

ZACHODNIOPOMORSKI UNIWERSYTET TECHNOLOGICZNY W SZCZECINIE
WYDZIAŁ INFORMATYKI

mgr inż. Jan Rodziewicz-Bielewicz

**Rzadkie sieci splotowe w układach sterowania z wizualnym
sprzężeniem zwrotnym**

Rozprawa doktorska

Promotor: dr hab. inż. Marcin Korzeń, prof. ZUT

Szczecin 2025

Streszczenie

W rozprawie poruszono zagadnienie zastosowania rzadkich sieci splotowych w układach sterowania z wizualnym sprzężeniem zwrotnym. Nacisk w badaniach położono na dane rzadkie w postaci trójwymiarowej (chmury punktów oraz rzadkie wokselizacje), które są naturalnym sposobem reprezentacji w przypadku kamer głębi. Stworzono autorskie modele sieci splotowych, które wytrenowano w oparciu o samodzielnie zebrane dane, pochodzące z pomiarów wizyjnych i odczytów z robotów.

Modele te zostały wykorzystane w pętli wizualnego sprzężenia zwrotnego oraz zwerifikowano je w zadaniach sterowania robotami mobilnymi.

W ramach badań wykazano empirycznie, że uwzględnienie rzadkości danych przy budowaniu modeli sieci splotowych pozwala na znacznie szybsze trening i predykcję. Pokazano też, że odpowiedni dobór struktury sieci pozwala na uzyskanie wysokiej jakości predykcji w zadaniach lokalizacji i orientacji. Uzyskana jakość była porównywalna lub wyższa niż modeli referencyjnych – PointNet, DGCNN i Point Cloud Transformer. Wykazano również, że autorskie architektury są użyteczne w zadaniach śledzenia i sterowania robotami w pomieszczeniu zamkniętym.

słowa kluczowe: chmura punktów, widzenie maszynowe, sieci splotowe, rzadkie warstwy splotowe, wokselizacja, wizualne sprzężenie zwrotne

Abstract

The dissertation deals the application of sparse convolutional networks in control systems with visual feedback. The research focuses on sparse data in three-dimensional form (point clouds and sparse voxelizations), which are a natural representation when using depth cameras. Custom convolutional network models were developed and trained using self-collected data obtained from visual measurements and robots telemetry. These models were employed in the visual feedback loop and validated in mobile robot control tasks.

The research empirically demonstrated that accounting for data sparsity during the development of convolutional network models enables significantly faster training and prediction. It was also shown that an appropriate choice architecture of network allows for high-quality predictions in localization and orientation tasks. The achieved quality was comparable to or better than that of reference models – PointNet, DGCNN, and Point Cloud Transformer. Additionally, the custom architectures proved useful in tasks related to tracking and controlling robots in indoor environments.

keywords: point cloud, machine vision, convolutional neural networks, sparse convolutional layers, voxelization, visual feedback

Spis treści

Wprowadzenie	7
Cel i teza pracy	7
1 Układy sterowania z wizualnym sprzężeniem zwrotnym	15
1.1 Elementy toru pomiarowego	17
1.2 Sensory	18
1.2.1 Kamera głębi	18
1.2.2 Lidar	19
1.2.3 Kamera zdarzeń	19
1.3 Transformacje geometryczne w 3D	20
2 Wykorzystanie sieci głębokich do pracy z chmurami punktów .	25
2.1 Dane w wizyjnych systemach pomiarowych	25
2.1.1 Obrazy RGB	25
2.1.2 Obrazy rzadkie. Chmury punktów	25
2.1.3 Zastosowanie chmur punktów w sterowaniu eye-to-hand	26
2.2 Praca z chmurami punktów	28
2.2.1 Metody wielowidokowe	28
2.2.2 Metody wolumetryczne	28
2.2.3 Metody pracujące bezpośrednio na chmurze punktów	31
2.2.4 Metody oparte o hybrydową reprezentację	33
2.3 Omówienie wybranych architektur	33
2.3.1 PointNet	34
2.3.2 Dynamic Graph CNN	34
2.3.3 Point Cloud Transformer	36
3 Architektury sieci dedykowane predykcji lokalizacji i orientacji .	37
3.1 LineCNN2D	38
3.2 SlicingCNN3D	40
3.3 SimpleCNN3D	42
3.4 DroneCropCNN3D	42

3.5	ProjectionCNN3D	44
4	Wyniki badań eksperymentalnych	47
4.1	Środowisko badawcze	47
4.2	Hipotezy badawcze	48
4.3	Przygotowanie eksperymentów	49
4.4	Badania wstępne – dane dwuwymiarowe Line2D	51
4.4.1	Sformułowanie problemu i zbiór danych	51
4.4.2	Wykorzystany model	52
4.4.3	Wyniki	52
4.5	Badania wstępne – dane trójwymiarowe Plane3D	55
4.5.1	Sformułowanie problemu i generowanie danych	55
4.5.2	Wykorzystany model	56
4.5.3	Wyniki	56
4.6	Sterowanie robotami mobilnymi iRobot 650	59
4.6.1	Stanowisko badawcze	59
4.6.2	Charakterystyka sterowanego obiektu iRobot 650	60
4.6.3	Przygotowanie zbioru danych	62
4.6.4	Wykorzystany model sieci spłotowej	62
4.6.5	Sterowanie	63
4.6.6	Wyniki	64
4.7	Śledzenie ramienia robota uArm Swift Pro	68
4.7.1	Stanowisko badawcze	68
4.7.2	Przygotowanie zbioru danych	70
4.7.3	Wykorzystane procedury pomiaru kątów	71
4.7.4	Wyniki	74
4.8	Sterowanie dronem Parrot ANAFI	84
4.8.1	Stanowisko badawcze	84
4.8.2	Przygotowanie zbioru danych	87
4.8.3	Modele wykorzystywane w eksperymentach oraz przygotowanie danych .	89
4.8.4	Pętla sterująca	90
4.8.5	Wyniki	92
4.8.6	Śledzenie drona Parrot ANAFI w środowisku symulowanym	102
5	Podsumowanie wyników i wnioski	107
5.1	Kierunki dalszych badań	109
	Spis literatury	111
	Bibliografia	111

Wprowadzenie

Układy robotyki oparte o systemy wizyjne znajdują wiele zastosowań, takich jak przemysłowe, m.in. chwytywanie obiektów [6], i medyczne [110]. Jedną z technik projektowania układów sterowania w takich systemach jest wykorzystanie wizyjnych lub opartych na wizji sensorów pomiarowych umieszczonych w pętli sprzężenia zwrotnego. W układach takich, obok zwykłych kamer RGB, wykorzystywane są bardziej zaawansowane systemy, takie jak: układy wielu kamer, kamery głębi, lidary czy kamery zdarzeń.

Przetwarzanie danych, których źródłem są kamery RGB, jest problemem dosyć dokładnie przebadanym, o czym świadczy choćby istnienie złożonych i szeroko stosowanych modeli jak YOLO [102], EfficientNet [120], czy Vision Transformer [27]. Bezpośrednie zastosowanie tego typu architektur – dostosowanych do pracy z gęstymi danymi 2D – nie jest jednak możliwe dla danych z bardziej zaawansowanych sensorów wizyjnych. W przypadku danych pozyskanych np. z kamery głębi lub lidaru naturalnym sposobem reprezentacji obrazów są trójwymiarowe chmury punktów, a w przypadku kamery zdarzeń – listy pikseli, dla których zaszyły zmiany jasności. Naiwne podejście, polegające na rasteryzacji takich obrazów i poddaniu typowej operacji splotu, nie wykorzystuje naturalnej rzadkości danych. Natomiast rozważanie danych trójwymiarowych bezpośrednio w postaci gęstej, jakkolwiek możliwe i koncepcyjnie naturalne, sprawia, że stosowanie takich rozwiązań jest kłopotliwe zarówno pamięciowo, jak i obliczeniowo. Gęste rozszerzenia problemu dwuwymiarowego do trójwymiarowego są tak samo naturalnym przejściem, jak dokonane już znaczenie wcześniej przejście od sygnałów jednowymiarowych do obrazów. Operacje splotu czy poolingu określa się analogicznie jak w przypadkach 1D i 2D, jednak sukcesywne podnoszenie wymiarowości danych prowadzi do problemów obliczeniowych związanych z przekleństwem wymiarowości.

Nieoszczędne gospodarowanie pamięcią znacznie wydłuża proces treningu modeli – do karty graficznej można przekazać mniejszą porcję danych. Z kolei koszt obliczeniowy sprawia, że odpowiedzi modelu nie są wystarczająco szybkie, by móc śledzić stan robota na podstawie sensorów wizyjnych z częstotliwością pozwalającą na efektywne sterowanie. Należy podkreślić, że w systemach robotyki czas przetwarzania (obok dokładności) jest jednym z kluczowych parametrów i rozsądne jest oczekiwanie czasów predykcji przynajmniej na poziomie czasu próbkowania sensora wizyjnego lub czasu przetwarzania jednej komendy przez robota.

W związku z tym problem opracowania wydajnych sieci głębokich, w tym sieci spłotowych, dla danych rzadkich – w tym chmur punktów – jest jednym z kierunków intensywnych badań realizowanych w środowisku naukowym w ostatnich latach. Rozwój ten jest dodatkowo wymuszany przez coraz nowocześniejsze sensory pomiarowe oparte na

wizji.

Można wskazać trzy główne podejścia w pracy z danymi w postaci chmur punktów: 1) metody wielowidokowe, 2) metody oparte o wokselizację (w tym rzadką wokselizację) i 3) metody pracujące bezpośrednio na chmurze punktów. Dwa ostatnie obszary wydają się być szczególnie przyszłościowe, w przeciwieństwie do metod wielowidokowych, które w ograniczony sposób wykorzystując informacje o trójwymiarowej geometrii obiektów.

Główne zadania badawcze rozważane w pracy dotyczą wizualnych torów pomiarowych, a zwłaszcza pomiaru pozycji i orientacji obiektów na podstawie danych z sensorów wizyjnych. W niniejszej rozprawie skupiono się głównie na badaniach dotyczących rzadkiej wokselizacji z użyciem operacji umożliwiających obsługę danych w postaci rzadkiej, w tym kluczowych operacji: splotu i poolingu. Jednym z wariantów operacji splotu zaadaptowanych do danych rzadkich jest tzw. operacja Sparse Submanifold Convolution (SSC). W pracy przedstawione zostaną badania i możliwości zastosowania budowanych z jej użyciem sieci do śledzenia i sterowania robotami metodą typu eye-to-hand. Budowanie modeli w ten sposób co prawda nieznacznie ogranicza dokładność reprezentacji danych, natomiast nie jest to duża niedogodność z punktu widzenia zastosowań. Rzeczywiste dane obciążone są również błędami, np. dane pozyskane z kamery głębi ZED 2i są obciążone błędem $< 1\%$ dla odległości obserwacji poniżej 3 metrów. Użycie operacji SSC pozwala natomiast na zachowanie interpretowalności modelu, na którą w ostatnich latach jest kładziony silny nacisk w badaniach w obszarze sztucznej inteligencji. W przestrzeni trójwymiarowej łatwo zwizualizować efekt działania poszczególnych warstw sieci.

Cel i teza pracy

Cel pracy

Głównym celem pracy jest badanie oraz opracowanie struktur sieci splotowych przystosowanych do wydajnej pracy z danymi pochodzącymi z nowoczesnych sensorów pomiarowych jak: kamery, kamery głębi, lidary, kamery zdarzeń (tj. np. obrazami 3D oraz chmurami punktów, danymi rzadkimi) oraz prezentacja jakości w wybranych zagadnieniach sterowania z wykorzystaniem wizualnego sprzężenia zwrotnego.

Teza pracy

Uwzględnienie rzadkości sygnałów wejściowych umożliwia opracowanie wydajniejszych (tj. szybszych z punktu widzenia uczenia oraz predykcji, przy utrzymaniu jakości predykcji) struktur sieci neuronowych w stosunku do klasycznych rozwiązań referencyjnych, w tym pracujących na danych gęstych.

Jakość predykcji należy odnosić do dokładności sensora pomiarowego. Przez wysoką jakość rozumie się tu błędy predykcji na poziomie porównywalnym lub niższym od błędów wprowadzanych przez system pomiarowy.

Teza zawarta w pracy została zweryfikowana empirycznie w zadaniach związanych ze śledzeniem i sterowaniem robotami z wykorzystaniem wizyjnych torów pomiarowych, głównie do pomiarów lokalizacji i orientacji sterowanych obiektów. Eksperymenty podzielono na dwie części. W pierwszej części weryfikacja odbywała się na wygenerowanych danych syntetycznych, natomiast w drugiej na danych rzeczywistych oraz z użyciem ka-

mer i robotów. W części drugiej modele pracujące na danych wizyjnych zastosowano do obserwacji stanu robotów oraz do zagadnień sterowania robotami.

Z tezą badawczą powiązane są następujące zagadnienia pomocnicze oraz dodatkowe pytania badawcze:

1. Czy z wykorzystaniem sieci splotowych można rozwiązać zadania lokalizacji i orientacji oraz jakie cechy i elementy struktur sieci są korzystne z punktu widzenia zadań wyznaczania lokalizacji i orientacji?
2. Czy sieci splotowe dostosowane do pracy z danymi rzadkimi pozwalają na osiągnięcie wysokiej jakości predykcji, zwłaszcza w odniesieniu do modeli gęstych?
3. Czy sieci splotowe dostosowane do pracy z danymi rzadkimi pozwalają na szybsze i oszczędniejsze pamięciowo uczenie niż sieci gęste?
4. Czy sieci splotowe dostosowane do pracy z danymi rzadkimi pozwalają na szybszą i oszczędniejszą pamięciowo predykcję niż sieci gęste?
5. Czy rzadkie sieci splotowe oparte na wokselizacji są konkurencyjne w stosunku do rozwiązań pracujących bezpośrednio na chmurze punktów?

Pytania nr 3 i 4, dotyczące przetwarzania danych rzadkich, wymagają również pewnego komentarza. Odpowiedzi na te pytania przy odpowiednio małej gęstości wydają się oczywiste w środowisku jednowątkowym, jednak jak pokazują eksperymenty oraz analiza sposobu wykonywania operacji, nie są one oczywiste, gdy obliczenia są wykonywane na kartach obliczeniowych w środowisku wielowątkowym.

Środowisko kart obliczeniowych jest dobrze dostosowane do obliczeń na macierzach gęstych i wiele algorytmów numerycznej algebry liniowej ma swoje wersje dla środowisk pracujących równolegle, np. biblioteka CUBLAS. Jednakże wiele operacji na macierzach rzadkich nie zachowuje stopnia rzadkości przy wielokrotnych złożeniach i zysk z rzadkości danych wejściowych może być ostatecznie mniejszy. Przykładowo wynik mnożenia dwóch macierzy rzadkich może być gęsty i ogólnie trudno wypowiedzieć się na temat wypełnienia wyniku iloczynu dwóch macierzy rzadkich, natomiast analiza złożoności obliczeniowej mnożenia dla macierzy rzadkich jest zagadnieniem niełatwym nawet w przypadku jednowątkowym (por. [3]). Podobnie jest w przypadku operacji splotu i poolingu, które zwykle zwiększają procent wypełnienia macierzy wynikowej. Z tych powodów w rzadkich sieciach splotowych wykorzystuje się wspomnianą wcześniej operację Submanifold Sparse Convolution zamiast zwykłego splotu, która – kosztem utraty niektórych własności splotu – zachowuje stopień wypełnienia. Ponadto jest dość istotna różnica pomiędzy zagadnieniem uczenia oraz predykcji. W przypadku uczenia na danych rzadkich niewątpliwa przewaga polega na możliwości przesłania większej liczby próbek w pojedynczym wsadzie do karty graficznej, natomiast w zagadnieniu predykcji w trybie online przesyłamy pojedynczą próbkę w częstości zdeterminowanej przez częstotliwość urządzenia pomiarowego. Stąd zysk związany z przesyłaniem danych widoczny w trakcie uczenia jest niwelowany w zagadnieniu predykcji w trybie online. Na wszystkie te aspekty kładziony był nacisk w trakcie eksperymentów przedstawionych w pracy.

W ramach badań zaprojektowano i zaimplementowano autorskie modele rzadkich splotowych sieci neuronowych, a wszystkie prezentowane eksperymenty przeprowadzono samodzielnie, wykorzystując zasoby Laboratorium Automatyki Mobilnej oraz Katedry Sztucznej Inteligencji i Matematyki Stosowanej, w której realizowano rozprawę. W trakcie badań zaplanowano eksperymenty i przygotowano autorskie zbiory danych. Całość prac

programistycznych wykonano w środowisku języka Python. Wybrane kody źródłowe, modele i dane udostępniono na platformach Zenodo [105], GitHub [104] oraz Edysk ZUT.

Główne wyniki uzyskane w pracy

W badaniach wstępnych, na danych symulacyjnych, wykazano, że rzadka sieć splotowa może być z powodzeniem wykorzystywana do zadań predykcji lokalizacji i orientacji obiektów. W pierwszym eksperymencie, dla danych dwuwymiarowych, dla zaprojektowanej architektury uzyskano dokładność przewidywania pozycji prostego obiektu na poziomie około 7 pikseli (przy rozdzielczości 3840×2160) oraz 0.04 radiana dla orientacji. Wyniki porównawcze modelu z warstwami przystosowanymi do obsługi danych rzadkich wykazały, że można uzyskać czas predykcji kilkukrotnie krótszy niż dla modelu gęstego. W drugiej części badań wstępnych – przewidywaniu lokalizacji i orientacji obiektu w przestrzeni trójwymiarowej – opracowano sieć korzystającą z operacji unfold (podział trójwymiarowego obrazu na mniejsze bloki prostopadłościenne) i uzyskano dokładność przewidywania pozycji na poziomie 0.01 m – czyli taką, jak rozmiar woksela na symulowanej scenie o wymiarach $3 \times 3 \times 3$ m. Dla predykcji kąta uzyskano dokładność na poziomie 0.04 – 0.05 rad przy czasie predykcji dwudziestokrotnie krótszym niż dla modelu gęstego.

W części badań poświęconej pracy z rzeczywistymi układami robotyki przeprowadzono trzy eksperymenty. Pierwszy dotyczył zagadnienia sterowania czterema robotami mobilnymi na podstawie pomiaru pozycji z użyciem kamery RGB oraz architektury YOLO. Na potrzeby uczenia modelu zebrano około 100 000 próbek. Po uwzględnieniu przesunięcia wynikającego z opóźnienia sterowania nadążnego, uzyskano dokładność odtwarzania trajektorii robotów na poziomie kilku pikseli – przy rozdzielczości obrazu 1280×720 . Pokazano w ten sposób, że sieć splotowa z powodzeniem może być użyta do sterowania robotami, z częstotliwością odpowiedzi zbliżoną do prędkości rejestrowania obrazu przez kamerę. Wyniki opublikowano w materiałach konferencji ICAISC 2023 [106].

W drugim eksperymencie rzeczywistym ograniczono się do śledzenia ramienia robota, natomiast wizyjny układ pomiarowy zmieniono na kamerę głębi Intel RealSense D435. Jej użycie wymusiło zastosowanie modelu przystosowanego do pracy z danymi w formie rzadkiej. W związku z tym zaprojektowano prostą, lecz bardzo wydajną sieć splotową i nauczono ją z użyciem około 5000 próbek zarejestrowanych w laboratorium. Dla czasu predykcji 20 ms (50 klatek na sekundę) w jednym wątku procesora (bez GPU) uzyskano dokładność przewidywania stanu robota (kątów skręcenia członów robota) na poziomie 2 – 3 stopni. Po rozwiązaniu zadania kinematyki prostej, przełożyło się to na dokładność przewidywania pozycji efektora robota w zakresie 6 – 12 mm przy rozmiarze woksela 4 mm oraz odległości obserwacji wynoszącej metr i dokładności kamery $< 2\%$ dla odległości do 2 m. Wstępne wyniki tych eksperymentów przedstawiono na konferencji PP-RAI 2022 [52], a końcowe opublikowano w artykule [107].

