	3	85	6:17
Noc	20	13	7	65	7:24

Tab 3.3. Tabela pomiarowa testów na torze w warunkach zewnętrznych przeprowadzonych w badaniach zasadniczych

Jak można zaobserwować na zamieszczonych tabelach najskuteczniejsze okazały się modele wyuczone na danych w warunkach dziennych. Uzyskane wyniki są zgodne z oczekiwaniami, gdyż otrzymane wartości w trakcie nauki modelów charakteryzują się najwyższą dokładnością oraz najniższym poziomem straty. Na kolizje, do których doszło w trakcie przeprowadzania badań bardzo duży wpływ miały dostępne warunki środowiskowe do przygotowania torów testowych. Co warto zaznaczyć miejsce zagłębienia jest terenem trudnym do eksploracji z względu na stromiznę zbocza utworzoną w ramach pracy magisterskiej a także konstrukcję robota związaną z małą szerokością podwozia oraz dużą frakcją skał podłoża. Wyniki czasowe uzyskane w przejazdach po torach w warunkach wewnętrznych są proporcjonalnie lepsze z względu na mniejszą powierzchnię toru oraz mniejszy stopień skomplikowania lecz przede wszystkim poprzez przyjęte odmienne scenariusze przejazdów.

8.5. Analiza wyników

Przeprowadzone badania umożliwiły weryfikację poprawności działania utworzonego systemu sterowania. Wykonane badania cząstkowe pozwoliły na weryfikację konstrukcyjną utworzonej platformy robotycznej oraz zastosowanych urządzeń peryferyjnych. Zaimplementowano podstawowe przykłady sterowania przy wykorzystaniu modeli głębokich sieci neuronowych. Działania te pozwoliły na znacznie szybszą pracę przy implementacji właściwych algorytmów sterowania. Badania zasadnicze zweryfikowały natomiast poprawność utworzonych zbiorów danych oraz wyuczonych modeli. Pozwoliły ocenić również czy utworzony system sterowania bazujący na głębokich sieciach neuronowych spełnia przyjęte założenia. Sprawdzono, że dodany ciemny element na torze testowym w warunkach wewnętrznych z scenariuszem nocnym został odpowiednio sklasyfikowany oraz omijany pomimo uzyskanych najgorszych wyników nauki spośród wszystkich zestawów danych. Wszystkie przeprowadzone badania potwierdziły, że system działa poprawne. Wybrane technologie oraz wytyczne umożliwiły wykonanie systemu sterowania zgodnie z oczekiwaniami. Badania zasadnicze dały informację zwrotną w kwestii rozbudowania zbioru danych uczących o miejsca newralgiczne, w których robot eksploracyjny tracił trasę i przejazd został przerywany. W trakcie badań zasadniczych zweryfikowano, że ciemny element umieszczony na torze testowym w warunkach wewnętrznych w scenariuszu nocnym został poprawnie klasyfikowany i algorytm odpowiednio podejmował działania powodujące jego ominięcie.

Rozdział 9

9. Podsumowanie i wnioski

W tym rozdziale zostały sformułowane wnioski powstałe podczas realizacji pracy dyplomowej magisterskiej oraz przedstawiono możliwości rozwoju w kontekście perspektyw prowadzenia dalszych badań nad tematyką poruszaną w niniejszym projekcie.

9.1. Podsumowanie

Niniejsza praca dokumentuje proces opracowania i implementacji systemu sterowania robotem eksploracyjnym bazujący na głębokich sieciach neuronowych. Przedstawione zostały wyniki analizy literatury w zakresie dostępnych platform robotycznych w zakresie rozwoju systemów sterowania bazujących na głębokich sieciach neuronowych oraz konwolucyjnych głębokich sieci neuronowych w kontekście systemów sterowania poprzez wykorzystanie obrazu z kamer. Przeprowadzone zostały modyfikacje obiektu sterowania – platformy robota eksploracyjnego Orzeł 7. System utworzony został z wykorzystaniem obrazu systemu projektu Jetbot AI przebudowanego w środowisku programistycznym JupyterLab przy wykorzystaniu języka Python oraz biblioteki PyTorch. Zbiór danych uczących powstał w oparciu o utworzone zdjęcia z przygotowanych w ramach pracy magisterskiej torów testowych. Proces uczenia przeprowadzony został na komputerze klasy laptop wyposażonym w układ GPU dedykowany procesom obliczeniowym. Wyuczony model sieci neuronowej oparty na architekturze AlexNet bądź ResNet zaimplementowany został na układzie Jetson Nano będącym jednostką sterującą utworzonej platformy robotycznej do nauki i rozwoju autonomicznych systemów sterowania w kontekście robotów eksploracyjnych. W trakcie badań weryfikacyjnych oceniono sprawność i użyteczność systemu w różnych warunkach testowych.