W ostatnim, najbardziej złożonym eksperymencie, połączono zagadnienia z dwóch wcześniejszych – zrealizowano śledzenie i sterowanie dronem (Parrot ANAFI) w pomieszczeniu zamkniętym, wykorzystując w pętli sterującej kamerę głębi ZED 2i oraz

opracowany model sieci splotowej z warstwami rzadkimi. W ramach eksperymentu zebrano ponad 750 000 próbek uczących i stworzono dwie sieci splotowe. W badania opracowano i przeanalizowano kilka modeli sieci splotowych pracujących na danych rzadkich. Pierwszy model, zaprezentowany na konferencji PP-RAI 2024 [53], umożliwia predykcję kąta obrotu drona w osi Z z wysoką dokładnością (0.09 rad). Jego wadą jest praca na obrazie 3D przedstawiającym wyciętego drona, co wymaga użycia dodatkowego detektora (np. YOLO) do detekcji obiektu i wyznaczenia jego pozycji. W związku z tym stworzono nową architekturę, dokonującą predykcji położenia i kąta obrotu jednocześnie, a wcześniejszy model potraktowano jako referencyjny. Drugi model przyjmuje na wejście obraz trójwymiarowy przedstawiający całą scenę, co pozwala na jednoczesne przewidywanie pozycji i orientacji badanego obiektu. Unikalną cechą charakterystyczną zaproponowanej sieci jest zastosowanie projekcji wstępnie przetworzonych obrazów 3D do podprzestrzeni jedno- i dwuwymiarowych. Porównanie obu architektur zaprezentowano i opublikowano na konferencji ECAI 2024 [105] (CORE A). W niniejszej rozprawie przedstawiono wyniki nieco lepsze niż przedstawione na konferencji, uzyskane po dotrenowaniu omawianego modelu. Ostatecznie w zadaniu śledzenia drona błąd przewidywania pozycji jest na poziomie woksela (1 cm) dla odległości obserwacji 1.5 – 4 m, a błąd przewidywania kąta obrotu wokół osi Z wyniósł 0.14 rad. W zadaniu sterowania w układzie zamkniętym, w którym dodatkowym składnikiem błędu jest błąd kontrolera drona oraz obecność zakłóceń, uzyskano dokładność odtwarzania zadanej trajektorii dla pozycji w granicach 4 cm oraz dla kąta obrotu 0.18 rad.

Badania dotyczące drona rozszerzono o porównanie stworzonego modelu z trzema strukturami pracującymi bezpośrednio na chmurze punktów: PointNet, Dynamic Graph CNN oraz Point Cloud Transformer. Uzyskane wyniki wykazały kilkukrotnie lepszą dokładność przewidywania pozycji oraz porównywalną dla predykcji kąta. Wstępne wyniki tych eksperymentów przedstawiono na konferencji ICSYS 2024 [51], a rozszerzone są zaprezentowane w rozprawie.

Początkowym celem badań związanych z predykcją stanu drona była próba rozpoznawania wszystkich trzech kątów obrotu. Próby nastrojenia modelu do przewidywania wszystkich kątów, przy wykorzystaniu danych rzeczywistych, napotkały jednak trudności związane z dużym szumem danych w stosunku do zmian kąta w pozostałych dwóch osiach (pitch i roll) oraz z uzyskaniem danych zawierających odpowiednią zmienność kąta w tych osiach. W związku z tym, dzięki współdziałaniu z pracownikami Katedry Systemów Multimedialnych, przygotowano środowisko do symulacji i tworzenia danych syntetycznych. Środowisko symulacyjne zawierało przybliżony obraz sceny z dronem i umożliwiało generowanie obrazów w postaci chmury punktów. Dla takich danych syntetycznych nastrojono dwa modele. Dla modelu użytego wcześniej do predykcji stanu drona uzyskano dokładność predykcji pozycji na danych symulowanych na poziomie 5 mm (czyli pół woksela) oraz 0.06 rad. Przy użyciu tych samych danych wyuczono model użyty w badaniach wstępnych do predykcji dla danych syntetycznych 3D. Model ten uzyskał dla danych symulowanych nieco niższą dokładność przewidywania pozycji – około 1 cm – oraz bardzo wysoką dokładność predykcji kąta obrotu w osi Z – 0.03 radiana. Działanie modeli strojonych wyłącznie na danych symulowanych przetestowano na danych rzeczywistych, uzyskując dokładność przewidywania pozycji na poziomie 4 – 14 cm oraz 0.18 radiana dla kąta wokół osi Z . Częściowe wyniki z tych eksperymentów zostały zaprezentowane

na konferencji PP-RAI 2025 [54].

Do najważniejszych wyników zawartych w rozprawie zaliczam: 1) opracowanie dwóch architektur sieci neuronowych zaadaptowanych do pracy z danymi w formie rzadkiej, szczególnie dobrze dostosowanych do predykcji pozycji i kąta obrotu, 2) wykazanie różnic w czasach uczenia i predykcji modeli dostosowanych do danych rzadkich względem modeli gęstych, 3) wykazanie konkurencyjności proponowanych architektur względem modeli referencyjnych spotykanych w literaturze, 4) wykazanie praktycznej użyteczności opracowanych modeli w zagadnieniu sterowania ruchem robotów mobilnych w pomieszczeniu zamkniętym.

Zawartość i układ rozprawy

Rozprawa składa się z czterech rozdziałów. Rozdziały 1 i 2 są teoretyczne i wprowadzają do tematyki rozprawy. Rozdział 1 poświęcony jest zagadnieniom sterowania z wizualnym sprzężeniem zwrotnym. Omówiono w nim sensory wizyjne, które mogą być stosowane w torze pomiarowym, a także scharakteryzowano przekształcenia geometryczne przestrzeni trójwymiarowej, których używano w trakcie eksperymentów. Rozdział 2 poświęcony jest tematyce sieci głębokich, które mogą być stosowane w opartych na wizji torach pomiarowych oraz możliwości ich użycia w zagadnieniach sterowania. Rozdział ten zawiera przegląd stanu badań i literatury dotyczącej sieci neuronowych dostosowanych do pracy z danymi w formie rzadkiej oraz chmurami punktów. Rozdział 3 stanowi wprowadzenie do części eksperymentalnej – przedstawiono w nim autorskie architektury rzadkich sieci splotowych, które stosowano w eksperymentach. Rozdział 4 zawiera omówienie i podsumowanie głównych wyników badań. Przedstawia stanowisko badawcze oraz zawiera szczegółowy opis sześciu eksperymentów. Rozprawę zakończono podsumowaniem, w którym wskazano także kierunki dalszych badań.

W pracy użyto następujące oznaczenia i skróty:

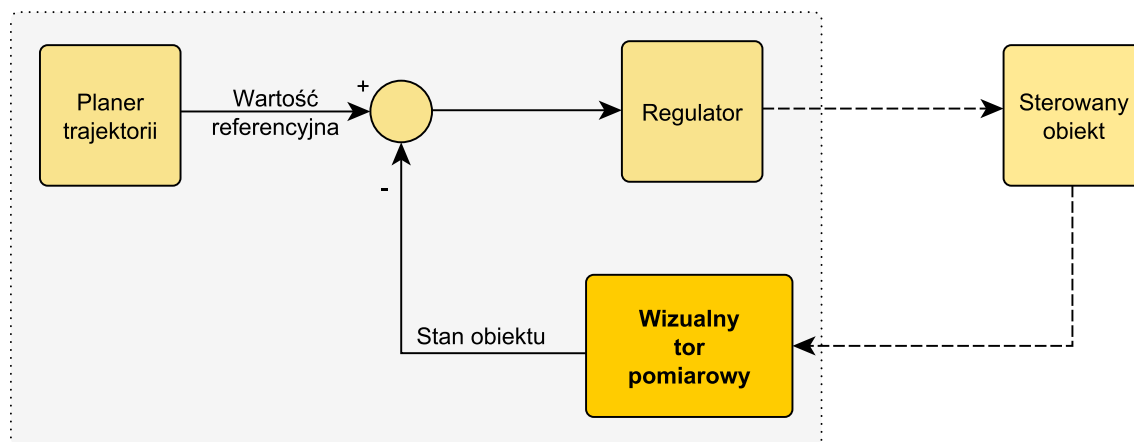
SSC	Sparse Submanifold Convolution (Zmodyfikowany operator rzadkiego splotu)
FPS	Frames Per Second (liczba klatek na sekundę)
GPU	Graphics Processing Unit (Procesor graficzny)
CPU	Central Processing Unit (Procesor)
IMU	Inertial Measurement Unit (Miernik bezwładności)
MAE	Mean absolute error (Średni błąd bezwzględny)
MSE	Mean Squared Error (Błąd średniokwadratowy)
MLP	Multilayer Perceptron (Perceptron wielowarstwowy)
kNN	k -nearest neighbors (Algorytm k najbliższych sąsiadów)
CNN	Convolutional Neural Network (Splotowa sieć neuronowa)
DGCNN	Dynamic Graph Convolutional Neural Network (Dynamiczna grafowa splotowa sieć neuronowa)
PCT	Point Cloud Transformer (Transformer dla chmur punktów)

1. Układy sterowania z wizualnym sprzężeniem zwrotnym

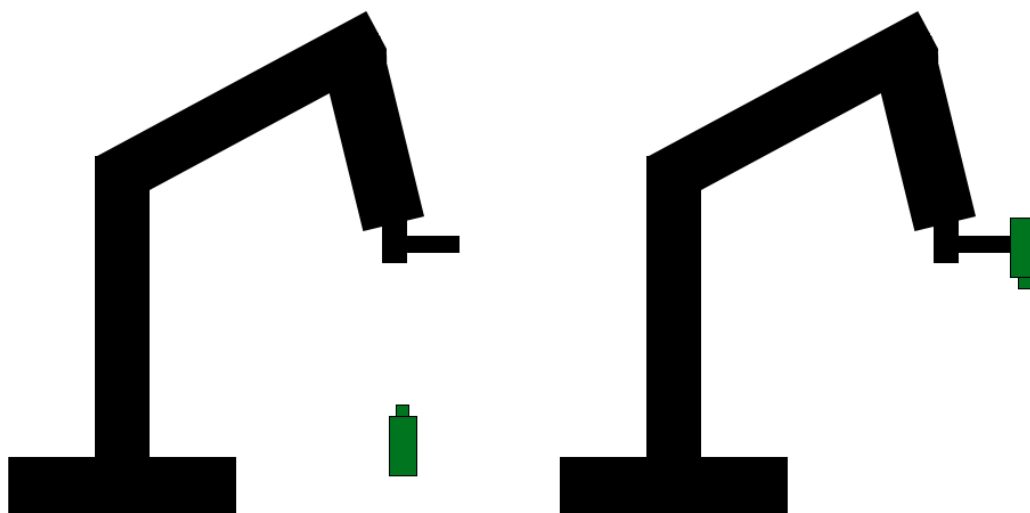
Jedną z technik projektowania układów sterowania w systemach robotyki jest wykorzystanie wizyjnych lub opartych na wizji sensorów pomiarowych umieszczonych w pętli sprzężenia zwrotnego. W układach takich obok zwykłych kamer RGB wykorzystywane są bardziej zaawansowane systemy, takie jak: układy wielu kamer, kamery głębi, lidary czy kamery zdarzeń. Układy takie stanowią główny przedmiot badań prowadzonych w ramach niniejszej pracy.

Użycie nowoczesnych sensorów wizyjnych, takich jak kamery, lidary czy kamery głębi, zwiększa możliwości śledzenia, lokalizacji, czy pomiaru orientacji obiektów. Sensory wizyjne znajdują zastosowanie w sytuacjach, gdy bezpośredni pomiar jest utrudniony, czy wręcz nie dysponujemy sensorami do bezpośredniego pomiaru wielkości wymaganych do sterowania, jak na przykład: pomiar pozycji sterowanego obiektu, pomiar orientacji obiektów o złożonej geometrii, detekcja obiektu połączona z lokalizacją, zadania lokalizacji i mapowania (SLAM), odometria czy zagadnienia fotogrametryczne. Ogólną strukturę zamkniętego układu sterowania z torem pomiarowym w pętli sprzężenia zwrotnego przedstawiono na rys. 1.1.

Dane z wizyjnych sensorów pomiarowych zwykle nie nadają się bezpośrednio do zastosowania w układzie regulacji i wymagają dodatkowych zabiegów numerycznych [18].



Rysunek 1.1: Ogólny schemat układu regulacji z wizualnym sprzężeniem zwrotnym. (Źródło: opracowanie własne)



Rysunek 1.2: Ilustracja idei eye-in-hand (po lewej) oraz eye-to-hand. (Źródło: opracowanie własne)

W zależności od typu sensora dostarczane dane mają różne formaty, z których najczęściej spotykane są zwykłe obrazy RGB, obrazy RGB-D oraz chmury punktów. Formaty te różnią się znacznie sposobem reprezentacji: te pierwsze przechowywane są w postaci gęstej, natomiast chmury punktów stanowią rzadką reprezentację obrazu. Reprezentacje rzadkie i chmury punktów są bardziej naturalną i oszczędną formą reprezentacji danych pochodzących z sensorów głębi.

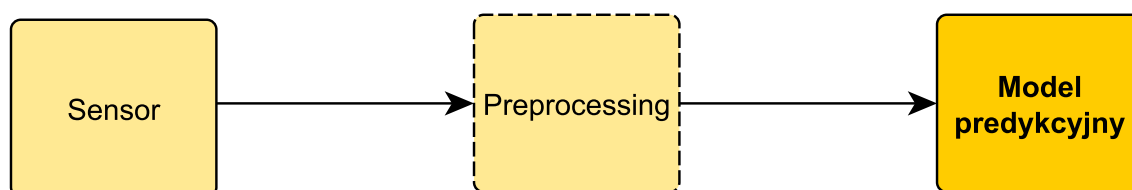
Modele zdolne do pracy z danymi w postaci rzadkiej, w połączeniu z dostarczającymi je sensorami, mogą być zastosowane w zadaniach sterowania – w tym w sterowaniu z wizualnym sprzężeniem zwrotnym [14]. Polega ono na tym, że stan obiektu ustalany jest z użyciem wizualnego toru pomiarowego. Uzyskana w ten sposób wartość jest porównywana z wartością referencyjną, generując błąd regulacji na podstawie którego regulator generuje sygnał sterujący (rysunek 1.1).

Układy wizyjne stosowane w systemach robotyki mogą być umieszczone względem robota/manipulatora zasadniczo na dwa sposoby (rysunek 1.2):

1. Eye-in-hand – kamera umieszczona na robocie mobilnym lub ramieniu robota poruszająca się wraz z robotem,
2. Eye-to-hand – kamera umieszczona nieruchomo względem środowiska, obserwująca sterowany obiekt.

Oba sposoby znajdują szerokie zastosowania w robotyce. Kamera umieszczona na robocie może być na przykład użyta do sterowania robotami endoskopowymi [64], obserwacji obiektów [40], przenoszenia obiektów [88], do sterowania mikrorobotami magnetycznymi [148]. Z kolei kamer obserwujących przestrzeń pracy robota można używać przykładowo w zadaniach sterowania robotem tnącym płyty [90], sterowania robotami mobilnymi [72], czy śledzenia i chwytania ruchomych celów [162].

1.1 Elementy toru pomiarowego



Rysunek 1.3: Wizualny tor pomiarowy. (Źródło: opracowanie własne)

Blok symbolizujący wizualny tor pomiarowy wymaga dokładniejszego omówienia. W jego ramach wyróżnić można trzy elementy (rysunek 1.3):

- **Sensor wizyjny** – dostarczający informacji wizualnej na temat środowiska i obiektu. Rolę tą w sterowaniu z wizualnym sprzężeniem zwrotnym mogą pełnić nowoczesne sensory wizyjne (np. kamera, lidar czy kamera głębi).
- **Preprocessing** – jest elementem opcjonalnym, aczkolwiek mogącym pozytywnie wpływać na efekty działania modelu predykcyjnego. Może obejmować takie elementy jak np.: skalowanie, wygładzanie i filtrację, zmianę układu odniesienia, usuwanie tła czy przepróbkowywanie danych.
- **Model predykcyjny** – jego zadaniem jest określanie stanu układu w postaci numerycznej na podstawie danych z sensora, w formie możliwej do zastosowania w algorytmie regulacji. Rolę modelu predykcyjnego może pełnić sieć neuronowa, w tym sieć spłotowa.

Główne sensory stosowane spotykane w układach z wizyjnym sprzężeniem zwrotnym:

- **Kamery RGB** to podstawowy rodzaj urządzenie rejestrującego obrazy w postaci tablicy zawierającej informację o pikselach w przestrzeni barw RGB.
- **Kamery głębi** (RGB-D) – oprócz kanałów RGB, kamery takie przesyłają dodatkowo dla każdego z pikseli informację na temat odległości pomiędzy elementem obrazu, a soczewką kamery. Za pomocą odpowiednich przekształceń obraz taki może być przetransformowany do chmury punktów.
- **Kamery zdarzeń** (ang. event camera) przesyłają do odbiorcy wyłącznie informację o tych pikselach, dla których jasność obrazu uległa zmianie. Dane dostarczane są w postaci obrazu rzadkiego. Kamery takie charakteryzują się znacznie krótszym czasem pobrania pojedynczej klatki w stosunku do typowych kamer RGB.
- **Lidary** rejestrują obraz z jednej lub więcej płaszczyzn (linii) w dużym zakresie kąta. Obraz ten jest w formie chmury punktów i pozbawiony kanałów RGB.

Porównanie sensorów dostarczających obrazy w postaci rzadkiej przedstawiono w tabeli 1.1.

Zasadnicza część pracy dotyczy analizy obrazów w postaci chmur punktów. Wygenerowana przez sensory chmura punktów jest bardziej naturalną reprezentacją trójwymiarowej przestrzeni operacyjnej robota niż dwuwymiarowe obrazy. Dane w takiej postaci można wykorzystać w wielu różnych zadaniach robotyki np.: w zadaniu chwytania obiektów [70], czy też do unikania kolizji [113]. Dane w takiej postaci można również łączyć z pomiarami z kamery RGB zamontowanej na robocie [22] czy na dronie [69].

Tabela 1.1: Porównanie sensorów wizyjnych.

Sensor	Zalety	Wady	Rozdzielczość	FPS
Kamera głębi	– trójwymiarowy obraz – rejestruje kolor	– ograniczona dokładność lub ograniczona odległość obserwacji	640×480 – 2208×1242	5 – 120
Kamera zdarzeń	– bardzo duża liczba klatek na sekundę – niski pobór energii – oszczędny transfer danych	– niska rozdzielczość – wysoki koszt – brak informacji o kolorze	320×240 – 1280×720	50000 – 300000
Lidar	– wysoka dokładność – szeroki kąt widzenia – obrazowanie na dużą odległość	– niska rozdzielczość pionowa – brak informacji o kolorze	4-128 linii, rozdzielczość kątowa (pozioma) do 0.1°	10 – 50

Same drony również mogą być śledzone i sterowane z użyciem wizualnego sprzężenia zwrotnego. Jest to szczególnie istotne w środowisku pozbawionym sygnału GPS, takim jak zamknięte pomieszczenia. Wiele rozwiązań opartych jest o kamerę RGB wbudowaną w drona [35] bądź sensor głębi [59] dodatkowo zamontowany na dronie. Rozbudowa drona o dodatkowe wyposażenie jest jednak nie zawsze możliwa, ze względu na wagę kamer i dodatkowych układów obliczeniowych.

Wśród rozwiązań śledzących i sterujących dronami w oparciu o kamerę zewnętrzną można wyróżnić te oparte o detektor taki jak YOLO, pracujący z obrazami RGB – w systemach jedno [95] i wielokamerowych [127]. Inną grupę rozwiązań stanowią te, które wykorzystują dodatkowe sensory: termiczne [126], akustyczne [119] czy też skanery częstotliwości radiowych [5]. Dodatkowe dane mogą być pozyskiwane również z lidara [26] – będą to wówczas dane w postaci rzadkiej.

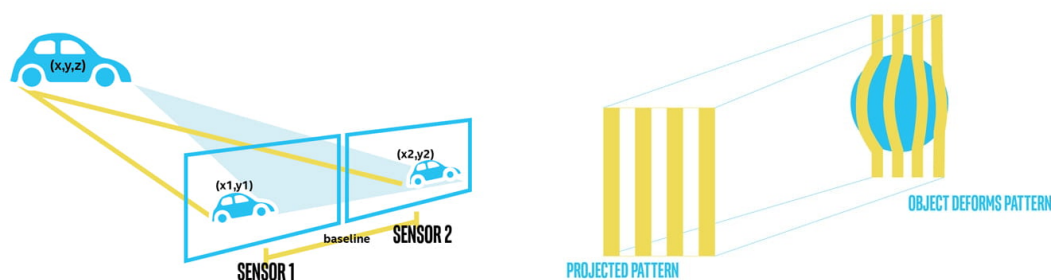
1.2 Sensory

1.2.1 Kamera głębi

Popularną techniką stosowaną w kamerach głębi jest sterowizja [9], mająca naśladować sposób tworzenia trójwymiarowego obrazu przez ludzki mózg i oczy. Tego rodzaju kamera wyposażona jest w dwie kamery RGB, które umieszczone są w ustalonej odległości od siebie. Na podstawie zależności geometrycznych pomiędzy nimi oraz zarejestrowanego obrazu z obu kamer możliwa jest estymacja głębi (rysunek 1.4). Dla obiektów położonych bliżej przesunięcie będzie większe niż dla tych, które umieszczone są dalej od sensorów. Dodatkowo – na potrzeby scen ze słabym oświetleniem – możliwe jest dodatkowe użycie projektora podczerwieni.