9.2. Wnioski

Podczas realizacji pracy magisterskiej nasunęły się następujące wnioski:

1. System sterowania działa zgodnie z przedstawionymi założeniami.
2. System może być stale udoskonalany poprzez wykorzystanie bardziej złożonych architektur sieci, dostrajanie hiperparametrów oraz zwiększenie zbioru danych uczących.
3. Zastosowanie dobrze wytrenowanego modelu przy wykorzystaniu głębokich sieci neuronowych umożliwia przejazd robota po złożonym torze bez interwencji operatora.
4. Konieczny jest ciągły nadzór operatora nad robotem eksploracyjnym z uwagi na występującą możliwość utknięcia robota w newralgicznych miejscach. Warto zaznaczyć, że nadzór jest konieczny na obecnym poziomie zawansowania prac. Docelowo łazik może być autonomiczny.
5. Największy wpływ na skuteczność nauki modelu sieci neuronowej ma zbiór danych uczących. Więcej przykładów uczących oraz lepsze określenie elementów newralgicznych wpływa na większą skuteczność działania systemu.
6. Proces nauki sieci neuronowej jest procesem czaso- oraz obliczeniochłonnym. Wymaga posiadania odpowiedniego sprzętu wyposażonego w układ graficzny, gdyż proces trenowania na jednostce graficznej jest dużo szybszym w porównaniu do wykonywania obliczeń na procesorze.
7. Zestaw deweloperski Jetson Nano A02 4GB pozwala na komfortową pracę z systemami sterowania opartymi o sieci neuronowe. Posiada moc obliczeniową wystarczającą do przeprowadzenia trenowania na niewielkiej liczbie epok. Należy jednak zastosować dodatkowe chłodzenie w postaci wentylatora wspomagającego radiator, gdyż procesor graficzny osiąga wysoką temperaturę podczas procesu nauki.
8. Transfer wiedzy umożliwił szybkie douczanie sieci z wykorzystaniem niezbyt licznego zbioru danych uczących.
9. Projekt Jetbot AI jest dobrą platformą typu open-source dla nauki sieci neuronowych z dobrze przygotowaną dokumentacją oraz wsparciem społeczności.
10. Brak jest na rynku platformy robotycznej do rozwoju systemów sterowania opartej na sieciach neuronowych o konstrukcji sześciokołowego robota eksploracyjnego.
11. Warunki oświetleniowe mają kluczowy wpływa na skuteczność działania modelu.

12. Platforma stanowi odpowiednie miejsce do rozbudowy, testowania i implementacji nowych modeli oraz różnych architektur sieci neuronowych.
13. Praca Jetsona jest optymalna przy przetwarzaniu obrazu podczas posiadania odpowiednio wydajnego źródła zasilania.

9.3. Kierunki rozwoju

W trakcie realizacji pracy magisterskiej zwrócono także uwagę na dalsze możliwości kierunku prac badawczych:

- Przeprowadzenie badań z wykorzystaniem różnych typów kamer. Przykładem może być wykorzystanie kamery stereowizyjnej Stereolabs ZED 2 [72]. Posiada ona zaimplementowane algorytmy oraz wykonuje wewnętrznie procesy obliczeniowe przetwarzania obrazu. Możliwa jest integracja algorytmu śledzenia pozycji udostępnianego przez firmę StereoLabs (dostosowany do śledzenia sylwetki człowieka) z rozbudową utworzonego w ramach pracy magisterskiej modelem klasyfikacji zajętości drogi. Implementacja takiego sytemu możliwa by była na odpowiednio przeskalowanej utworzonej platformie robota eksploracyjnego charakteryzującej się dobrymi właściwościami terenowymi. System ten możliwy by był do wykorzystania w warunkach wojskowych jako wspomaganie żołnierza piechoty dla noszenia za nim ekwipunku co jest istotne w dobie obecnej sytuacji geopolitycznej.
- Zastosowanie inne architektury sieci neuronowych np. bardziej rozbudowane wersje ResNet z 34,50,101 warstwami splotowymi lub wydajnej sieci MobielNet będącą kompromisem między wydajnością a niskimi zasobami obliczeniowymi.
- Rozbudowanie zbioru danych uczących o zdjęcia wykonanie w miejscach newralgicznych zidentyfikowanych w ramach badan zasadniczych tj. karter oraz profilowane kamienie.