Inna metoda obrazowania głębi to wykorzystanie światła strukturalnego oraz światła

kodowanego (structured, coded light). Kamery działające w ten sposób posiadają jeden obiektyw oraz dodatkowo emitują światło (zwykle podczerwone), w którym zakodowany jest pewien wzorec. Może to być wzorec wizualny, czasowy lub połączenie obu. Sposób zniekształcenia wzorca na obrazowanym obiekcie pozwala oszacować odległość. Na rysunku 1.4 (po prawej) przedstawiono przykład, w którym emitowane paski zmieniają kształt na piłce.



Rysunek 1.4: Zasada działania kamery głębi. Po lewej: estymacja głębi w oparciu o dwa obrazy i wiedzę o odległości między sensorami RGB. Po prawej: przykładowy wzorec światła strukturalnego emitowany przez kamerę głębi. (Źródło: [9])

1.2.2 Lidar

Lidar wykorzystuje do pomiaru wiązkę lasera [1], która po natrafieniu na obiekt odbija się i wraca od odbiornika. Na podstawie informacji o czasie powrotu wiązki oraz prędkości światła możliwe jest obliczenie odległości od obiektu, zgodnie z wzorem:

$$d = \frac{c \cdot t}{2} \quad (1.1)$$

Przemysłowy dostawca SICK AG wyróżnia trzy kategorie lidarów [44]: jedno-, dwu- i trójwymiarowe. Lidar jednowymiarowy umożliwia wyłącznie pomiar odległości. Z kolei dwuwymiarowy pozwala na pozyskanie informacji z jednej płaszczyzny. Jest to możliwe dzięki odbijaniu impulsu świetlnego od obracającego się lustra. Lidar trójwymiarowy daje natomiast możliwość zebrania informacji z więcej niż jednej płaszczyzny.

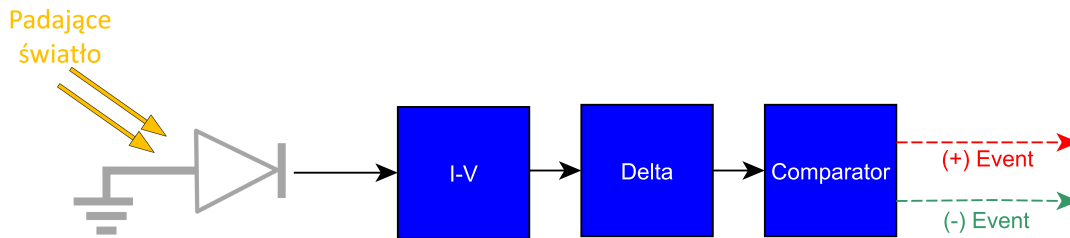
1.2.3 Kamera zdarzeń

Główną ideą, o którą oparta jest kamera zdarzeń [73], jest naśladowanie sposobu odbierania zmian światła przez ludzkie oko [32]. Kamera taka odbiera zmiany światła w sposób asynchroniczny i niezależny dla każdego z pikseli. Jest to odmienne podejście niż w konwencjonalnych kamerach, w których obraz rejestrowany jest przez migawkę globalną (cała matryca jednocześnie) lub migawkę postępową (sekwencyjne odświeżanie).

Na rysunku 1.5 przedstawiono schemat działania kamery zdarzeń dla pojedynczego piksela. Padające światło jest zamieniane przez fotodiodeę na napięcie (blok I-V). Następnie obliczana jest różnica stosunku do napięcia odniesienia (blok Delta). Jeśli wartość progowa

zostanie przekroczona (blok Comparator), następuje wygenerowanie zdarzenia: zmiana wartości piksela.

Kamera zdarzeń dostarcza pomiaru w postaci rzadkiego obrazu 2D – zawarta jest wyłącznie informacja o tych pikselach, w których doszło do zmiany światła (oraz znacznik czasu takiej zmiany). Umożliwia to znaczną oszczędność transferu. Kamery takie posiadają również wysoką rozdzielczość czasową (do 1 μ sec).



Rysunek 1.5: Schemat działania kamery zdarzeń. (Źródło: opracowanie własne w oparciu o [32])

1.3 Transformacje geometryczne w 3D

W sekcji tej zebrano najważniejsze informacje na temat odwzorowań geometrycznych wykorzystywanych w dalszej pracy – przede wszystkim izometrii. Przekształcenie $f : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ nazywamy izometrią przestrzeni $\mathbb{R}^3$, jeśli dla dowolnych $a, b \in \mathbb{R}^3$ spełniony jest warunek:

$$d(a, b) = d(f(a), f(b)),$$

gdzie d jest pewną metryką. Izometrią jest więc przekształcenie zachowujące odległość.

W zagadnieniach robotyki szczególną rolę odgrywają dwie izometrie – translacje oraz obroty, w szczególności w przestrzeni trójwymiarowej. Przekształcenia te mają znaczenie także dla modeli predykcyjnych. Powinny one zachowywać się w sposób translacyjnie i rotacyjnie niezmienniczy, a więc rozpoznawać położenie śledzonego obiektu niezależnie od jego obrotu oraz jego orientację niezależnie od położenia na obserwowanej scenie.

W ogólnym przypadku, translację oraz obrót w przestrzeni $\mathbb{R}^3$ można zapisać z użyciem jednej macierzy o wymiarze 4×4 [84], nazywanej macierzą jednorodnego przekształcenia:

$$F = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix},$$

gdzie T to wektor translacji o wymiarze 1×3 , a R to ortogonalna macierz obrotu o wymiarze 3×3 . Dla tak skonstruowanej macierzy F , punkt $X \in \mathbb{R}^3$, $X = [x, y, z]$, można przekształcić (obrócić i następnie przesunąć) wykonując pojedyncze mnożenie:

$$X' = \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Obie operacje przesunięcia i obrotu są odwracalne. Ważną właściwością macierzy przekształcenia jest sposób wyznaczenia jej odwrotności, która wyraża przekształcenie odwrotne:

$$F^{-1} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} R^T & -R^T T \\ 0 & 1 \end{bmatrix}.$$

Do uzyskania macierzy odwrotnej do macierzy R wystarczające jest wykonanie transpozycji – dzieje się tak, ponieważ macierz R to macierz ortogonalna. Dalszego wyjaśnienia wymagają dwie operacje wchodzące w skład przekształcenia jednorodnego – translacja i obrót.

Translacja

Dla danego wektora $X = [x, y, z]$ w przestrzeni afinicznej $\mathbb{R}^3$ translacją o wektor $T = [T_x, T_y, T_z]$ nazywamy przekształcenie:

$$X' = T(X) = X + T = \begin{bmatrix} x + T_x \\ y + T_y \\ z + T_z \end{bmatrix}.$$

Jeżeli T nie jest wektorem zerowym, przekształcenie to nie posiada punktów stałych, tj. nie zmieniających się na skutek przekształcenia.

Obrót

Obrót najprościej jest zdefiniować jako izometrię posiadającą punkty stałe - tj. punkty leżące na osi obrotu. Taka definicja obejmuje również jednak również tzw. obroty niewłaściwe czyli np. obicia względem prostych/płaszczyzn. W przestrzeni trójwymiarowej obrót może być zadany za pomocą macierzy ortogonalnej (tj. takiej, dla której $RR^T = I$ lub $R^{-1} = R^T$). Macierz taka może być przedstawiona jako złożenie obrotów prostych (tj. wokół osi układu) lub równoważnie przedstawiona w postaci iloczynu macierzy obrotów wokół ustalonych osi układu.

W przestrzeni trójwymiarowej można wyróżnić trzy główne obroty: względem osi X , Y oraz Z układu współrzędnych. Wyrażone są one za pomocą poniższych macierzy obrotu [25]:

$$R_x(\alpha_x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix},$$

$$R_y(\alpha_y) = \begin{bmatrix} \cos(\alpha) & 0 & \sin(\alpha) \\ 0 & 1 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) \end{bmatrix},$$

$$R_z(\alpha_z) = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Złożenia obrotów można uzyskać poprzez mnożenie odpowiednich macierzy obrotu. Dla zestawu kątów $\alpha_x, \alpha_y, \alpha_z$ finalna macierz obrotu zostanie uzyskana następująco:

$$R(\alpha_x, \alpha_y, \alpha_z) = R_z(\alpha_z)R_y(\alpha_y)R_x(\alpha_x) \quad (1.2)$$

Należy przy tym pamiętać o tym, że w przestrzeni trójwymiarowej – w przeciwieństwie do przypadku dwuwymiarowego – złożenie obrotów nie jest przemienne i kolejność wykonywania obrotów ma znaczenie, a kolejność wykonywania obrotów wymaga przyjęcia pewnej konwencji.

Zgodnie z twierdzeniem Eulera o obrocie [31] złożenie obrotów względem osi układu tak naprawdę zadaje w przestrzeni obrót wokół pewnej osi oraz o pewien kąt. Oś tę najwygodniej jest wyznaczyć znajdując wektor własny odpowiadający wartości własnej równej 1. Jeżeli macierz obrotu jest właściwa (tj. nie określa odbić lub obrotów o 180 stopni) to pozostałe wartości własne są liczbami zespolonymi sprzężonymi zawierającymi informację o kącie obrotu. Niewłaściwa macierz obrotu ma wartość własną równą -1 .

Macierz obrotu w przestrzeni trójwymiarowej określona jest za pomocą dziewięciu parametrów, natomiast twierdzenie Eulera o obrocie je wskazuje, że parametry te nie są niezależne, tj. możliwe jest opisanie orientacji obiektu w przestrzeni za pomocą jedynie trzech liczb – kątów obrotu wokół odpowiednich osi. W zależności od kolejności wykonywania obrotów oraz tego, czy osie układu zmieniają się wraz z obracaniem obiektem czy nie, uzyskuje się różne zestawy kątów. Ogólnie koncepcja ta nosi nazwę kątów Eulera. Należy przy tym zaznaczyć, że w różnych źródłach można spotkać się z używaniem tej nazwy w stosunku do różnych zestawów obrotów. W związku z tym, na potrzeby niniejszej pracy, w oparciu o [109] przyjęto podział na:

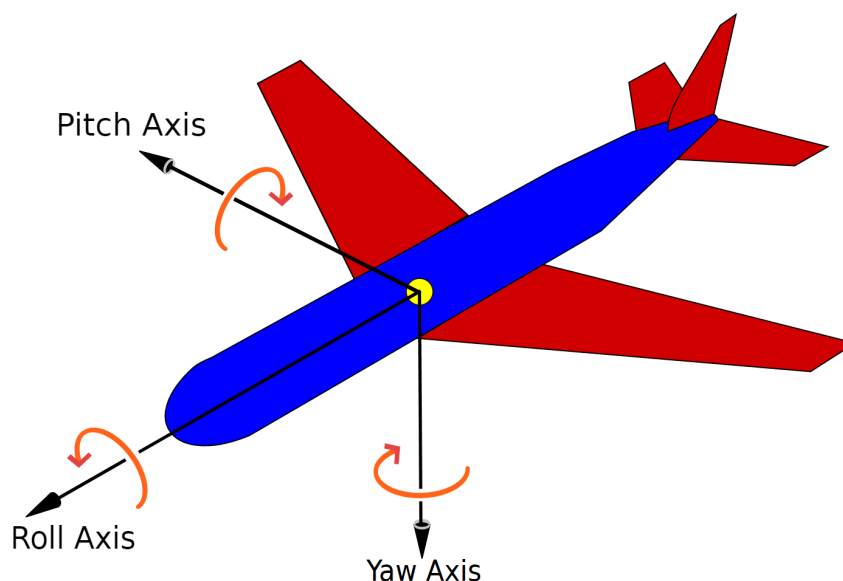
- właściwe kąty Eulera, dla których pierwsza i trzecia oś obrotu jest taka sama jak pierwsza [21]: $XYX, XZX, YXY, YZY, ZZX, ZYZ$;
- kąty Taita-Bryana, w których każdy kąt odnosi się do innej osi: $XYZ, XZY, YXZ, YZX, ZXY, ZYX$.

Przy stosowaniu obu zestawów obrotów rozróżnić można ich potraktowanie jako obroty wewnętrzne (ang. intrinsic rotations) lub zewnętrzne (ang. extrinsic rotations). Obroty wewnętrzne odnoszą się do sytuacji, gdy obrót obiektu dokonywany jest względem układu zorientowanego z obracającym obiektem. Z kolei obroty zewnętrzne wykonywane są wówczas, gdy obiekt obracany jest względem osi nieruchomego układu współrzędnych.

Jeden z zestawów kątów Taita-Bryana (XYZ) nosi nazwę kątów roll-pitch-yaw. Ta sekwencja obrotów używana jest w lotnictwie, tworząc układ obrotów skojarzony z samolotem (rysunek 1.6). Z osią X , skierowaną wzdłuż samolotu i zorientowaną w kierunku przodu, stowarzyszony jest obrót roll. Oś Y (obrot pitch) zorientowana jest w kierunku prawego skrzydła, a oś Z w dół, co zgodnie z regułą prawej dłoni powoduje, że obrót yaw kreci się zgodnie z ruchem wskazówek zegara. Należy o tym pamiętać, gdyż zwykle nieruchomy układ współrzędnych posiada oś skierowaną do góry i w takim układzie kąt obrotu wokół osi Z , zgodnie z regułą prawej dłoni, mierzony jest w stronę przeciwną do kąta yaw.

Rzut ortogonalny, macierze projekcji

Rzutem nazywa się odwzorowanie liniowe, dla którego niezmiennikiem jest obraz tego odwzorowania. Czyli P jest projekcją, jeżeli dla każdego $v \in \mathfrak{V}(P)$, mamy $Pv = v$,



Rysunek 1.6: Ilustracja osi roll, pitch i yaw. (Źródło: Wikipedia)

lub równoważnie dla każdego $x \in \mathbb{R}^n$ mamy $P(P(x)) = P^2x = Px$. Wynika stąd, że jeżeli obraz jest niepusty to 1 musi być wartością własną odwzorowania P . Jeżeli jądro odwzorowania P jest niepuste, czyli dla pewnego $x \in \mathbb{R}^n, x \neq 0$ zachodzi $P(x) = 0$. Oznacza to, że 0 również jest wartością własną tego odwzorowania. Na tej podstawie macierze projekcji można zdefiniować jako takie macierze, które posiadają tylko wartości własne 0 lub 1. Jeżeli dane są dwa jednostkowe wektory należące do obrazu projekcji zapisane kolumnowo, np. wersory osi X i Y :

$$A_{xy} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix},$$

to macierz projekcji P_{xy} może być przedstawiona jako iloczyn AA^T :

$$P_{xy} = A_{xy}A_{xy}^T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

2. Wykorzystanie sieci głębokich do pracy z chmurami punktów

2.1 Dane w wizyjnych systemach pomiarowych

2.1.1 Obrazy RGB

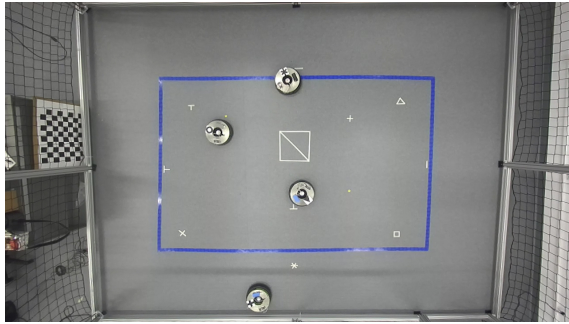
Standardowa kamera RGB generuje sekwencję obrazów, z których każdy przechowywany jest w formie tablicy o wymiarach $W \times H \times C$, gdzie W oznacza szerokość obrazu, H wysokość (obie wartości w pikselach), a C oznacza liczbę kanałów - typowo używane są trzy (R – czerwony, G – zielony, B – niebieski). Sposób kodowania kolorów jak i kolejność wymiarów mogą się różnić w zależności od stosowanych formatów lub bibliotek. Z numerycznego punktu widzenia kluczowe jest, że obraz trzymany jest w tablicy, która stanowi ciągły obszar w pamięci – co może mieć znaczenie dla przetwarzania. Niezależnie od wypełnienia klatki obrazy takie mają zawsze jednakowy rozmiar, dzięki czemu ilość wykorzystywanej pamięci może być łatwa do oszacowania.

2.1.2 Obrazy rzadkie. Chmury punktów

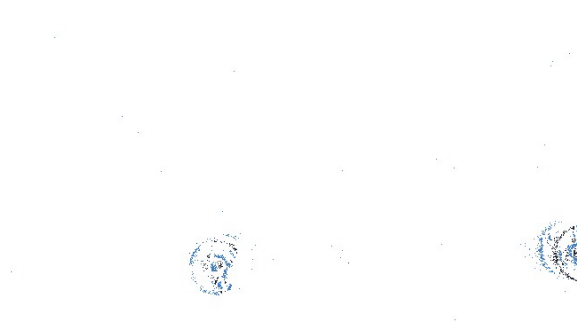
Zależnie od stopnia wypełnienia obraz gęsty może być przekształcony do postaci rzadkiej na wiele różnych sposobów. Obraz gęsty może zostać przekształcony do formy obrazu rzadkiego z zastosowaniem technik preprocessingu, takich jak na przykład usunięcie tła, czy zastosowanie filtrów krawędziowych i progowania.

Jak wspomniano wcześniej, niektóre sensory generują na wyjściu obrazy zakodowane w postaci rzadkiej. Dane w takiej formie dostarczają przykładowo kamery głębi oraz kamery zdarzeń (ang. event camera). Używając kamery zdarzeń można więc w naturalny sposób usunąć statyczne tło. Na rysunku 2.1b przedstawiono przykładowy obraz rzadki uzyskany z kamery zdarzeń. Najbardziej intuicyjnym sposobem przechowywania obrazu rzadkiego wydaje się lista zawierająca współrzędne niezerowych pikseli, wraz z informacją o ich wartości (tzw. format COO, COOrdinate format). Spotykane są również inne reprezentacje macierzy rzadkich, np. zoptymalizowane pod wybór wierszy (CSR, Compressed Sparse Row array) lub wybór kolumn (CSR, Compressed Sparse Column array).

Szczególnym przypadkiem obrazu rzadkiego jest tzw. chmura punktów. Jest ona reprezentacją pewnego obiektu trójwymiarowego w postaci zbioru punktów. Każdy z punktów opisany jest informacją o położeniu w przestrzeni $\mathbb{R}^3$, wyrażone trójką uporządkowaną (x, y, z) . Dodatkowo może posiadać również informację o kolorze punktu, np. wyrażonym w modelu RGB - trójką (r, g, b) . Przykładową chmurę, liczącą n punktów, można przedstawić w postaci macierzy:



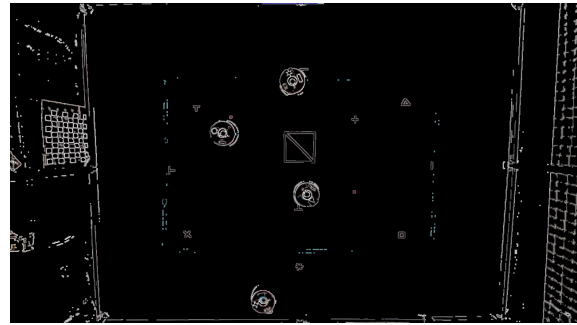
a)



b)



c)



d)

Rysunek 2.1: Obraz oryginalny (gęsty – a) oraz przykłady obrazów o wysokim stopniu rzadkości: b) obraz z kamery zdarzeń, c) obraz rzadki uzyskany poprzez usunięcie statycznego tła, d) obraz po zastosowaniu filtru krawędziowego. (Źródło: opracowanie własne)

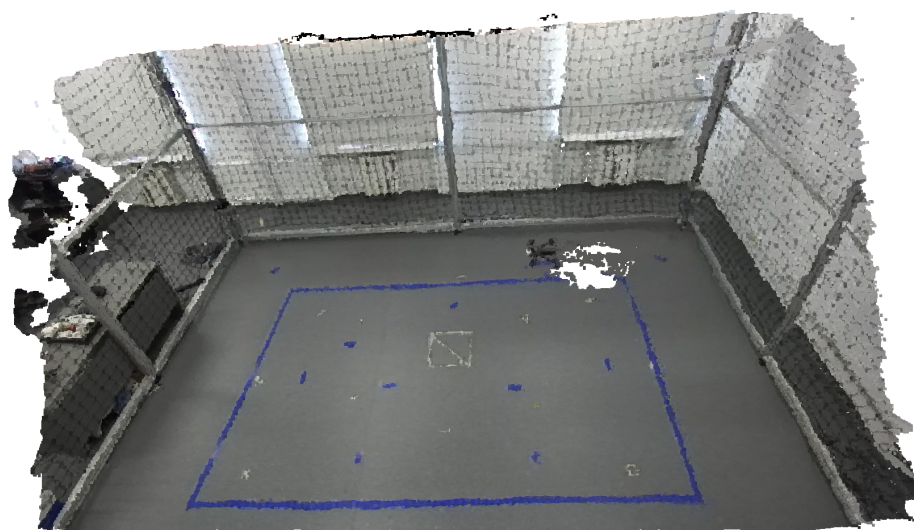
$$\left[\begin{array}{ccc|ccc} x_1 & y_1 & z_1 & r_1 & g_1 & b_1 \\ x_2 & y_2 & z_2 & r_2 & g_2 & b_2 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ x_n & y_n & z_n & r_n & g_n & b_n \end{array} \right]$$

Źródłem chmury punktów mogą być takie sensory jak kamery głębi czy lidar. Przykładową chmurę punktów pochodzącą z kamery głębi przedstawiono na rysunku 2.2.