Wykonane badania pozwoliły na przygotowanie publikacji pt. „Dydaktyczny robot eksploracyjny z systemem omijania przeszkód”. Publikacja ta zakwalifikowała się do prezentacji w ramach studenckiej konferencji Metody Komputerowe 2022 [73]. Publikacja została opublikowana w materiałach konferencyjnych, a także podczas konferencji został wygłoszony referat przez autora niniejszej pracy.

Literatura

[1] I. Goodfellow T. Bengio A. Courville *Deep Learning Systemy uczące się* PWN 2018.

[2] T.S. Huang *Computer Vision: Evolution and Promise* University of Ilinois at Urbana-Champaign.

[3] W. Dobrosielski. *Zastosowanie sztucznych sieci neuronowych do rozpoznawania obrazów*, 2010.

[4] K. Krawiec *Segmentacja obrazu, Przetwarzanie i Rozpoznawanie Obrazów* Instytut Politechniki Poznańskiej.

[5] R. Tadeusiewicz, B. Borowik, T. Gąciarz,. *„Odkrywanie właściwości sieci neuronowych przy użyciu programów w języku C#”* , Polska Akademia Umiejętności.

[6] M. Szuster Z. Hendzel. *Sieci neuronowe i systemy rozmyte*, Rzeszów 2010.

[7] S.Weidman *Sztuczna inteligencja. Uczenie głębokie od zera – Podstawy implementacji w Pythonie,* Helion 2020.

[8] R.Zadeh, B.Ramsundar *TensorFlow for Deep Learning*, O’Reilly 2021.

[9] P.Polnik *Koncepcja systemu sterowania robotem eksploracyjnym.* Praca przejściowa. 2022.

[10] Oficjalna strona zawodów Europen Rover Chalenge (ERC) - https://roverchallenge.eu/zawody-robotyczne-erc/

[11] Oficjalna strona projektu Mars Curiosity Rover https://mars.nasa.gov/msl/home/

[12] Dokumentacja techniczna firmy KORAD *Power Supply Specification Sheet KORAD Single Chanel 90W-150W*

[13] Dokumentacja techniczna firmy NVIDIA *DATA SHEET NVIDIA Jetson Nano System-on-Module Maxwell GPU + ARM Cortex-A57 + 4GB LPDDR4 + 16GB eMMC*

[14] Dokumentacja techniczna firmy INIU *INIU Portable Charrger Power Bank BI B41*

[15] Strona internetowa produktu FLUKE 87V https://www.fluke.com/pl-pl/produkt/testowanie-instalacji-elektrycznych/multimetry-cyfrowe/fluke-87v [Online; dostęp 30.08.2022].

[16] Strona interetowa produktu AKYGA AK-DC-01 https://pl.akyga.com/produkty/311-kabel-usb-a-dc-5-5-x-2-1mm-ak-dc-01.html [Online; dostęp 29.07.2022].

[17] Oficjalna strona oprogramowania PuTTY https://www.chiark.greenend.org.uk/~sgtatham/putty/ [Online; dostęp 30.08.2022].

[18] Rezpozytorium GitHub - projekt R. Bonghi – Jetson Stats https://github.com/rbonghi/jetson_stats [Online; dostęp 19.07.2022].

[19] Strona internetowa oprogramowania Autodek Fusion 360 https://www.autodesk.pl/products/fusion-360/overview?term=1-YEAR&tab=subscription [Online; dostęp 25.08.2022].

[20] Strona internetowa Prusa Slicer https://www.prusa3d.com/pl/strona/prusaslicer_424/ [Online; dostęp 20.07.2022].

[21] M.Prabhu *CNN Architectures — LeNet, AlexNet, VGG, GoogLeNet and ResNet*, Medium 2018.

[22] Fully Connected Layers in Convolutional Neural Networks: The Complete Guide.

[23] A.Horzyk Wykłady z kursu *Sztuczna inteligencja i inteligencja obliczeniowa.*

[24] A.Kwasigroch, M.Grochowski. Rozpoznawanie obiektów przez głębokie sieci neurnowe.