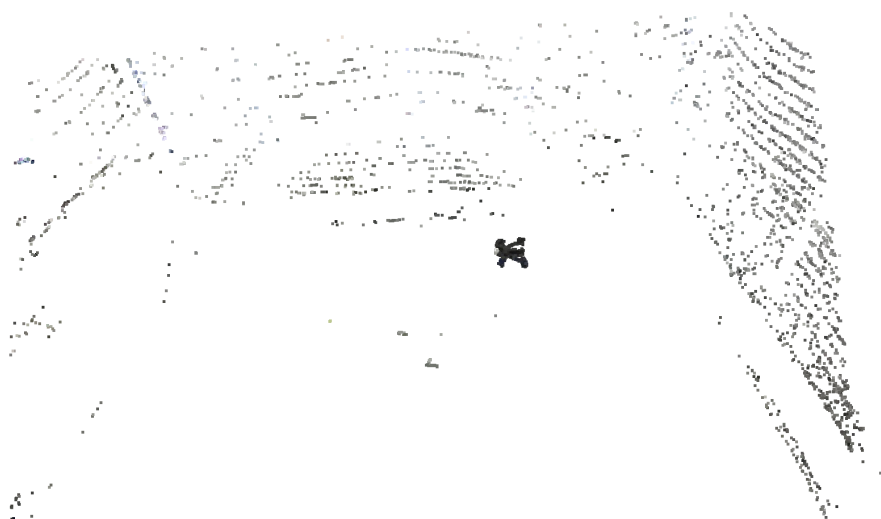
Co więcej, chmury punktów także mogą być poddawane preprocessingowi w celu redukcji liczby punktów - na przykład poprzez usunięcie tła. Efekt takiego zabiegu, wykonanego, na powyższej chmurze punktów, przedstawiono na rysunku 2.2 b).

2.1.3 Zastosowanie chmur punktów w sterowaniu eye-to-hand

W przypadku sterowania z wizualnym sprzężeniem zwrotnym eye-to-hand praca odbywa się w globalnym układzie współrzędnych, zależnym od położenia kamery. Dla zadania sterowania robotami i dronami zasadnym wydaje się przejście do innego, bardziej intuicyjnego układu, który odnosi się do przestrzeni roboczej robota. Dla tego celu konieczne jest wykonanie prostej kalibracji, której efektem jest macierz przekształcenia jednorodnego.



a)



b)

Rysunek 2.2: a) Obraz przedstawiający przykładową chmurę punktów - zarejestrowaną w trakcie badań prowadzonych w ramach niniejszej rozprawy doktorskiej oraz b) ujęcie przedstawiające tę samą scenę, po usunięciu tła. (Źródło: opracowanie własne)

Budując macierz obrotu należy wskazać dwa prostopadłe (lub zbliżone do prostopadłych) wektory v_1 i v_2 wchodzące w skład chmury punktów z kamery. Następnie należy sprawdzić ich ortogonalność i w razie potrzeby wykonać ortogonalizację np. Grama-Schmidta. Ostatnim krokiem jest obliczenie iloczynu wektorowego $v_3 = v_1 \times v_2$, co dla wektorów ortogonalnych pozwoli na wyznaczenie wektora normalnego do płaszczyzny. Umieszczenie uzyskanych wektorów w macierzy pozwoli na uzyskanie macierzy obrotu. Z kolei wektor translacji można znaleźć wyznaczając współrzędne punktu $O = (0, 0, 0)$ docelowego układu we współrzędnych kamery.

2.2 Praca z chmurami punktów

W ostatnich latach głębokie sieci splotowe [7] stały się popularnym narzędziem do przetwarzania i analizy obrazów w zadaniach takich jak klasyfikacja, detekcja czy segmentacja. Wykorzystanie tego podejścia do pracy z chmurami punktów stanowi wyzwanie, ponieważ są one nieuporządkowanym i nieregularnym zbiorem punktów [77]. Nie jest więc możliwe bezpośrednie zastosowanie splotu dyskretnego w sposób identyczny jak w przypadku obrazów dwuwymiarowych. Z tego powodu opracowano szereg metod, które umożliwiają wykorzystanie potencjału sieci głębokich (w tym splotowych) do pracy z danymi trójwymiarowymi, w szczególności z chmurami punktów [10]. Ze względu na sposób reprezentacji chmury punktów stanowiącej wejście do modelu metody te można podzielić na cztery główne grupy (por. [156]):

1. metody wielowidokowe,
2. metody oparte o wokselizację,
3. metody pracujące bezpośrednio na chmurze punktów,
4. metody stosujące hybrydową reprezentację.

Grupy te będą szczegółowo omówione w kolejnych podrozdziałach.

2.2.1 Metody wielowidokowe

Intuicyjnym rozszerzeniem podejścia stosowanego do obrazów dwuwymiarowych wydaje się wykonanie wielu dwuwymiarowych projekcji lub przecięć kształtu trójwymiarowego. Tak uzyskane obrazy dwuwymiarowe mogą zostać przekazane na wejście do sieci splotowej 2D, a wyniki – po zagregowaniu – przetworzone kolejną siecią. Przedstawiony sposób postępowania zaprezentowano po raz pierwszy w sieci Multi-view Convolutional Neural Network [118] w 2015 roku. Zastosowano w niej operacja max pooling do agregacji cech z wielu widoków, co prowadzi do utraty informacji na skutek pominięcia części pozostałych widoków.

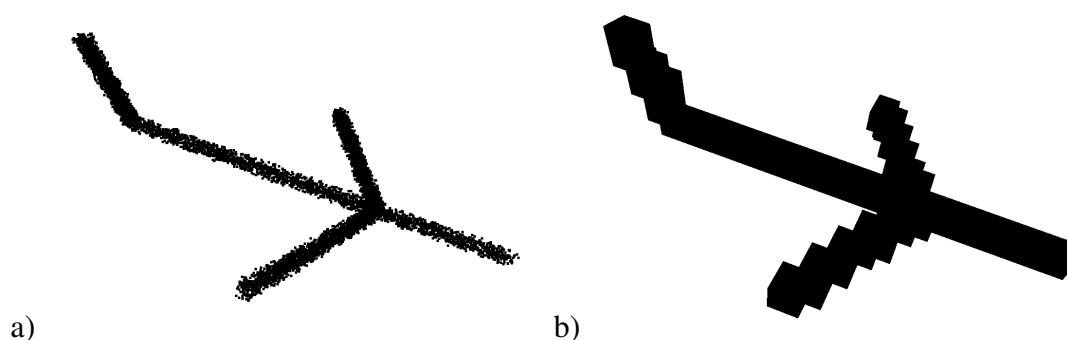
Nie jest konieczne ograniczanie się do dominującego widoku. Możliwe jest ich grupowanie w oparciu o zdolność dyskryminacyjną [33], czy też klasteryzacja i pooling widoków z tego samego klastra [131]. Efekty działania warstw splotowych mogą być agregowane również z użyciem operacji poolingu dwuliniowego (bilinear pooling) [151].

Bardziej zaawansowane podejście dla metod wielowidokowych to odejście od z góry ustalonych widoków. Spotykane są rozwiązania, w których umiejscowienie wirtualnej kamery traktowane jest jako zmienna do nauczania [42, 58] i w ten sposób wybierane są te widoki, które pozwalają na osiągnięcie najlepszych wyników predykcji.

W oparciu o przedstawione metody można wywnioskować, że istotnym ograniczeniem modeli wielowidokowych jest utrata informacji o trzecim wymiarze ze względu projekcję do dwóch wymiarów. Podejście takie pozwala natomiast na łatwe użycie wstępnie wytrenowanych modeli pracujących na danych dwuwymiarowych. Możliwa jest również łatwa implementacja metod z użyciem popularnych frameworków języka Python takich jak PyTorch [93] czy TensorFlow [2].

2.2.2 Metody wolumetryczne

Kolejną grupę metod pozwalającą na pracę z obrazami rzadkimi są metody wolumetryczne. Oparte są one na uproszczonej reprezentacji chmury punktów, uzyskiwaną jako



Rysunek 2.3: Chmura punktów przed (a) i po (b) wokselizacji. (Źródło: opracowanie własne)

efekt procedury **wokselizacji** (rysunek 2.3). Wokselizacja kojarzona może być z rasteryzacją kształtów dwuwymiarowych i polega na podziale przestrzeni na siatkę, a następnie agregacji punktów położonych wewnątrz tego samego elementu siatki. Reprezentacja taka jest trójwymiarowym odpowiednikiem tworzenia standardowych obrazy dwuwymiarowych.

Jednym z pierwszy modeli sieci splotowej wykorzystującym wokselizację dla danych trójwymiarowych był VoxNet [86] z 2015 roku. Sieć ta przyjmowała na wejście reprezentację sceny w formie siatki o rozmiarze $32 \times 32 \times 32$, co dla jednego z badanych przez autorów zbiorów odpowiadało rozmiarowi woksela $(0.1 \text{ m})^3$. Dane przekazywano do sieci w formie macierzy gęstej i dla takiej formy macierzy wykonywano również operację splotu. Sieć miała prostą strukturę – zbudowana była z dwóch warstw splotowych, jednej warstwy typu pooling oraz części liniowej.

Podejście zastosowane w VoxNecie – a więc używanie gęstej wokselizacji – nie wykorzystuje naturalnej sporej rzadkości chmur punktów, co z kolei przekłada się na dużą zajętość pamięci. Innym problemem jest koszt obliczeniowy splotu w trójwymiarowej przestrzeni – dla stałego rozmiaru sceny wraz ze zmniejszaniem rozmiaru woksela złożoność obliczeniowa rośnie z sześciannym rozmiaru. Sprawia to, że sieci bazujące na gęstej wokselizacji mają ograniczone możliwości.

W związku z tym w literaturze pojawił się szereg propozycji zrealizowania rzadkiej wokselizacji oraz modyfikacji operatora splotu. Przy rzadkich reprezentacjach, podobnie jak przy chmurach punktów, pojawia się problem szybkiego wyszukiwania elementów bliskich w sensie geometrycznym. Stąd reprezentacje takie wykorzystując znane mechanizmy indeksowania danych wielowymiarowych, przykładowo niektóre z nich oparto o drzewa ósemkowe [133], również w połączeniu z modyfikacją operatora splotu [103], lub Kd-drzewa [61, 153].

Wykorzystanie danych rzadkich, mimo że jest naturalnym podejściem, powoduje pewne problemy z utrzymaniem rzadkości w trakcie kolejnych operacji. Przykładowo w przypadku mnożenia macierzy trudno jest kontrolować stopień wypełnienia wyniku. Łatwo znaleźć przykłady, w których wynik takiej operacji będzie macierzą w pełni gęstą. Zastosowanie poolingu również zwykle zwiększa stopień wypełnienia. Podobnie jest dla splotu – pomimo, że dane przechowywane są w oszczędny (rzadki) sposób, to cechą tej operacji jest w ogólności rozmywanie i poszerzanie domeny, co również może powodować znaczne zmniejszenie rzadkości obrazu. Zjawisko to zilustrowano na rysunku 2.4.



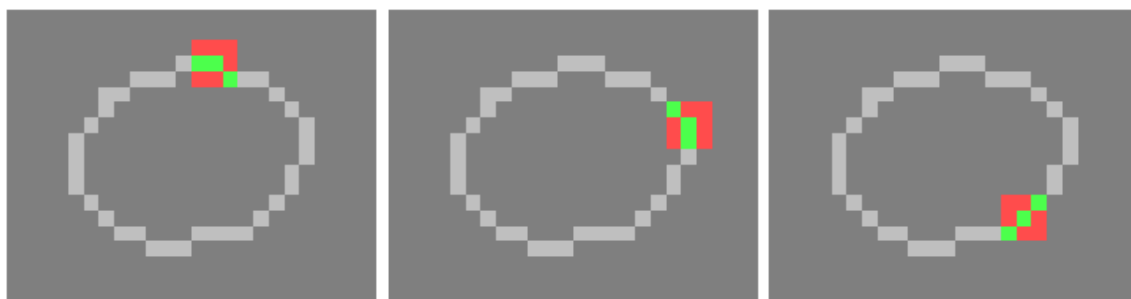
Rysunek 2.4: Efekt poddania przykładowej krzywej dwukrotnej operacji splotu – skutkuje to sukcesywnym zmniejszeniem rzadkości obrazu. (Źródło: [38])

Aby zachować lub kontrolować stopień rzadkości, zaproponowano w literaturze modyfikacje operacji splotu. Szczególnie ciekawe propozycje zmodyfikowanych operatorów splotu to Focal Sparse Convolution [17] oraz Sparse Submanifold Convolution (SSC) [38]. Sporą popularność zyskał ten drugi operator, zaprezentowany w 2017 roku. Od typowego splotu różni się tym, że obliczenia są wykonywane wyłącznie wtedy, gdy środek jądra splotu pokrywa się z niezerowym elementem obrazu (rysunek 2.5). Skutkuje to nie tylko przyspieszeniem obliczeń, ale także zachowaniem rzadkości obrazu, nawet po kilkukrotnym zastosowaniu takiego operatora.

Operator ten z powodzeniem stosowany jest m. in. w zadaniach detekcji [149, 155, 161], niekiedy w połączeniu z jednoczesną wokselizacją i pillaryzacją [47].

Dane przechowywane w postaci wokseli są poddawane analizie także z użyciem transformerów [85, 128], niekiedy dodatkowo w połączeniu z Focal Sparse Convolution [160]. Spotkać też można rozwiązania, w których rekurencyjna sieć LSTM [45] połączona jest z siecią splotową 3D opartą o wokselizację [76].

Istotną wadą stosowania wokselizacji jest utrata dokładności reprezentacji chmury punktów. Taki sposób przedstawienia chmury punktów pozwala natomiast na stosunkowo łatwe rozszerzenie operacji splotu dwuwymiarowego na trójwymiarowy. Umożliwia to – podobnie jak w przypadku sieci dwuwymiarowych – uzyskanie interpretowalności modelu. Jest to możliwe dzięki zachowaniu wymiarowości przestrzeni, co pozwala na jej łatwą wizualizację po każdej warstwie sieci. Stosowanie operacji zachowujących rzadkość,



Rysunek 2.5: Ilustracja pola percepcyjnego operatora Sparse Submanifold Convolution. (Źródło: [37])

takich jak SSC pozwala na dużą oszczędność pamięci i czasu obliczeń.

Sieci korzystające z wokselizacji można budować z użyciem PyTorch i Tensorflow. Ponieważ API PyTorch dla danych rzadkich jest w fazie beta i ma ograniczoną funkcjonalność, dla rzadkiego spłotu oraz operatorów takich jak SSC wymagane jest użycie innych bibliotek. Zaliczają się do nich SparseConvNet [39], Minkowski Engine [19], SpConv [116] oraz będąca wciąż w fazie rozwoju TorchSparse++ [121].

2.2.3 Metody pracujące bezpośrednio na chmurze punktów

Szeroko badana w ostatnich latach jest grupa metod, które przygotowane są do pracy bezpośrednio na chmurze punktów, bez wokselizacji czy też projekcji. Przy takim podejściu pojawiają się dodatkowe wymagania – przykładowo sieć powinna być odporna na permutację punktów.

Metody te zwykle wymagają stałej liczby punktów wejściowych i zazwyczaj wykonują próbkowanie punktów lub klasteryzację, które - podobnie jak wokselizacja - są również pewną formą utraty dokładności reprezentacji chmury punktów.

Przy reprezentacji danych bezpośrednio za pomocą chmury punktów kluczowe zagadnienie polega na określaniu sąsiedztwa oraz indeksowaniu przestrzennym danych. Osobnym zagadnieniem jest fakt, że do określenia spłotu potrzebna jest zwykle bardziej regularna postać danych niż przypadkowo rozmieszczone punkty. W związku z tym w literaturze określono wiele sposobów reprezentacji danych oraz definicji spłotu, kładąc w różnym stopniu nacisk na wydajność oraz wierne odwzorowanie operacji spłotu. Tematyka ta jest intensywnie rozwijana w ostatnich latach.

Historycznie pierwszą metodą z tej grupy jest sieć PointNet [13] z 2017 roku. Istotną cechą tej architektury jest wykorzystanie perceptronów wielowarstwowych (MLP) do ekstrakcji cech oraz symetrycznej funkcji (takiej jak max pooling) do uzyskania jednowymiarowego wektora cech opisującego chmurę punktów.

Lokalna agregacja cech

Podejście zaprezentowane przez autorów PointNetu pozwala na uzyskanie globalnej agregacji cech, co nie zawsze jest pożądanym efektem. Z tego powodu szeroko badane były możliwości uzyskania lokalnej agregacji cech. Zrealizowano ją rozszerzając metodę pierwotną i tworząc sieć PointNet++ [98]. W sieci tej, po etapie próbkowania i grupowania chmury punktów, użyto sieci PointNet dla każdej z grup punktów. W ten sposób nie można było jednak znaleźć informacji o wpływie poszczególnych punktów na inne punkty w klastrze, co rozwiązano w sieci PointWeb [158].

W niektórych rozwiązaniach kładziony jest nacisk na reprezentację chmury punktów, ponieważ losowe próbkowanie może prowadzić do negatywnych efektów, wymagających zniwelowania [46]. Zamiast tego można zastosować inny rodzaj próbkowań wykorzystujący porządkowanie punktów w oparciu o krzywe wypełniające powierzchnie (jak np. krzywa typu Z-order stosowana w sieci [16]). Podejście takie znane jest z algorytmów grafiki komputerowej i pozwala na przechowywanie blisko siebie w pamięci (i na krzywej) punktów, które są geometrycznie bliskie. Inne stosowane metody to uproszczenie reprezentacji chmury punktów poprzez zastosowanie mapy samoorganizującej [62, 68], czy też podział danych na kontur oraz wnętrze [144].

Próba użycia sieci spłotowej bezpośrednio na chmurze punktów wymaga określenia

i modyfikacji operacji splotu, która dostosowane będą do pracy z takim typem danych. Jedną z możliwości jest uciąganie chmury punktów poprzez aproksymację określonej na niej funkcji i zastosowanie ciągłej operacji splotu [8]. Rozszerzeniem tego podejścia jest posłużenie się sferycznymi funkcjami jądra [97]. Podobną techniką, wykorzystującą uciąganie operacji splotu oraz reprezentację jądra za pomocą sieci MLP, a także aproksymację chmury punktów za pomocą estymacji jądrowej jest operator PointConv [141]. Nieco odmienną metodą obliczania splotu dla chmury punktów jest transformacja otoczenia rozpatrywanego punktu do takiej postaci, która pozwala na zastosowanie typowego splotu. Można to osiągnąć z użyciem MLP [71] lub wybierając najbliższych k punktów [63]. Niektóre architektury realizujące splot dla chmur punktów umożliwiają ekstrakcję cech opisujących geometryczne zależności między punktami [79, 101], m. in. z użyciem wielomianów Taylora [145].

Inne podejście do zagadnienia lokalnej agregacji cech to wykorzystanie sieci grafowych, w tym grafowych sieci splotowych [12, 112]. W takich sieciach, oprócz samej chmury punktów, określona jest struktura sąsiedztw określająca bliskość tych punktów. Struktura ta może podlegać uczeniu lub nie. Wśród sieci grafowych można wyróżnić dwie kategorie [67]: spektralne i przestrzenne. W podejściach spektralnych [60] konwolucje grafowe wykorzystują rozkład względem wartości własnych macierzy Laplace’a skojarzonej z grafem (ang. Laplacian matrix - jest różnicą macierzy diagonalnej określającej rząd wierzchołka i macierzy sąsiedztwa). Przykładową siecią bazującą na podejściu spektralnym jest PointWavelet [139]. Z kolei sieci przestrzenne wyznaczają sploty bezpośrednio z macierzy sąsiedztw grafu, jak np. w sieci PointNGCNN [83].

Wadą pierwszych przestrzennych grafowych sieci splotowych, takich jak ECC [115], była niezmienność grafu sąsiedztwa w kolejnych warstwach sieci. Odejście od takiego mechanizmu zaprezentowano w architekturze Dynamic Graph CNN (DGCNN) [136]. W każdej warstwie graf generowany jest dynamicznie w oparciu o algorytm k -NN, a cechy są liczone w oparciu o autorski operator EdgeConv, stosowany na grafie rozpiętym na lokalnym sąsiedztwie danego punktu. Architektura ta zyskała popularność i była rozwijana w różnych wersjach np. [157], czy też w [135], gdzie wykorzystano ją do jednoczesnej detekcji wielu obiektów. Innym kierunkiem rozwoju przestrzennych grafowych sieci splotowych jest definiowanie ich w taki sposób, by jądro splotu było adaptacyjne [138]. W niektórych architekturach możliwe jest uczenie nie tylko wag jądra, ale i jego kształtu [75].

W pracy z chmurami punktów zastosowanie znalazł również mechanizm uwagi (ang. Attention), pierwotnie stosowany w modelach językowych. Użyto go m. in. w strukturze PointTransformer [159] czy w Point Attention Transformer [147]. W tym drugim modelu użyto tzw. mechanizm Group-Shuffle Attention, który jest mniej kosztowny obliczeniowo niż Multi-Head Attention. Zaproponowano także metodę próbkowania chmury punktów, którą można uczyć. Stosowane bywają także modyfikacje mechanizmu uwagi, takie jak zastąpienie przestrzennego okna uwagi poprzez trzy ortogonalne (dwuwymiarowe) płaszczyzny [28]. W niektórych podejściach używane jest również połączenie mechanizmu uwagi z grafowym splotem [15, 132].

Globalna agregacja cech

Nieco węższą grupę stanowią metody używające globalnej agregacji cech. Zaliczają się do nich m. in. modele oparte o architekturę transformera. Jednym z nich jest Point Transformer [29], w którym zastosowano trenowalny moduł do wyboru najważniejszych lokalnych cech oraz lokalno-globalny mechanizm uwagi. Znana z modelu BERT [24] koncepcja maskowania również może zostać przeniesiona na chmury punktów [91, 152]. Spotykane są także rozwiązania takie jak obliczenie samouwagi na wybranych punktach [11], kilkukrotne stosowanie tego mechanizmu dla mapy cech w różnych skalach [43] czy modyfikacje jego działania [41]. Podobnie jak w przypadku agregacji lokalnej, transformery bywają łączone z sieciami grafowymi [81, 99, 130], niekiedy z jednoczesną modyfikacją mechanizmu uwagi [82].

Na potrzeby globalnej agregacji cech stosowany bywa również moduł globalny [134]. Jednym z przykładów jego użycia jest sieć PointASNL [146], wykorzystująca adaptacyjne próbkowanie punktów. Podejście takie jest odporniejsze na szum, który towarzyszy często danym rzeczywistym. Można też stosować agregację punktów leżących na krzywych [143].

Rekurencyjne sieci neuronowe również mogą być stosowane w pracy z chmurami punktów, w szczególności w połączeniu z innymi metodami, np. PointNet [30], PointNet++ [137] lub mechanizmem uwagi [78].

Podsumowując: metody pracujące bezpośrednio na chmurze punktów są ciekawym kierunkiem badań. W przeciwieństwie do metod opartych na splocie dla typowych obrazów rastrowych 2D i 3D, pozwalają na stosowanie globalnej agregacji cech, której wadą jest jednak koszt obliczeniowy. Metody pracujące na chmurze punktów można implementować bezpośrednio z użyciem biblioteki PyTorch. Pewna liczba procedur wspomagających obliczenia na chmurach punktów oraz sieciach grafowych została zebrana w bibliotece PyTorch Geometric [34].

2.2.4 Metody oparte o hybrydową reprezentację

W pracy z chmurami punktów stosowane są również metody oparte o hybrydową reprezentację danych. Przykładem takiego podejścia jest PointGrid [66], gdzie w każdym punkcie siatki zawarta jest taka sama liczba punktów. Spotyka się również sieci pozwalające na łączenie reprezentacji danych w postaci chmury punktów i wielowidokowej [150] oraz wyrenderowanego obrazu 2D z obrazem 3D [4]. Innym podejściem jest jednoczesne używanie reprezentacji w postaci wokseli oraz chmury punktów [80, 89, 154]. Metodę taką można stosować między innymi wykorzystując PointNet [114] czy też PointNet++ [55]. Reprezentację punktowo-wokselową spotkać można również w przypadku sieci grafowych.

2.3 Omówienie wybranych architektur

W oparciu o wykonany przegląd literatury wybrano kilka metod, które stanowiły rozwiązania referencyjne w części eksperymentalnej. Wyselekcjonowane zostały następujące modele, pracujące bezpośrednio na chmurze punktów:

- **PointNet** – jako historycznie pierwszą metodę pracującą na chmurze punktów,
- **DGCNN** – jako przykład sieci grafowej wykorzystującej lokalną agregację cech,

- **PointCloudTransformer** – jako przykład metody wykorzystującej globalną agregację cech oraz jako przykład transformera.

Powyższe modele pracują na chmurach punktów o stałej wymiarowości. Co za tym idzie, dla zmiennej liczby punktów w poszczególnych klatkach konieczne jest próbkowanie danych.

2.3.1 PointNet

PointNet (rysunek 2.6) to historycznie pierwsza sieć głęboka, która pozwala na pracę bezpośrednio na chmurze punktów, bez wykonywania projekcji czy też wokselizacji. Przyjmuje ona na wejściu macierz o wymiarze $n \times 3$, gdzie n to liczba punktów, opisywanych przez trzy cechy. Liczbę cech można zmodyfikować w razie potrzeby, na przykład dodając informację o kolorze w formacie RGB.

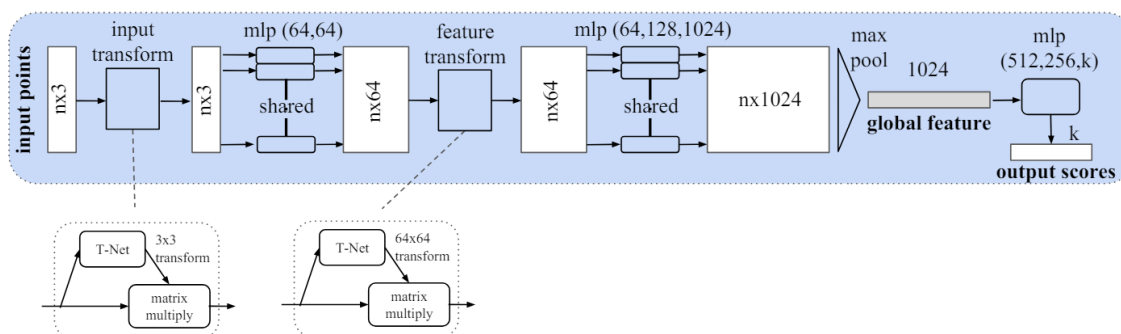
Ważnym elementem sieci jest moduł T-net, który znajduje macierz przekształcenia afinicznego o wymiarze 3×3 . Przekształcenie to jest wykonywane, by sprowadzić wejściową chmurę punktów do przestrzeni kanonicznej. Takie działanie ma zapewnić translacyjną i rotacyjną niezmienniczość cech. Podobne działanie wykonywane jest w celu wyrównania cech (macierz przekształcenia ma wymiar 64×64). Oprócz tego w sieci wykorzystywane są standardowe warstwy gęste.

W końcowym segmencie sieci wykonywana jest agregacja cech, której efektem jest 1024-elementowy wektor opisujący całą chmurę punktów. Do agregacji wykorzystywana jest symetryczna funkcja, taka jak max pooling.

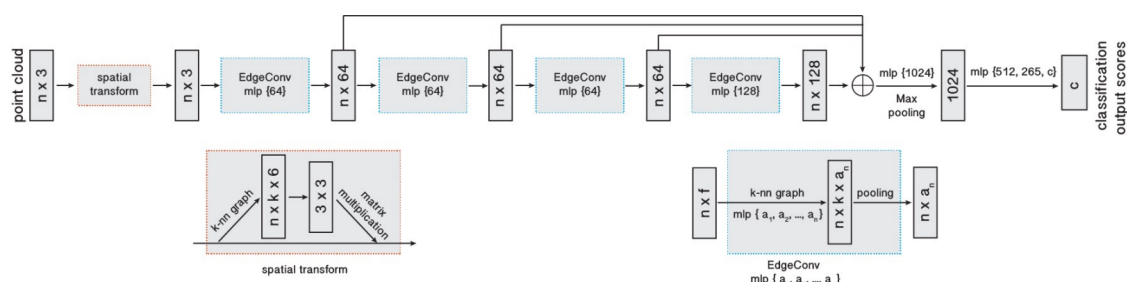
2.3.2 Dynamic Graph CNN

Dynamic Graph CNN (rysunek 2.7) to grafowa sieć przestrzenna, wykorzystująca lokalną agregację cech. Architektura ta oparta jest o operator EdgeConv, który ma stanowić odpowiednik splotu dla struktur grafowych. Dodatkową innowacją, w stosunku do wcześniejszych grafowych sieci splotowych, jest aktualizacja grafu lokalnego sąsiedztwa po każdej warstwie. Ma to na celu dążenie do tworzenia grafu w ten sposób, by obejmował punkty podobne semantycznie.

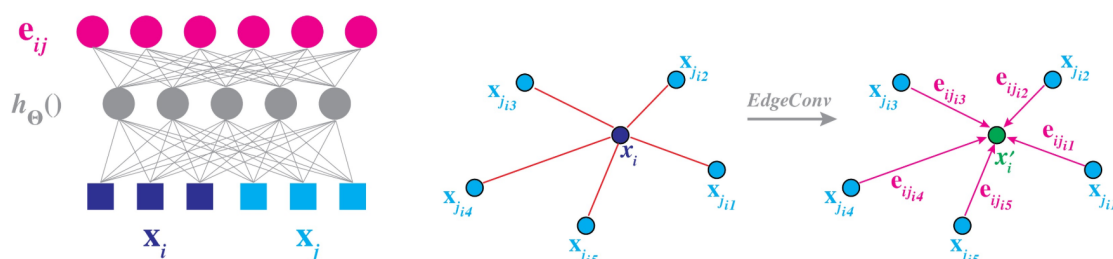
DGCNN przyjmuje na wejście chmurę punktów o wymiarze $n \times f$, gdzie f to liczba



Rysunek 2.6: Architektura sieci PointNet dla zadania klasyfikacji. (Źródło: modyfikacja obrazu z pracy [13])



Rysunek 2.7: Architektura sieci DGCNN dla zadania klasyfikacji. (źródło: modyfikacja obrazu z pracy [136])



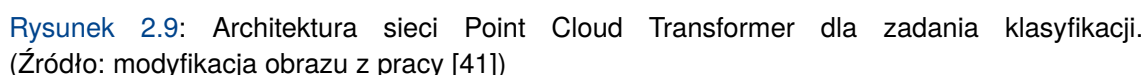
Rysunek 2.8: Ilustracja działania operatora EdgeConv. Po lewej: Obliczanie cechy krawędziowej e_{ij} dla pary punktów x_i oraz y_j . Po prawej: działanie operatora na grafie – agreguje on cechy krawędziowe powiązane ze wszystkimi krawędziami wychodzącymi z każdego połączonego wierzchołka. (Źródło: na podstawie [136])

cech. Chmura ta jest następnie poddawana – podobnie jak w PointNecie – transformacji do przestrzeni kanonicznej. Macierz przekształcenia (o wymiarze 3×3) jest estymowana z użyciem wektora, który zawiera informację o współrzędnych wszystkich punktów, a także informację o różnicy współrzędnych między ich k najbliższymi punktami.

Przekształcona chmura punktów jest następnie kilkakrotnie poddawana operacji EdgeConv, a wszystkie wyniki są łączone i agregowane z użyciem Max Poolingu i liniowych warstw gęstych do jednowymiarowego wektora cech (rysunek 2.8).

Dokładniejszego omówienia wymaga operator EdgeConv (rysunek 2.8), który ma stanowić grafowy odpowiednik operacji splotu. Operacja ta polega na obliczeniu wartości cechy dla wybranego punktu x_i w oparciu o wartości sąsiednich punktów — a więc jest to analogia do działania splotu na sąsiednich pikselach. Lokalne sąsiedztwo punktu x_i wybierane jest z użyciem algorytmu kNN, a następnie jest na nim rozpinany skierowany graf. Dla każdej pary punktów x_i oraz x_j liczona jest cecha krawędziowa z użyciem warstw gęstych. Po obliczeniu cech krawędziowych dla wszystkich par punktów jest wykonywana ich agregacja z użyciem symetrycznej funkcji – można użyć operatora uśredniającego lub max pooling.

Autorzy metody dowiedli w pracy [136], że metoda jest permutacyjnie niezmiennicza oraz częściowo niezmiennicza dla translacji.



Sieć Point Cloud Transformer (rysunek 2.9) jest tak naprawdę enkoderem i podobnie jak wcześniejsze modele przyjmuje wejście w postaci chmury punktów o wymiarze $n \times f$. Pierwszym elementem architektury PCT jest warstwa kodująca wejście do przestrzeni o większej wymiarowości, w której łatwiej wyrazić zależności między punktami. Tak przetworzona chmura punktów stanowi wejście do sekwencji czterech modułów uwagi, których zadaniem nauczanie się bogatej semantycznie i rozróżniającej reprezentacji dla każdego punktu. Wyjścia z wszystkich warstw uwagi są łączone i przekazywane na wejście do warstwy liniowej, a następnie poddawane max- i average poolingowi. Ostatnią częścią sieci jest sekwencja gęstych warstw liniowych.

Na rysunku 2.10 przedstawiono architekturę modułu uwagi. Elementy oznaczone jako switch wskazują miejsca, w których można dokonać zmiany, by uzyskać mechanizm samouwagi (linie przerywane) lub Offset-Attention (linie ciągłe). Główna różnica mechanizmu Offset-Attention w stosunku do standardowej samouwagi jest taka, że wyjście z opisywanego modułu jest offset (różnica) pomiędzy cechami uzyskanymi przez samouwagę a cechami wejściowymi.

3. Architektury sieci dedykowane predykcji lokalizacji i orientacji

W niniejszym rozdziale zawarto opis architektur rzadkich splotowych sieci neuronowych, które zaprojektowano w ramach badań. Sieci te dedykowane są zadaniom lokalizacji i orientacji i pracowały (poza jednym wyjątkiem) z obrazami trójwymiarowymi poddanymi wokselizacji.

Zestawienie wykorzystanych w pracy modeli przedstawiono w tabeli 3.1. Umieszczono w niej również informacje o modelach referencyjnych, które wykorzystano w części eksperymentalnej. Badania wykonywane były w środowisku języka Python z wykorzystaniem frameworka Pytorch [93]. Pakiet spconv dostarczył implementacje warstw rzadkich, w szczególności rzadkie warstwy splotowe (SSC) oraz rzadkie warstwy poolingu. Dla celów weryfikacji jednej z hipotez badawczych wybrane architektury zostały przygotowane w dwóch wariantach tj. z warstwami rzadkimi oraz gęstymi.

Sieci te składają się z warstw obsługujących dane reprezentowane w sposób rzadki, jak również z warstw dostosowanych dla danych gęstych. Wybór miejsca w sieci, w którym przechodzi się z reprezentacji rzadkiej na gęstą, był również przedmiotem badań. Przejście do reprezentacji gęstej jest zwykle konieczne, gdyż na końcowych etapach przetwarzania (często po zastosowaniu wielu operacji poolingu) cechy posiadają coraz większy stopień wypełnienia co powoduje, że zysk z operacji wykonywanych w sposób rzadki staje się

Tabela 3.1: Zestawienie modeli wykorzystanych w badaniach.

Model	Architektura	Liczba parametrów
LineCNN2D	autorska	416040
SlicingCNN3D	autorska	4017966
SimpleCNN3D	autorska	128193
DroneCropCNN3D	autorska	114178
ProjectionCNN3D	autorska	326529
PointNet	referencyjna	3463376
Dynamic Graph CNN	referencyjna	1764567
PointCloudTransformer	referencyjna	2735751

niewielki.

W przypadku większości modeli starano się je projektować w sposób wykorzystujący wiedzę o rozwiązywanym zadaniu. Przykładowo dla jednej z sieci dla zadania lokalizacji wykonywano projekcję trójwymiarowego wyjścia z warstwy splotowej na trzy płaszczyzny: XY , XZ , YZ . Konstruując sieci badano też wizualnie cechy uzyskiwane na poszczególnych etapach przetwarzania, czyli wyjścia z poszczególnych warstw. Interesujące jest sprawdzenie, czy cechy te stają się niezależne oraz czy są skorelowane z cechami badanych obiektów. Przykładowo dla jednej z sieci rozpoznających kąt obrotu drona zaobserwowano efekt rozdzielania wejściowego obrazu na korpus drona oraz jego ruchomą kamerę pokładową.

3.1 LineCNN2D

Sieć LineCNN2D to jedyna zaprojektowana sieć, która pracuje z obrazami dwuwymiarowymi. Na wejściu przyjmuje jednokanałowy (w skali szarości) obraz w rozdzielczości 4K (3840×2160). Sieć była uczona na podstawie syntetycznie generowanego zbioru obrazów, przedstawiających linię, która ma na jednym z końców dołączony okrąg. Zadaniem sieci było przewidzenie:

- kąta, pod jakim obrócony jest obiekt,
- pozycji x i y obiektu we współrzędnych obrazu,
- grubości obiektu.

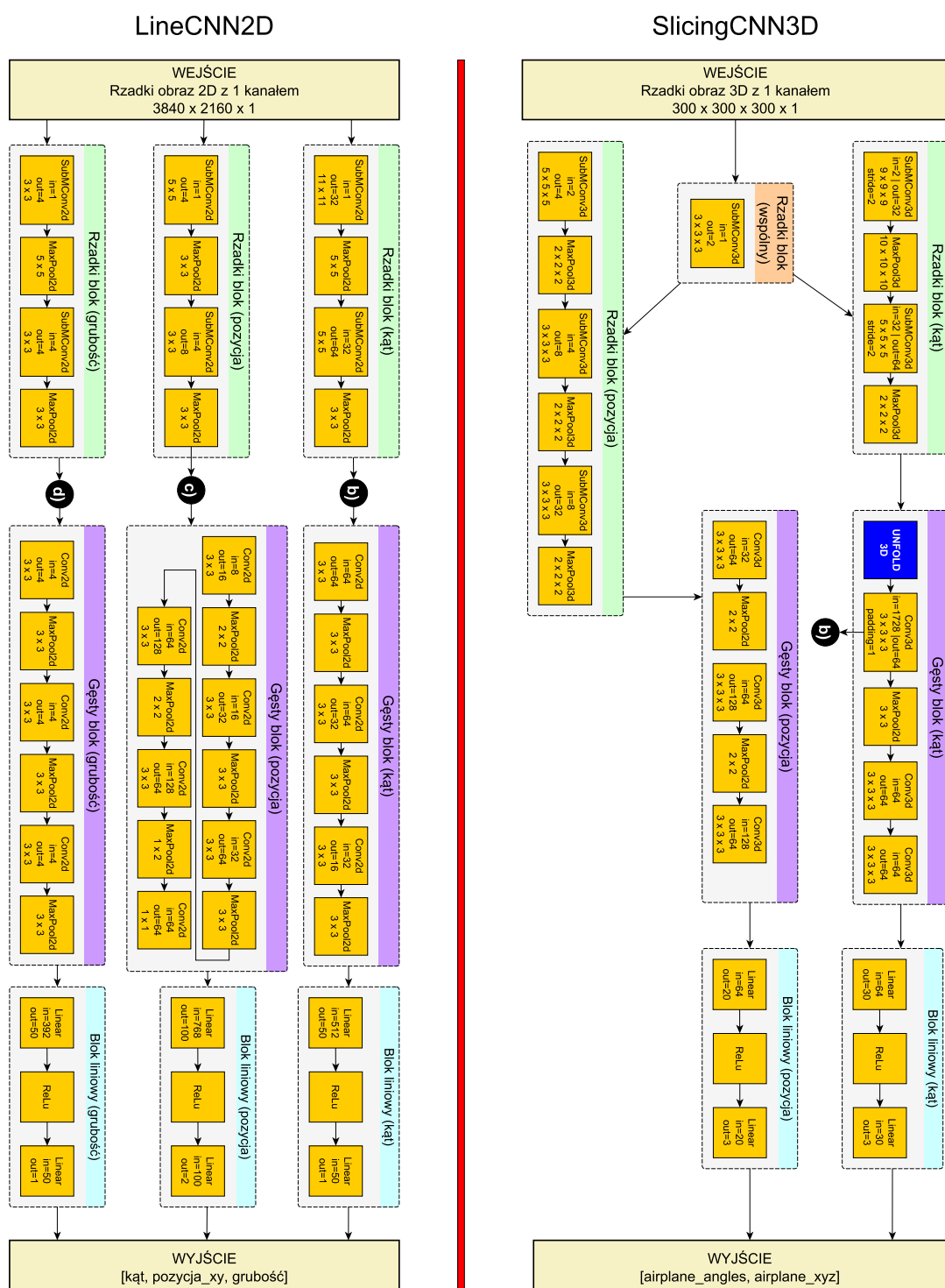
W architekturze tej (rysunek 3.1 po lewej) wyróżnić można trzy gałęzie, odpowiadające wyżej wymienionym zadaniom. Każda z gałęzi składa się z części rzadkiej, wykorzystującej SSC, części gęstej, która zawiera typowe operacje splotu, oraz części linowej, która dokonuje predykcji w oparciu o wyekstrahowane cechy.