[25] Y.LeCun, Y.; Boser, B.; Denker, J. S.; Henderson, D.; Howard, R. E.; Hubbard, W.; Jackel, L. D. (December 1989). "Backpropagation Applied to Handwritten Zip Code Recognition". Neural Computation.

[26] A.Krizhevsky, I.Sutskever, G. Hinton *ImageNet Classification with Deep Convolutional Neural Networks.*

[27] K.Piszczak *Klasyfikacja dźwięku za pomocą splotowych sieci neuronowych* Rozprawa doktorska, 2018.

[28] K.He, X.Zhang, S.Ren, J.Sun *Deep Residual Learning for Image Recognition – 2015.*

[29] Oficjalna strona biblioteki PyTorch https://pytorch.org/ [Online; dostęp 26.07.2022].

[30] K.He, X.Zhang, S.Ren, J.Sun *Deep Residual Learning for Image Recognition – 2016.*

[31] P.Nepal *VGGNet Architecture Explained.*

[32] K.He, X.Zhang, S.Ren, J.Sun *Deep Residual Learning for Image Recognition – 2016b.*

[33] Strona internetowa oprogramowania SD Card Formatter. https://www.sdcard.org/downloads/formatter/eula-windows/ [Online; dostęp 20.08.2022].

[34] Strona internetowa oprogramowania balenaetcher. https://www.balena.io/etcher/, [Online; dostęp 22.08.2022].

[35] Strona internetowa oprogramowania Ubuntu 20.04 https://releases.ubuntu.com/20.04/ [Online; dostęp 24.08.2022].

[36] Karta techniczna produktu FormLabs Flexible https://cadxpert.pl/wp-content/uploads/2020/07/2001419-SDS-PL-0.pdf [Online; dostęp 28.08.2022].

[37] Karta techniczna produktu Elgoo ABS-Like https://drive.google.com/file/d/1_suQfC-jo10O4AdqDMZ9Z0ThpgQsitJC/view [Online; dostęp 29.08.2022].

[38] Karta techniczna produktu Formlabs Form3 https://formlabs.com/3d-printers/form-2/tech-specs/. [Online; dostęp25.08.2022].

[39] McCulloch, W. S. & W. Pitts *A Logical Calculus of the Ideas Immanent in Nervous Activity*. The Bulletin of Mathematical Biophysics, 1943.

[40] F. Rosenblatt *The Perceptron, a Perceiving and Recognizing Automaton. Cornell Aeronautical Laboratory*. 1957.

[41] M.Minsky & S. Papert. *Perceptrons*. MIT. 1969.

[42] K.Shen *Effect of batch size on training dynamics,* Medium 2018.

[43] B.Brian *Finding the ideal num_workers for Pytorch Dataloaders* 2020.

[44] Dokumentacja techniczna firmy Nvidia *CUDA Toolkit Documentation v11.7.1*.

[45] J.Brownlee *How to Configure the Learning Rate When Training Deep Learning Neural Networks* https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/ [Online; dostęp 12.07.2022].

[46] Adams, James Lowell, *Remote control with long transmission delays,* PhD in Mechanical Engineering, 1961.

[47] E.Dickmanns, *Dynamic Vision for Perception and Control of Motion*, SPRINGER VERLAG GMBH 2007 .

[48] *Navlab: The Carnegie Mellon University Navigation Laboratory* *https://www.cs.cmu.edu/afs/cs/project/alv/www/index.html* [Online; dostęp 06.07.2022].

[49] Oficjalna strona Agencja Zaawansowanych Projektów Badawczych w Obszarze Obronności – DARPA https://www.darpa.mil/ [Online; dostęp 07.07.2022].

[50] M.Buehler, K.Lagnemma, S.Singh *The 2005 DARPA Grand Challenge: The Great Robot*, Springer 2007.

[51] Oficjalna strona projektu autopilota firmy Tesla *Autopilot and Full Self-Driving Capability* *https://www.tesla.com/support/autopilot* [Online; dostęp 09.07.2022].

[52] Oficjalna strona firmy Caterpillar https://www.cat.com/en_AU/news/machine-press-releases/cat-autonomous-mining-trucks-haul-one-billion-tonnes.html [Online; dostęp 14.07.2022].

[53] Strona łazika Perversence https://mars.nasa.gov/mars2020/ [Online; dostępne dn. 11.07.2022].

[54] Strona łazika Curiosity https://mars.nasa.gov/msl/home/ [Online; dostępne dn. 11.07.2022].