Dla podsieci odpowiedzialnej za predykcję kąta zastosowano splot o stosunkowo dużym rozmiarze jądra (11×11). Kierowano się przy tym ideą objęcia możliwie największej części obiektu filtrem splotowym. Z podobnych powodów bezpośrednio po pierwszej warstwie splotowej wykonano max pooling o dużym rozmiarze (5×5), oraz splot z jądrem 5×5 .

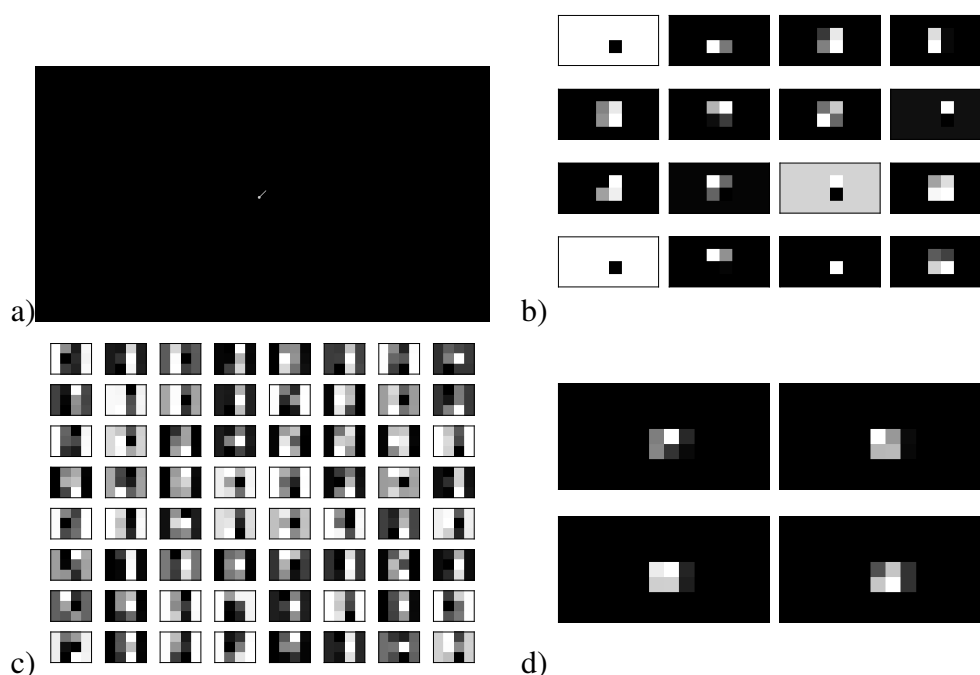
Dla zadania lokalizacji użyto natomiast początkowo mniejszy filtr (3×3), a redukcję wymiarowości wykonywano za pomocą wielokrotnego stosowania max pooling o nie-dużym rozmiarze (3×3 lub 2×2). Jedną z operacji max pooling (o wymiarze 1×2) zmieniła kształt obrazu tak, by był on bliższy kwadratowemu. Korzystając z warstw splotowych stworzono natomiast dużą liczbę kanałów (maksymalnie 128). Z kolei dla zadania predykcji grubości obiektu stworzono mało rozbudowaną architekturę, w której najwyższą uzyskana liczba kanałów to cztery.

Na rysunku 3.2 przedstawiono a) przykładowe wejście do sieci oraz wyjścia z rzadkich bloków warstw splotowych - odpowiednio b) kąta, c) pozycji, d) grubości. Na schemacie sieci zaznaczono miejsca, z których pozyskano narysowane wyjścia.

Na potrzeby dowodzenia tezy stworzono odpowiednik tej architektury, który wykorzystuje wyłącznie typowe operacje splotu i pracuje z danymi gęstymi.



Rysunek 3.1: Schematy sieci: LineCNN2D (po lewej) oraz SlicingCNN3D (po prawej). (źródło: opracowanie własne)



Rysunek 3.2: Architektura LineCNN2D: a) przykładowe wejście do sieci oraz wyekstrahowane cechy przez podsieci b) dla kąta c) dla pozycji d) dla grubości (cechy wskazano na rys. 3.1) (Źródło: opracowanie własne)

3.2 SlicingCNN3D

SlicingCNN3D (rysunek 3.1 po prawej) to sieć przyjmująca na wejście trójwymiarowy rzadki obraz jednokanałowy o rozdzielczości $300 \times 300 \times 300$. Sieć była trenowana na danych syntetycznych, gdzie rozpoznawany był obiekt o kształcie samolotu. Zadaniem sieci jest pomiar położenia obiektu oraz predykcja jego kątów obrotu w osiach X , Y i Z .

W tym celu obraz wejściowy jest przetwarzany przez wspólną warstwę splotową, z której wyjście stanowi obraz dwukanałowy. Wstępnie przetworzony obraz stanowi wejście do dwóch podsieci – oddzielnej dla pozycji oraz oddzielnej dla kąta.

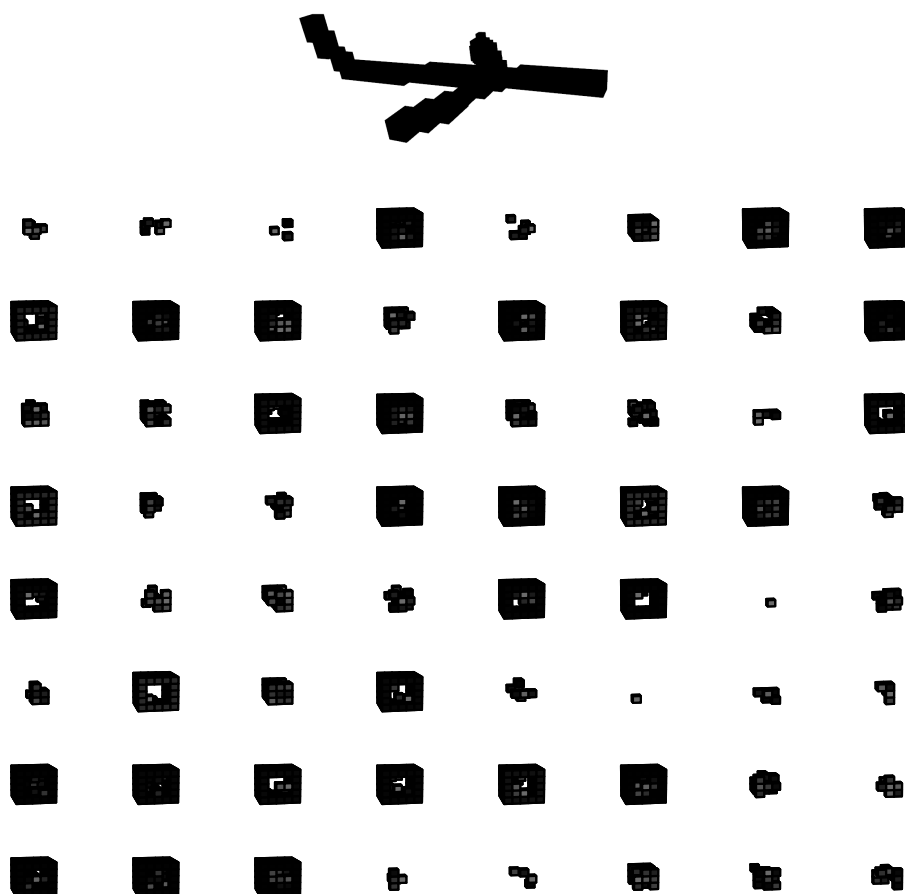
Podsieć dla pozycji złożona jest z dwóch części splotowych (rzadkiej i gęstej) oraz części liniowej. W częściach splotowych (za wyjątkiem pierwszej warstwy) stosowane są małe filtry – o rozmiarze 3×3 . Użycie splotów oraz max pooling pozwala na uzyskanie 64 cech, na podstawie których część liniowa wykonuje predykcję położenia drona w osiach X , Y i Z .

Podsieć dla kąta również złożona jest z dwóch części splotowych oraz części liniowej. Różni się jednak ona istotnie w sposobie przetwarzania obrazu. W części rzadkiej użyto operatora splotu SSC z rozmiarem filtra $10 \times 10 \times 10$, co ma pozwolić na uzyskanie wysokich wartości cech dla tego obszaru obrazu, w którym położony jest obiekt. Z kolei w części gęstej wykonywana jest operacja unfold 3D, która polega na podziale trójwymiarowego obrazu na nieduże fragmenty sześciennie. Docelowo, w jednym (lub kilku z nich) powinien znaleźć się rozpoznawany obiekt. Operacja unfold zaniedbuje informacje

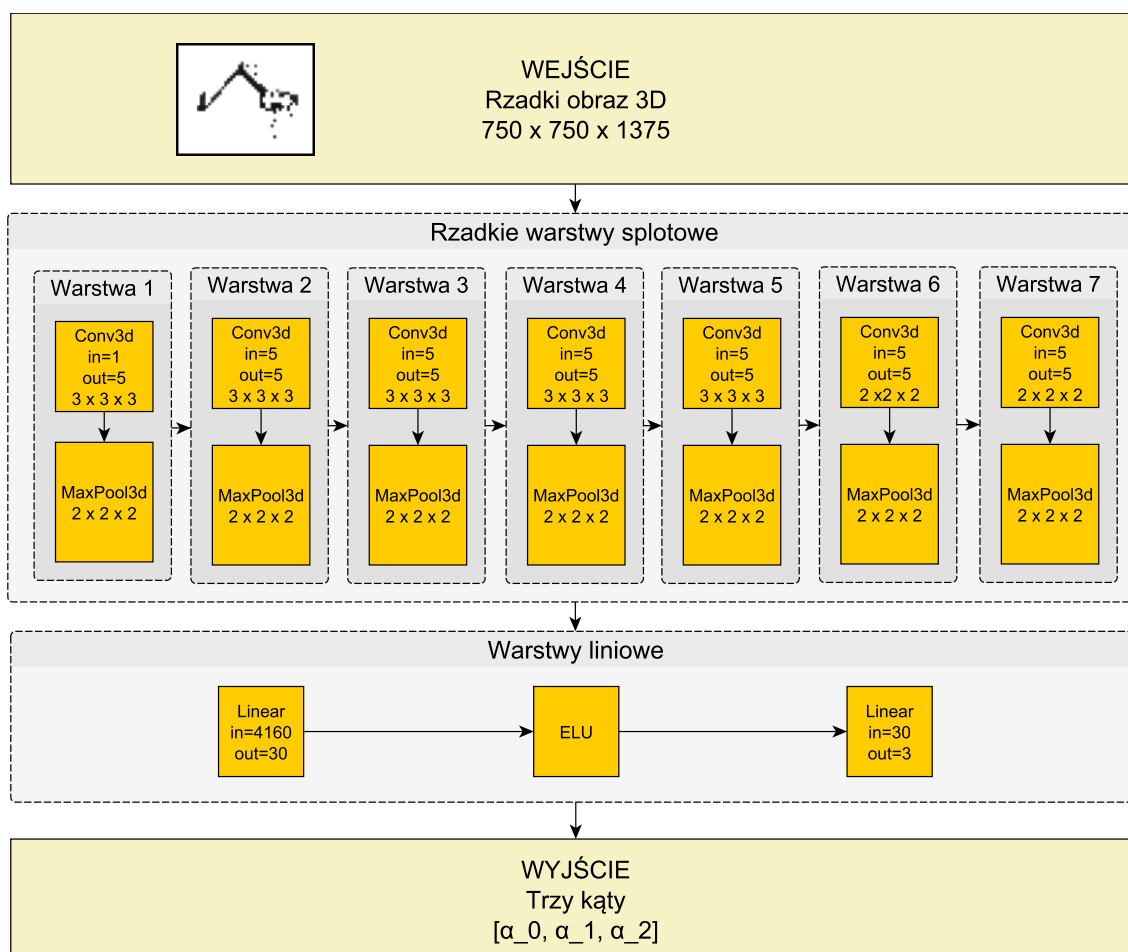
o położeniu, natomiast wzmacnia informacje o orientacji. Finalnie uzyskiwane są 64 cechy, z użyciem których część liniowa przewiduje kąty obrotu drona względem osi układu współrzędnych. Zastosowanie operacji unfold znacznie podniosło dokładność przewidywania kąta. Niestety operacja ta nie została zaimplementowana w postaci rzadkiej (nie jest dostępna w pakietach `torch.sparse` oraz `spconv`), stąd konieczne było jej wykonanie w postaci gęstej (z użyciem pakietu PyTorch), co powoduje niepotrzebny narzut pamięciowy. Optymalne byłoby wykonanie tej operacji w postaci rzadkiej, a następnie przejście do postaci gęstej po zmniejszeniu liczby kanałów.

Na rysunku 3.3 przedstawiono a) przykładowe wejście do sieci oraz b) efekt działania operacji unfold po przetworzeniu cech z użyciem jednej warstwy splotowej.

Podobnie jak dla poprzedniego modelu, stworzono odpowiednik tej sieci pracujący wyłącznie z danymi gęstymi.



Rysunek 3.3: Przykładowe wejście do sieci SlicingCNN3D oraz efekt działania splotu po operacji unfold (prezentowane cechy wskazano na rysunku 3.1) (Źródło: opracowanie własne)



Rysunek 3.4: Schemat sieci SimpleCNN3D. (Źródło: opracowanie własne)

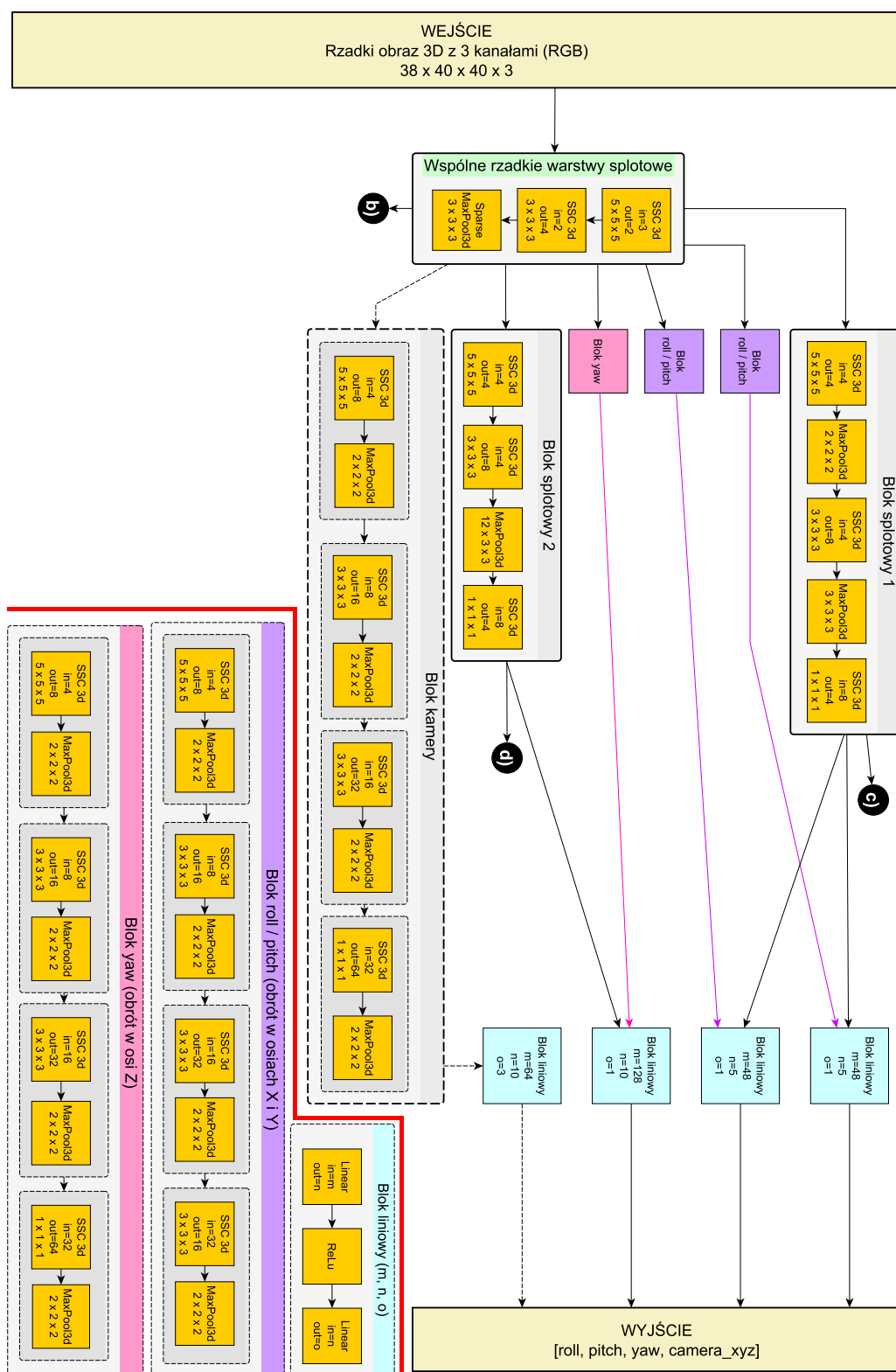
3.3 SimpleCNN3D

Kolejna z przedstawianych architektur to SimpleCNN3D (rysunek 3.4), której zadaniem jest predykcja kątów pomiędzy członami ramienia robota na podstawie obrazu trójwymiarowego o rozdzielczości $750 \times 750 \times 1375$. Warto zwrócić uwagę, że dane gęste przy takiej rozdzielczości miałyby rozmiar rzędu $7 \cdot 10^8$. Sieć ta ma stosunkowo prostą budowę i nie-dużą liczbę parametrów (128193). Jest to wynik założenia przyjętego przy projektowaniu – sieć powinna umożliwiać predykcję z szybkością kilkudziesięciu próbek na sekundę przy wykorzystaniu wyłącznie jednego rdzenia procesora.

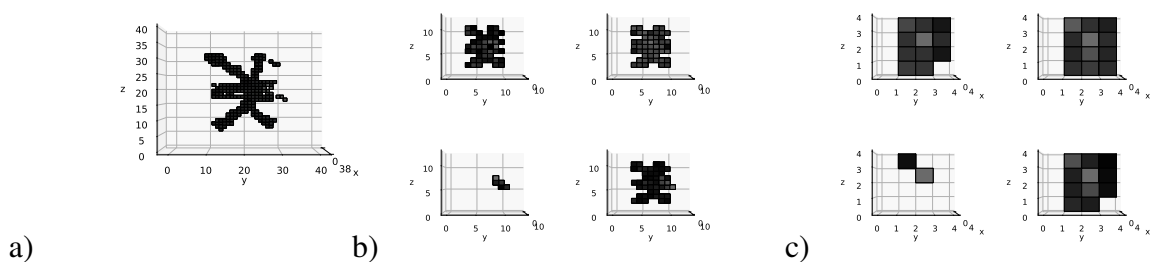
Sieć zbudowana jest z sekwencji siedmiu bloków, z których każdy zawiera rzadką warstwę splotową i warstwę maxpooling. Finalnie uzyskiwany jest wektor 4160 cech opisujących obraz. Stanowi on wejście do części liniowej, która przewiduje kąty charakteryzujące ułożenie członów robota.