[55] Strona projektu „Marsowe analogi do eksploracji kosmosu" https://cordis.europa.eu/article/id/229925-detecting-mars-ancient-life-forms-is-possible-although-more-difficult-than-first-thought-rese [Online; dostępne dn. 19.07.2022].

[56] NASA - Cele misji z Księżyca do Marsa https://www.nasa.gov/sites/default/files/atoms/files/moon-to-mars-objectives-.pdf [dostępne dn. 19.07.2022].

[57] C.Clifford *There Will Be 20 Million Self-Driving Cars On the Road by 2025* [Online; dostępne dn. 20.07.2022].

[58] Oficjalna strona projektu F1TENTH https://f1tenth.org/ [Online; dostępne dn. 20.07.2022].

[59] Oficjalna strona kursu F1TENTH CourseKit https://f1tenth-coursekit.readthedocs.io/en/latest/ [Online; dostępne dn. 20.07.2022].

[60] Oficjalna strona projektu AutoRally https://autorally.github.io/ [Online; dostępne dn. 20.07.2022].

[61] Biblioteka GitHub projektu AutoRally https://github.com/AutoRally/autorally [Online; dostępne dn. 20.07.2022].

[62] Dokumentacja platformy Duckiebot MOOC Founder's https://docs.rs-online.com/c57a/A700000007343652.pdf [Online; dostępne dn. 02.08.2022].

[63] Oficjalna strona internetowa projektu Dockietown https://www.duckietown.org/# [Online; dostępne dn. 02.08.2022].

[64] Oficjalna strona internetowa Clearpath Jackal https://clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/ [Online; dostępne dn. 02.08.2022].

[65] Strona internetowa języku programowania Python w wersji 3.9 https://www.python.org/downloads/release/python-390/ [Online; dostępne dn. 08.08.2022].

[66] Strona internetowa oprogramowania Jupyter. https://jupyter.org/, [Online; dostępne dn. 12.08.2022].

[67] Dokumentacja projektu JetBot AI https://github.com/NVIDIA-AI-IOT/jetbot/ [Online; dostępne dn. 12.08.2022].

[68] Strona internetowa oprogramowania Linux Ubuntu w wersji 20.4. https://releases.ubuntu.com/focal/ [Online; dostępne dn. 12.08.2022].

[69] Dokumentacja Adafruit Stepper + DC Motor FeatherWing https://cdn-learn.adafruit.com/downloads/pdf/adafruit-stepper-dc-motor-featherwing.pdf [Online; dostępne dn. 12.08.2022].

[70] Strona internetowa oprogramowania Docker https://www.docker.com/ [Online; dostępne dn. 21.08.2022].

[71] Strona internetowa oprogramowania Anaconda. https://www.anaconda.com/, [Online; dostępne dn. 25.08.2022].

[72] Strona internetowa produktu SteroLabs Zed 2. https://www.stereolabs.com/zed-2/, [Online; dostępne dn. 17.08.2022].

[73] P.Polnik *Dydaktyczny robot eksploracyjny z systemem omijania przeszkód*. Studencka Konferencja Naukowa „Metody Komputerowe 2022", Gliwice 2022.

[74] Strona internetowa produktu PCA9685 TB6612 DUAL – HAITRONIC https://www.fabian.com.mt/en/products/webshop/18847/stepper-motor-driver-pca9685-tb6612-dual---haitronic.htm [Online; dostępne dn. 30.08.2022].

[75] K. He, X. Zhang, S. Ren i J. Sun. „*Deep Residual Learning for Image Recognition*". W: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016.

[76] Y.Ozaki, M. Yano, M, Onis. *Effective hyperparameter optimization using Nelder-Mead method in deep learning.*

[77] J.Anderson, N.Kalra, P.Sorensen *„Autonomous Vehicle Technology",* Rand Comrporation.

[78] Article „*Hisotry of Autonomous Cars" https://www.tomorrowsworldtoday.com/2021/08/09/history-of-autonomous-cars/* [Online; dostępne dn. 24.08.2022].

[79] J.Waldschmidt *"Cat's Autonomous Haul Truck New Record"* https://www.equipmentworld.com/equipment/article/15290173/cats-autonomous-trucks-haul-1-billion-tons-in-less-than-a-year [Online; dostępne dn. 27.08.2022].

[80] B.Moce *„Autonomous vehicles were supposed to push gas-powered cars off the road in the future. Will we ever get there?"* [Online; dostępne dn. 02.09.2022].

[81] J.Ocón, E.Rivero, A.Sanchez *„Multi-agent Frameworks for Space Applications".*

[82] Article „*ALEX Net: ImageNet Classification with Deep Convolutional Neural Network"* https://www.google.com/search?q=ALEX+Net%3A+ImageNet+Classification+with+Deep+Convolutional+Neural+Network&sourceid=chrome&ie=UTF-8 [Online; dostępne dn. 04.09.2022].

[83] Y.Bengio, J.Bergstra *„Random Search for Hyper-Parameter Optimation"*, Univeristy of Montreal, 2012.

[84] Oficjalna strona produktu Elgoo Sasturn https://www.elegoo.com/products/elegoo-saturn-4k-mono-lcd-3d-printer [Online; dostępne dn. 08.09.2022].

Użyte oprogramowanie

Poniżej zamieszczono listę oprogramowania wykorzystanego w trakcie realizacji pracy dyplomowej magisterskiej:

- Microsoft Office Excel,
- Microsoft Office Word,
- GIMP v2.10.12,
- Microsoft Visual Studio Code v1.52.1,
- Linux Ubuntu 20.4,
- Project Jupyter (wersja 3.4.5),
- PyTorch (wersja 1.12.0),
- Anaconda (wersja 3),
- Python (wersja 3.9),
- Autodesk Fusion 360,
- Diagrams.net,
- Windows 10 Home.

Wszystkie wykorzystane programy posiadają darmowe wersje próbne, bezpłatne wersje dla studentów lub są całkowicie nieodpłatnymi narzędziami.

POLITECHNIKA ŚLĄSKA
WYDZIAŁ MECHANICZNY TECHNOLOGICZNY
KATEDRA PODSTAW KONSTRUKCJI MASZYN
Kierunek studiów: Automatyka i Robotyka

Tytuł: „SYSTEM STEROWANIA ROBOTEM EKSPLORACYJNYM BAZUJĄCY NA GŁĘBOKICH SIECIACH NEURONOWYCH”
Autor: inż. Paweł Polnik
Promotor: dr inż. Marcin Januszka

STRESZCZENIE

Słowa klucze: uczenie głębokie, sieci neuronowe, robot eksploracyjny, druk 3D

Niniejsza praca dyplomowa magisterska przedstawia system sterowania robotem eksploracyjnym. Celem pracy było opracowanie oraz implementacja systemu sterowania robotem eksploracyjnym bazującym na głębokich sieciach neuronowych. Uszczegółowiono koncepcję robota eksploracyjnego poprzez wprowadzenie modyfikacji opisanych w pracy magisterskiej. Utworzono tory testowe w warunkach wewnętrznych oraz zewnętrznych symulujący realia marsjańskie. Opracowano zestawy danych uczących zarówno w warunkach dziennych, jak i nocnych na obydwu torach. Wytrenowano modele oparte na architekturze AlexNet oraz ResNet oraz przeprowadzono badania nad optymalizacją hiperparamtrów. Przeprowadzone badania weryfikacyjne dowiodły poprawnego działania systemu sterowania robotem eksploracyjny. Pracę dyplomową magisterską zakończono sformułowaniem wniosków oraz określeniem perspektywy dalszego rozwoju.

SILESIAN UNIVERSITY OF TECHNOLOGY
FACULTY OF MECHANICAL ENGINEERING
DEPARMENT OF FUNDAMENTALS OF MACHINERY DESIGN
Field of study: Automatic Control and Robotics

Title: „CONTROL SYSTEM OF EXPLORATORY ROBOT BASED ON DEEP NEURAL NETWORKS”
Author: Paweł Polnik Eng.
Supervisor: Marcin Januszka PhD Eng.

ABSTRACT

Key words: Deep learning, neural networks, exploratory robot, 3D printing

This thesis presents an exploratory robot control system. The aim of the thesis was to develop and implement an exploratory robot control system based on deep neural networks. The concept of the exploratory robot was refined by introducing the modifications described in the thesis. Indoor and outdoor test tracks simulating Martian realities were created. Learning data sets were developed for both day and night conditions on both tracks. Models based on the AlexNet and ResNet architectures were trained and hyperparameter optimisation studies were carried out. The verification tests carried out proved the correct operation of the exploratory robot control system. The thesis concluded with the formulation of conclusions and the identification of a perspective for further development.