3.4 DroneCropCNN3D

DroneCropCNN3D (rysunek 3.5) jest siecią przyjmującą na wejście trzykanałowy obraz



Rysunek 3.5: Architektura sieci DroneCropCNN3D. (Źródło: opracowanie własne)



Rysunek 3.6: Przykładowe wejście oraz wyjścia z wybranych warstw sieci DroneCropCNN3D. (Źródło: opracowanie własne) .

trójwymiarowy o wymiarze $38 \times 40 \times 40$, który przedstawia wyciętego i skadowanego drona. Zadaniem sieci jest ustalenie orientacji drona, tzn. przewidzenia kąta obrotu w osiach X , Y i Z . Dodatkowo, na potrzeby wzmocnienia przewidywania kąta obrotu w osi Z , sieć przewiduje położenie kamery zamontowanej na dronie, która jest jego charakterystycznym elementem.

Jednym z głównych elementów sieci jest blok wspólnych warstw splotowych. Efekt jego działania stanowi wejście do sześciu sekwencji warstw. Trzy z nich dedykowane są kątom obrotu – odpowiednio w osiach X , Y i Z . Dwa bloki służą do wspomaganie predykcji kątów, a ostatni służy do ekstrakcji cech odpowiedzialnych za przewidywanie położenia kamery.

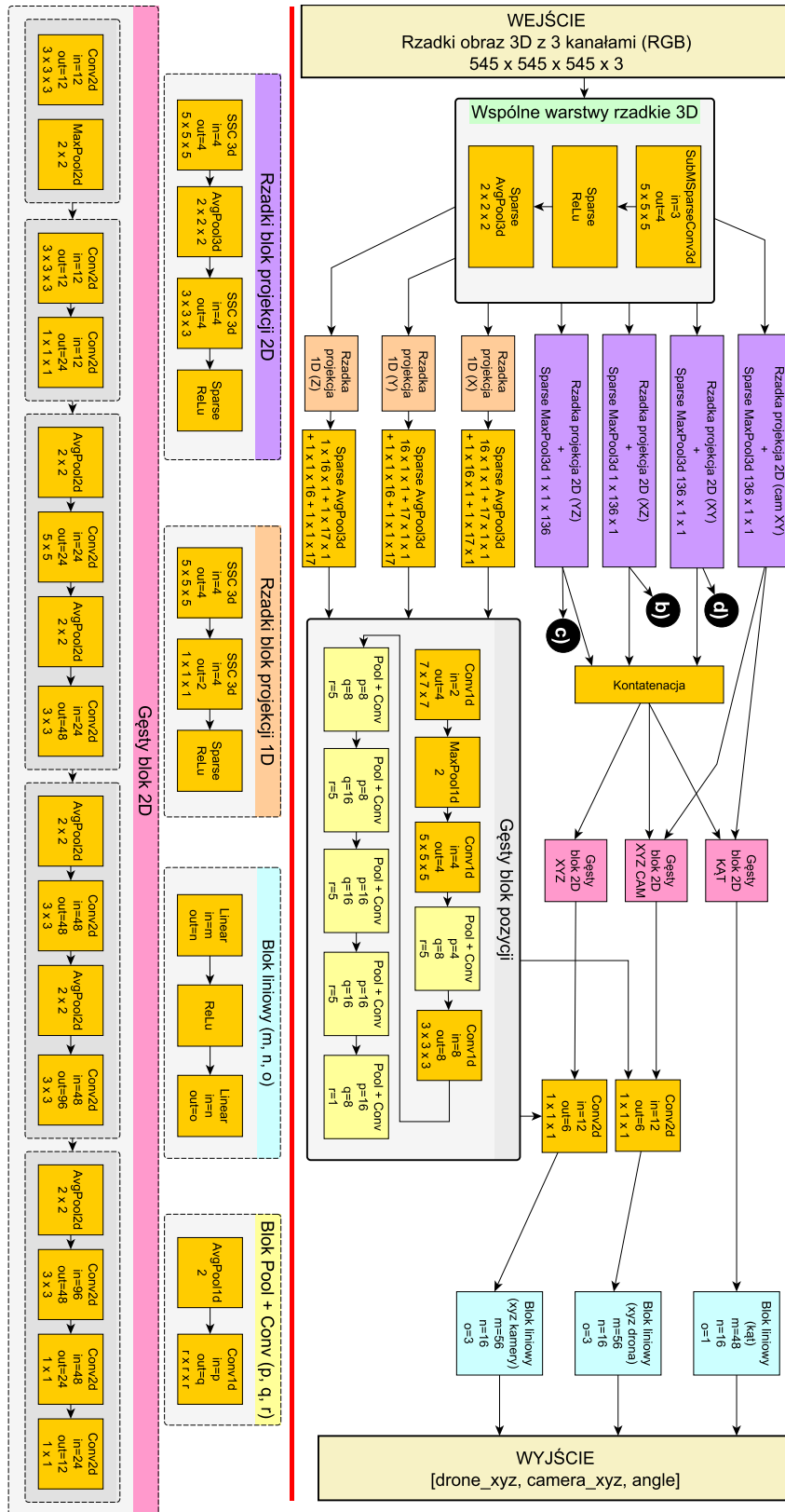
Efektem działania warstw splotowych jest kompaktowy opis sceny, stanowiący wejście do gęstych warstw liniowych. Do predykcji kątów roll i pitch używanych jest 48 cech, do yaw 128 cech, a do pozycji kamery 64 cechy.

Na rysunku 3.6 przedstawiono a) przykładowe wejście do sieci. Przedstawiono również wyjścia z wybranych warstw splotowych. Na części b) rysunku (wyjście z 1. bloku warstw splotowych) widać, że jeden z kanałów wyjściowych zawiera wyłącznie kamerę drona, podczas gdy kanał inny zawiera drona pozbawionego kamery. Na rysunkach c) i d) przedstawiono wyjścia z warstw wspierających predykcję kątów. Dodatkowo na schemacie sieci (rysunek 3.5) oznaczono miejsca, w których pobrano dane do rysunków.

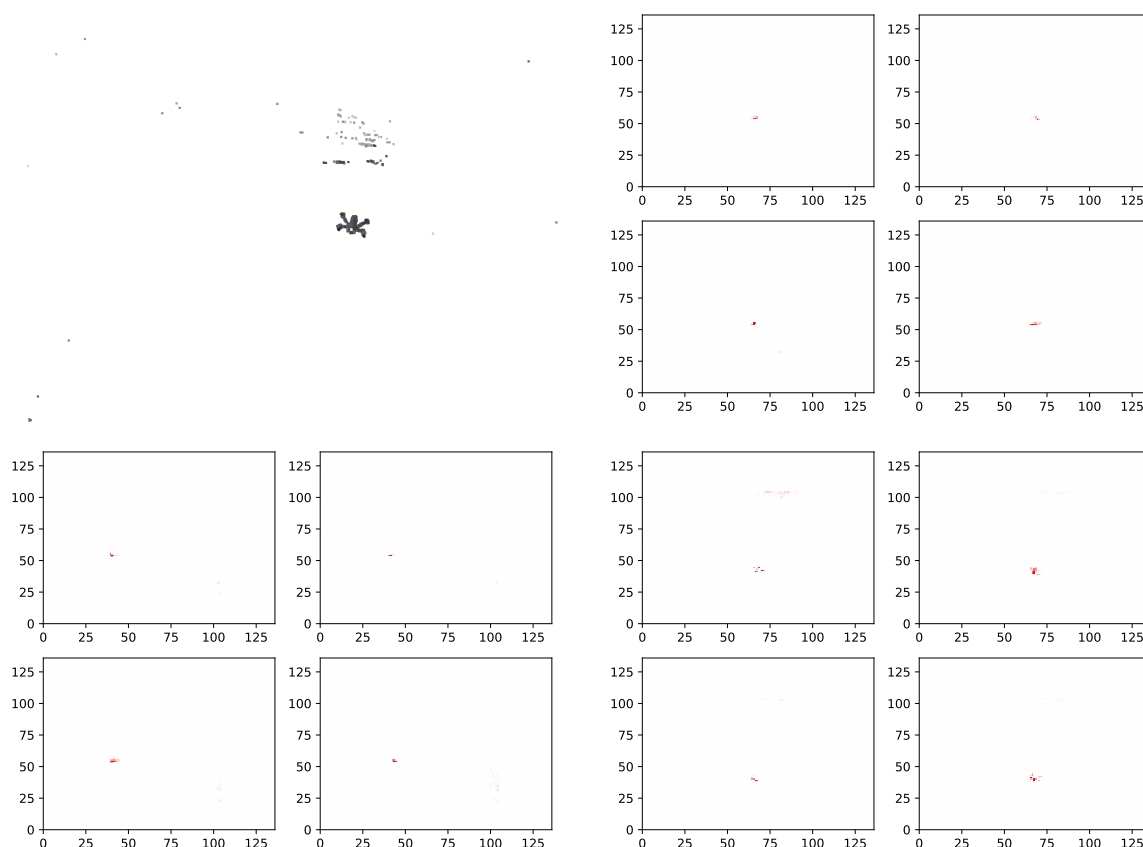
3.5 ProjectionCNN3D

Na rysunku 3.7 przedstawiono architekturę sieci ProjectionCNN3D. Sieć tą zastosowano w zadaniu predykcji lokalizacji i orientacji drona na podstawie rzadkiego obrazu trójwymiarowego o rozdzielczości $545 \times 545 \times 545$. W tym przypadku dane gęste miałyby rozmiar rzędu 10^8 . Sieć uczona była z użyciem danych rzadkich pozyskanych z kamery głębiu ZED 2i. Dodatkowo sieć umożliwia przewidzenie pozycji kamery drona, aczkolwiek opcja ta dodana została wyłącznie na potrzeby treningu modelu – dzięki temu uzyskano lepszą dokładność przewidywania kąta obrotu drona.

Sieć ta posiada blok preprocessingu 3D, który składa się z rzadkich warstw splotowych SSC, jednostek odcinających ReLu oraz pooling uśredniającego. Kluczową i unikalną



Rysunek 3.7: Architektura sieci ProjectionCNN3D. (Źródło: opracowanie własne)



Rysunek 3.8: Sieć ProjectionCNN3D: a) przykładowe wejście; projekcje na płaszczyzny b) XZ , c) YZ , d) XY , po 4 kanały na każdy z rzutów. (Źródło: opracowanie własne)

cechą tej sieci są rzadkie projekcje przetworzonego obrazu trójwymiarowego do podprzestrzeni dwuwymiarowych i jednowymiarowych (rzuty na płaszczyzny XY , XZ i YZ oraz osie X , Y , Z). Operacja projekcji powinna sprzyjać zadaniu lokalizacji drona w osiach X , Y i Z . Zastosowanie tego rodzaju projekcji motywowane jest również tym, że powinny być one pomocne w ustalaniu kąta obrotu drona w osi Z – ponieważ kąt ten jest dobrze widoczny w rzucie na płaszczyźnie XY . Na rysunku 3.8 przedstawiono przykładowe wejście do sieci oraz rezultat projekcji na trzy płaszczyzny: XZ , YZ oraz XY . Ponieważ dron wychyla się w nieznacznym stopniu w osiach X i Y , projekcja na płaszczyznę XY wydaje się szczególnie przydatna dla predykcji kąta obrotu w osi Z .

Po wykonaniu projekcji dane są przetwarzane w sposób gęsty – w oparciu o gęste bloki 2D (po jednym dla kąta, pozycji i pozycji kamery) oraz blok pomocniczy, operujący na gęstych danych jednowymiarowych. Za predykcję odpowiedzialne są bloki liniowe, przyjmujące na wejście 48 cech (kąt) oraz 56 cech (pozycja drona i pozycja kamery).

4. Wyniki badań eksperymentalnych

4.1 Środowisko badawcze

Badania prowadzono w Laboratorium Automatyki Mobilnej Wydziału Informatyki ZUT (rysunek 4.1). W wyposażeniu laboratorium znajdują się m. in.:

- kamery RGB,
- dwie kamery głębi Intel RealSense D435,
- dwie kamery głębi ZED 2i,
- kamera zdarzeń SilkyEvCam (VGA),
- cztery roboty mobilne iRobot 650,
- ramię robota uArm Swift Pro,
- dwa drony Parrot ANAFI,
- dron Parrot Bebop 2.

Dodatkowo laboratorium wyposażone jest w dwa komputery o następujących parametrach:

- 8-rdzeniowy procesor Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz,
- 128 GB pamięci RAM DDR 4,
- karta graficzna NVIDIA GeForce RTX 3090 (24 GB),
- dysk NVMe PCIe 4.0 x4,
- dysk HDD,
- karta sieciowa i karta Wi-Fi.

Oprócz tych jednostek dostępny do badań był także serwer obliczeniowy Wydziału Informatyki, współdzielony pomiędzy wszystkich zainteresowanych pracowników. Serwer wyposażony jest w:

- 2 procesory AMD EPYC 7H12 (każdy z nich posiada 64 rdzenie),
- 8 kart graficznych NVIDIA A100 40 GB,
- 4 TB RAM,
- 20 TB przestrzeni dyskowej.

Serwer umożliwia tworzenie maszyn wirtualnych i konfigurowanie ich z dosyć dużą swobodą, z wyjątkiem kart graficznych – do jednej maszyny może być przypisana maksymalnie jedna karta (ewentualnie jej część, np. 1/7).

Centralnym punktem laboratorium jest metalowa klatka o wymiarach $5.4 \times 4.1 \times 3$ m (rysunek 4.1), która stanowi obszar operacyjny dla badań związanych z dronami i robotami mobilnymi. Klatka pokryta jest siatką ochronną, która zabezpiecza przed wylotem drona poza jej obszar.

W badaniach wykorzystywano środowiska języka Python jako główne środowisko obliczeniowe – wraz z dostępnymi bibliotekami, w tym głównie: Pytorch, spconv, opencv,



Rysunek 4.1: Wyposażenie Laboratorium Automatyki Mobilnej. Po lewej – u góry kamera zdarzeń, na dole drony oraz ramię robota. Po prawej: widok na klatkę oraz roboty mobilne. (Źródło: opracowanie własne)

numpy, scipy, matplotlib, pandas.

4.2 Hipotezy badawcze

Główna teza badawcza brzmi: „Uwzględnienie rzadkości sygnałów wejściowych umożliwia opracowanie wydajniejszych (tj. szybszych z punktu widzenia predykcji, przy utrzymaniu jakości predykcji) struktur sieci neuronowych w stosunku do klasycznych rozwiązań referencyjnych, w tym pracujących na danych gęstych.”. Teza ta była weryfikowana w zadaniach związanych ze sterowaniem robotów mobilnych z wykorzystaniem wizyjnych torów pomiarowych, głównie do pomiarów lokalizacji i orientacji sterowanych obiektów. Z tezą badawczą związane są zagadnienia pomocnicze oraz dodatkowe pytania badawcze:

1. Czy z wykorzystaniem sieci splotowych można rozwiązać zadania lokalizacji i orientacji? Jakie cechy i elementy struktur sieci są korzystne z punktu widzenia zadań wyznaczania lokalizacji i orientacji?
2. Czy sieci splotowe dostosowane do pracy z danymi rzadkimi pozwalają na osiągnięcie wysokiej jakości predykcji?
3. Czy sieci splotowe dostosowane do pracy z danymi rzadkimi pozwalają na szybsze i oszczędniejsze pamięciowo uczenie niż sieci gęste?
4. Czy sieci splotowe dostosowane do pracy z danymi rzadkimi pozwalają na szybszą i oszczędniejszą pamięciowo predykcję niż sieci gęste?

Jakość predykcji należy traktować w odniesieniu do dokładności sensora pomiarowego. Przez wysoką jakość rozumie się tu błędy na poziomie porównywalnym lub niższym od błędów wprowadzanych przez system pomiarowy. Dwa ostatnie pytania dotyczące przetwarzania danych rzadkich wymagają również pewnego komentarza. Odpowiedzi na te pytania wydają się oczywiste w środowisku jednowątkowym, jednak jak pokazują eksperymenty oraz analiza sposobu wykonywania operacji nie są one oczywiste, gdy obliczenia są wykonywane na kartach obliczeniowych w środowisku wielowątkowym.

Środowisko kart obliczeniowych jest dobrze dostosowane do obliczeń na macierzach gęstych i wiele algorytmów numerycznej algebry liniowej ma swoje wersje dla środowisk pracujących równolegle. Z drugiej strony wiele operacji na macierzach rzadkich nie zachowuje stopnia rzadkości. Przykładowo wynik mnożenia macierzy rzadkich może być gęsty i w ogólności trudno wypowiedzieć się na temat wypełnienia wyniku iloczynu dwóch macierzy rzadkich, natomiast analiza złożoności obliczeniowej jest zagadnieniem niełatwym nawet w przypadku jednowątkowym (por. [3]). Podobnie jest w przypadku operacji splotu, która zwykle zwiększa stopień wypełnienia macierzy wynikowej, tak jak i operacja poolingu. Z tych powodów w rzadkich sieciach splotowych wykorzystuje się operację Submanifold Sparse Convolution zamiast zwykłego splotu, która – kosztem utraty niektórych własności splotu – zachowuje stopień wypełnienia. Ponadto jest dość istotna różnica pomiędzy zagadnieniem uczenia oraz predykcji. W przypadku uczenia na danych rzadkich niewątpliwa przewaga polega na możliwości przesłania większej liczby próbek w pojedynczym wsadzie, jednak w zagadnieniu predykcji w trybie on-line przesyłamy na kartę pojedynczą próbkę w częstości zdeterminowanej przez częstotliwość urządzenia pomiarowego. Na wszystkie wyżej wymienione aspekty kładziony był nacisk w trakcie eksperymentów.

4.3 Przygotowanie eksperymentów

W ramach empirycznego uzasadniania tez badawczych przeprowadzono kilka eksperymentów o różnym stopniu złożoności, w trakcie których powstały autorskie zbiory danych oraz autorskie architektury splotowych sieci neuronowych. Wytrenowane modele wykorzystane zostały w zadaniach śledzenia i sterowania robotami z wykorzystaniem sensorów wizyjnych.

Rozpoczęto od dwóch badań wstępnych wykonanych na danych syntetycznych. Badania te prowadzono na wygenerowanych danych dwu- i trójwymiarowych (Line2D oraz Plane3D). W eksperymentach tych nie wykorzystywano sensorów wizyjnych bezpośrednio – użyto tylko symulowanych danych, zbliżonych do uzyskiwanych z sensorów. Celem była weryfikacja podstawowych założeń dotyczących sieci splotowych wykorzystujących warstwy rzadkie.

Kolejne trzy eksperymenty wykonano na rzeczywistych stanowiskach badawczych z wykorzystaniem kamer RGB oraz kamer głębi, były to kolejno:

1. Zadanie sterowania robotami mobilnymi z wykorzystaniem pomiarów wizyjnych (zbiór iRobot4).
2. Zadanie obserwacji stanu ramienia robota za pomocą kamer głębi Intel RealSense D435 (zbiór (uArm)).

Tabela 4.1: Zadania i zbiory danych wykorzystywane w trakcie eksperymentów.

Dane	Opis	Rozmiar próby (l. próbek)	Rozmiar pojedynczej próbki/ obrazu
Line2d	Dane syntetyczne – odcinek, predykcja pozycji, kąta obrotu oraz grubości (x, y, α, w) .	∞	3840×2180
Plane3d	Dane syntetyczne – samolot w 3D, predykcja pozycji (x, y, z) , kątów obrotu $(\alpha_x, \alpha_y, \alpha_z)$.	∞	$300 \times 300 \times 300$
iRobot4	Sterowanie pozycją czterech robotów mobilnych iRobot 650, predykcja pozycji i kąta obrotu na podstawie danych z kamery RGB (x, y, α) .	34.1 GB (101 000)	1280×720
uArm	Predykcja stanu robota uArm Swift Pro – kątów skręcenia 3 członów – na podstawie chmur punktów z dwóch kamer głębi Intel Realsense $(\alpha_0, \alpha_1, \alpha_2)$.	5.4 GB (10 440)	$750 \times 750 \times 1375$
Drone	Dane dotyczące stanu (pozycji i kąta) drona Parrot ANAFI – obserwacja na podstawie chmur punktów z kamery ZED 2i (x, y, z, α) .	136 GB (803 740)	$545 \times 545 \times 545$
DroneCrop	Dane dotyczące stanu (kąta) drona Parrot ANAFI – obserwacja na podstawie fragmentów chmur punktów z kamery ZED 2i, które przedstawiają wyciętego drona (α) .	15.7 GB (803 740)	$38 \times 38 \times 40$
DroneSim	Dane dotyczące stanu (pozycji i kątów) drona Parrot ANAFI – obserwacja na podstawie chmur punktów z kamery ZED 2i $(x, y, z, \alpha_x, \alpha_y, \alpha_z)$.	64.7 GB (1 700 000)	$545 \times 545 \times 545$

3. Zadanie sterowania dronem z wykorzystaniem w torze sprzężenia zwrotnego kamery głębi ZED 2i (zbiory Drone oraz DroneCrop).

Ostatni zrealizowany eksperyment stanowi rozszerzenie badań dotyczących drona i otwiera nowy kierunek badań, związany z analizą jakości uczenia w środowisku symulowanym pod kątem możliwości wykorzystania w środowisku rzeczywistym.

Charakterystykę problemów i dane wykorzystywane w części eksperymentalnej przedstawiono w tabeli 4.1.

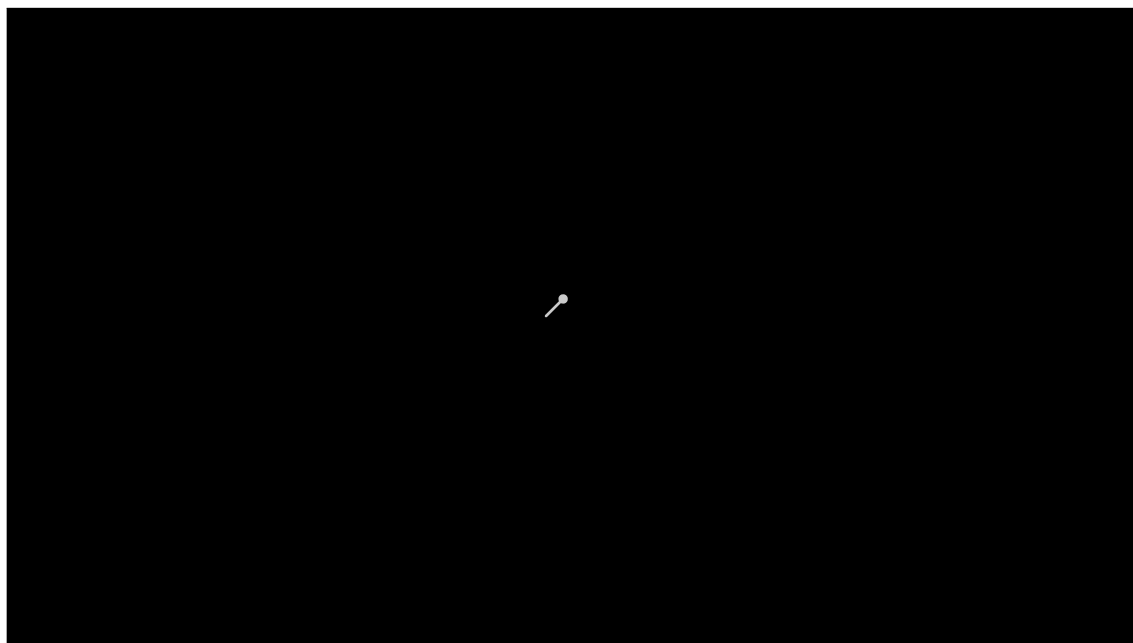
4.4 Badania wstępne – dane dwuwymiarowe Line2D

4.4.1 Sformułowanie problemu i zbiór danych

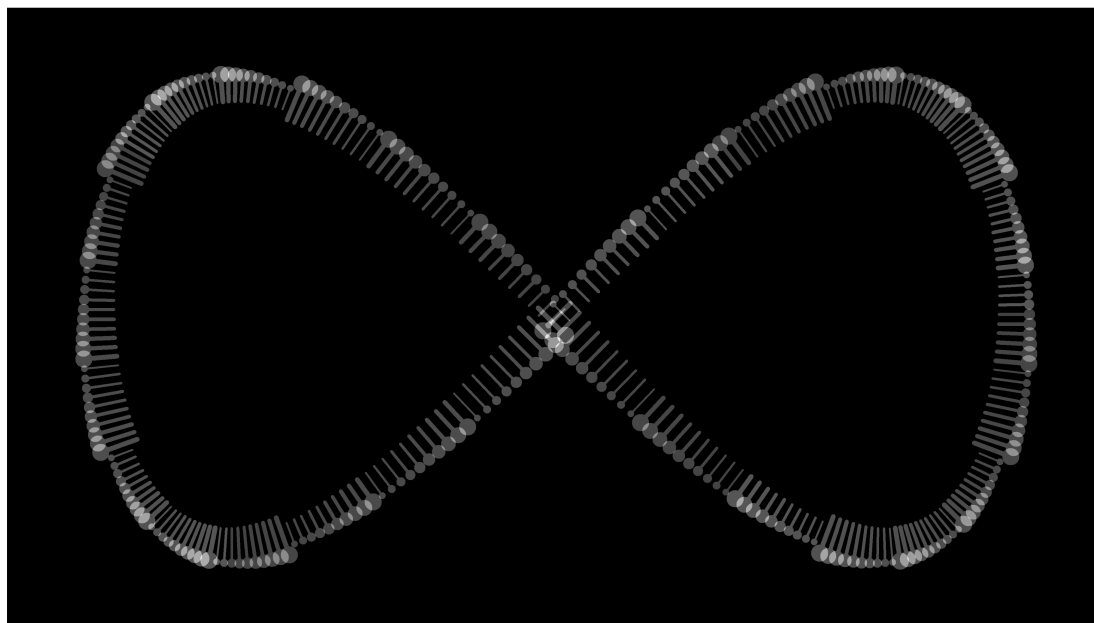
W pierwszej części badań wstępnych skupiono się na syntetycznych danych dwuwymiarowych. Zadanie dotyczyło analizy możliwości wyznaczania pozycji oraz orientacji prostego obiektu na obrazie 2D przy wykorzystaniu sieci splotowej. Przygotowano zbiór danych Line2D składający się z obrazów przedstawiających odcinek zakończony z jednej strony wypełnionym okręgiem, a także z pewną zawartością szumu (rysunek 4.2). Ponieważ dane generowane są syntetycznie, w eksperymencie tym dysponujemy dokładną wartością etykiet tj. pozycji, kąta obrotu oraz grubości. Kropka na końcu odcinka powoduje, że predykcja kąta jest jednoznaczna – w przeciwnym wypadku predykcja byłaby dana jedynie z dokładnością do obrotu o kąt 180° .

Zbiór złożony był z monochromatycznych (jednokanałowych) obrazów w rozdzielczości 4K (3840×2160), które generowano w sposób losowy i nieznacznie zaszumiano tło. Dane były generowane w dwóch równoważnych formach: rzadkiej i gęstej. Zadaniem sieci neuronowej było ustalenie położenia (x, y) obiektu w układzie współrzędnych obrazu, a także ustalenie kąta obrotu ($\alpha \in [-\pi, \pi]$) oraz grubości linii (6 – 15 pikseli). Kolor obiektu zmieniał się w zakresie 200 – 255, a długość w zakresie 80 – 140 pikseli. Promień okręgu wynosił dwukrotność grubości linii.

W trakcie uczenia dane generowano losowo w sposób ciągły, natomiast wcześniej przygotowano również oddzielny, niezmienny zbiór testowy, który składał się z 300 próbek, w których obiekt przemieszcza się po kształcie ósemki, jednocześnie zmieniając kolor i grubość (rysunek 4.3). Z danych uczących wydzielono próbę walidującą (około 10%), którą wykorzystano do właściwego wyznaczenia momentu zakończenia procesu uczenia.



Rysunek 4.2: Przykładowy obraz przedstawiający lokalizowany obiekt.
(Źródło: opracowanie własne)



Rysunek 4.3: Nałożenie wszystkich obrazów stanowiących zbiór testowy dla zbioru przedstawiającego linię. (Źródło: opracowanie własne)

4.4.2 Wykorzystany model

Do rozwiązania zadania dla danych rzadkich posłużono się modelem LineCNN2D oraz analogicznym modelem referencyjnym LineCNN2D_Dense.

W trakcie treningu dane nieznacznie zaburzano, dodając dwa rodzaje szumu:

- dodanie pewnej losowej liczby punktów tła,
- dodanie lub zmiana losowej liczby punktów wokół obiektu.

Trening obu modeli: rzadkiego i gęstego prowadzono na wydziałowym serwerze, na maszynie wirtualnej z jedną kartą graficzną A100 i 16 procesorami. Predykcję prowadzono na komputerze w laboratorium z kartą graficzną RTX 3090. W obu przypadkach badania prowadzono posługując się system operacyjnym Ubuntu 22, środowiskiem Python 3.10, z bibliotekami: SpConv 2.3.8 oraz PyTorch 2.5.1. z wersją CUDA 12.1 na serwerze, oraz CUDA 12.7 na komputerze stacjonarnym w laboratorium.

4.4.3 Wyniki

W tabeli 4.2 oraz na rysunku 4.4 przedstawiono wyniki dotyczące jakości predykcji modeli na zbiorze testowym. Czas uczenia obu modeli ustawiono na odpowiednio długi – tak aby uzyskać gwarancję zbieżności, czyli liczby iteracji, po zwiększeniu której nie następowało już zmniejszenie błędu na danych walidujących. Wyraźnie widoczna jest różnica w dokładności na korzyść modelu LineCNN2D. Interpretacja tego wyniku nie jest jednoznaczna – jedyna rozbieżność w strukturze modeli to sposób realizacji operacji splotu, tj. początkowe warstwy rzadkie tej sieci realizują operację Sparse Submanifold Convolution, która nie jest równoważna operacji splotu.

Sieć rzadka projektowana była w oparciu o operację Sparse Submanifold Convolution,

Tabela 4.2: Porównanie jakości predykcji na zbiorze testowym czterech obserwowanych zmiennych: kąt obrotu α , pozycja (x, y) oraz grubość, dane Line2D.

	α [rad]	x [px]	y [px]	grubość [px]
LineCNN2D	0.037	7.471	6.966	0.245
LineCNN2D_Dense	0.059	15.686	11.259	0.359
zakres wartości	$[-\pi; \pi)$	[0; 3840]	[0; 2160]	[6; 15]

Tabela 4.3: Porównanie parametrów uczenia na karcie graficznej NVIDIA A100 dla zbioru liczącego 5000 próbek, dane Line2D.

Model	Czas treningu pojedynczej epoki [s]	Maks. rozmiar batcha [liczba próbek]
LineCNN2D	28.04	570
LineCNN2D_Dense	422.71	13

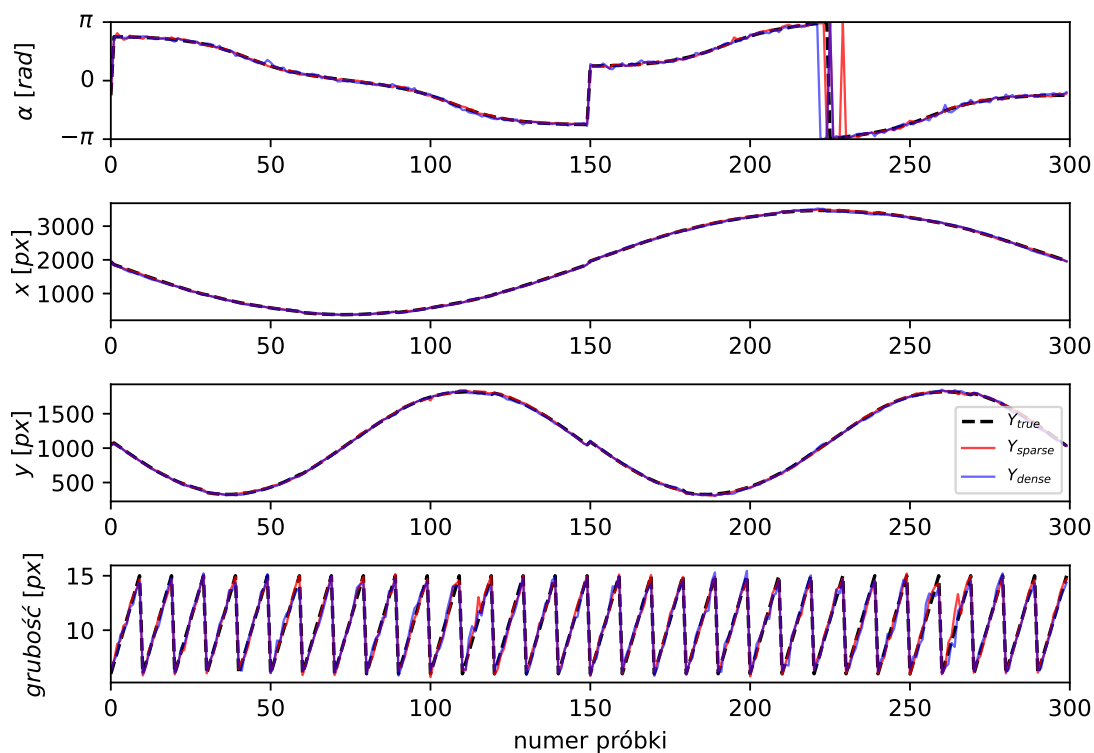
Tabela 4.4: Porównanie parametrów predykcji na karcie graficznej NVIDIA RTX 3090, dla zbioru danych Line2D.

Model	Czas predykcji dla 1 próbki [s]	Maks. rozmiar batcha [liczba próbek]
LineCNN2D	$9.61 \cdot 10^{-3}$	710
LineCNN2D_Dense	$2.20 \cdot 10^{-2}$	20

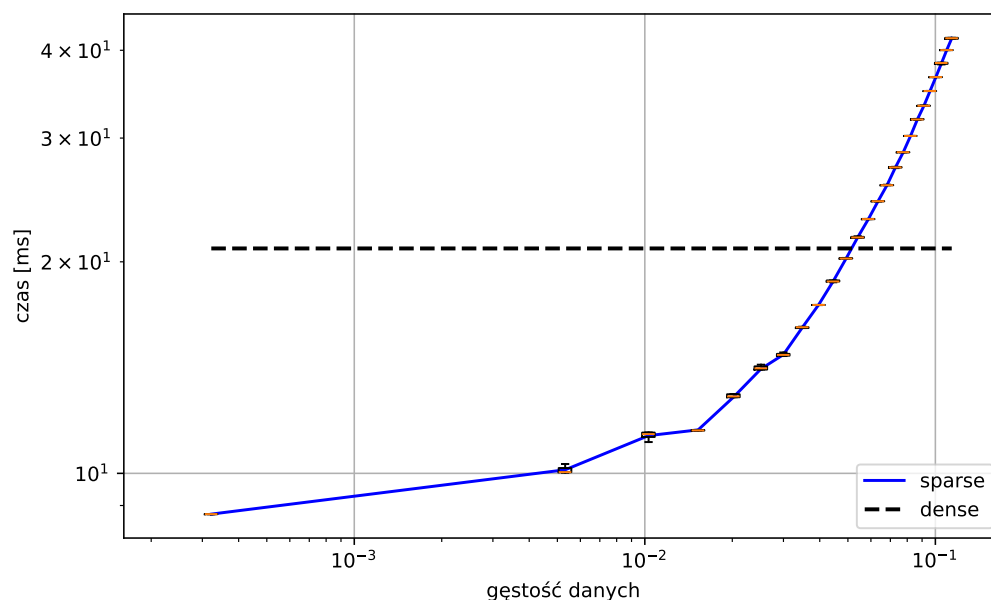
której efekt działania jest inny niż typowego splotu, stąd prosty mechanizm transfer learning do modelu gęstego nie dawał pożądanych rezultatów i konieczne było douczenie modelu. Aby uzyskać lepszy wynik dla sieci gęstej konieczne byłyby dalsze eksperymenty m.in. z modyfikacją architektury.

W tym eksperymencie kluczowe jednak było obiektywne porównanie czasu przetwarzania dla obu architektur. W tabelach 4.3 oraz 4.4 przedstawiono porównanie czasów przetwarzania dla modelu gęstego i rzadkiego oraz dla etapów uczenia oraz predykcji. W tabelach zamieszczono również informacje dotyczące wymagań pamięciowych. Wyraźnie widać, że uwzględnienie rzadkości danych pozwalana na znacznie sprawniejsze przeprowadzenie treningu modelu. Ponadto dla pewnych rzadkości znacząco przyspiesza predykcję, a jednocześnie prowadzi do znacznej oszczędności pamięci.

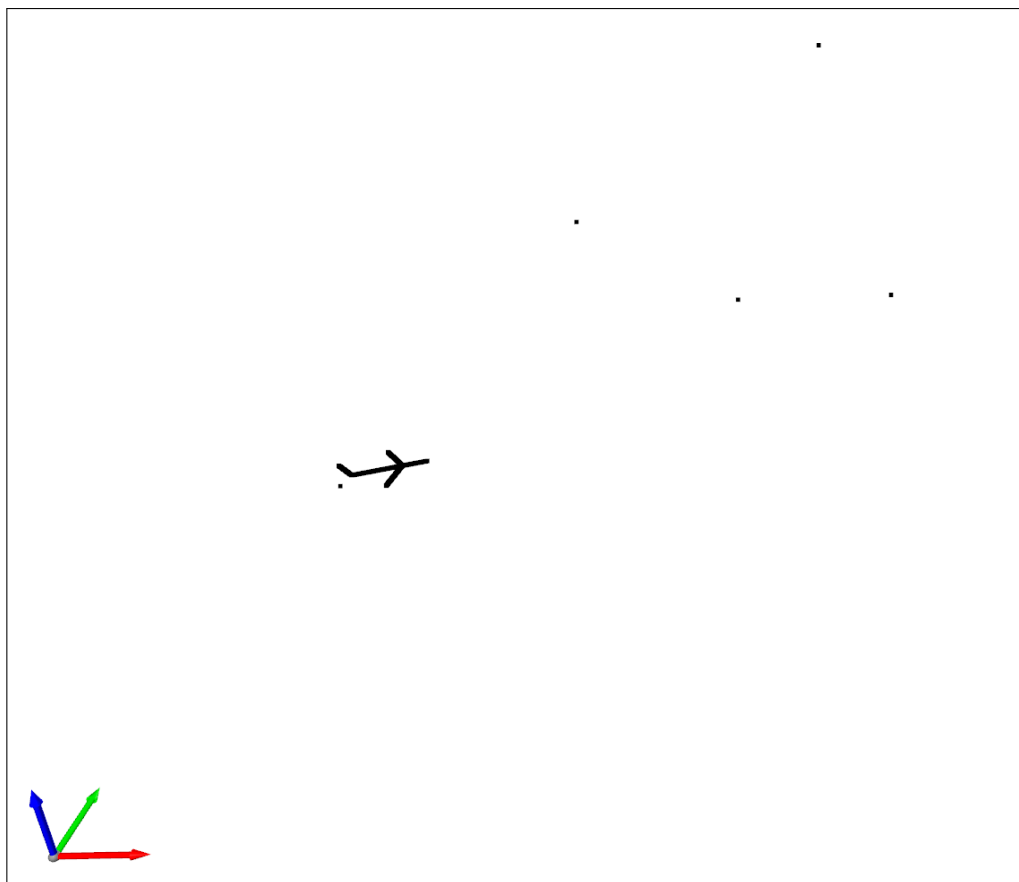
Dodatkowo zbadano wpływ stopnia rzadkości danych na czas predykcji modelu rzadkiego – w odniesieniu do modelu gęstego. Stopniem gęstości danych sterowano zasumiając tło lokalizowanego obiektu. Wyniki przedstawiono na rysunku 4.5. Uzyskane wyniki pokazują ponad dziesięciokrotną przewagę czasu uczenia na korzyść modelu rzadkiego przy wypełnieniu macierzy danych na poziomie 10^{-2} i niższych. W przypadku predykcji przewaga nie jest tak wyraźna – około dwukrotna. Spowodowane jest to głównie tym, że w trakcie uczenia możemy przesłać większą porcję danych rzadkich, natomiast w przypadku predykcji w trybie online liczony jest czas przetwarzanie pojedynczej próbki co jest zgodne ze sposobem pobierania danych z sensora wizyjnego. Wyniki te pokazują również,



Rysunek 4.4: Porównanie jakości predykcji na zbiorze testowym modeli LineCNN2D i LineCNN2D_Dense dla danych Line2D. (źródło: opracowanie własne)



Rysunek 4.5: Porównanie czasu predykcji dla pojedynczej próbki w przypadku modelu rzadkiego i gęstego dla danych Line2D. Na osi odciętych przedstawiono rzadkość, czyli stopień wypełnienia macierzy danych wejściowych. (źródło: opracowanie własne)



Rysunek 4.6: Przykładowy widok sceny zawierającej prosty model samolotu dane Plane3d. W lewym dolnym rogu zaznaczono układ współrzędnych. (Źródło: opracowanie własne)

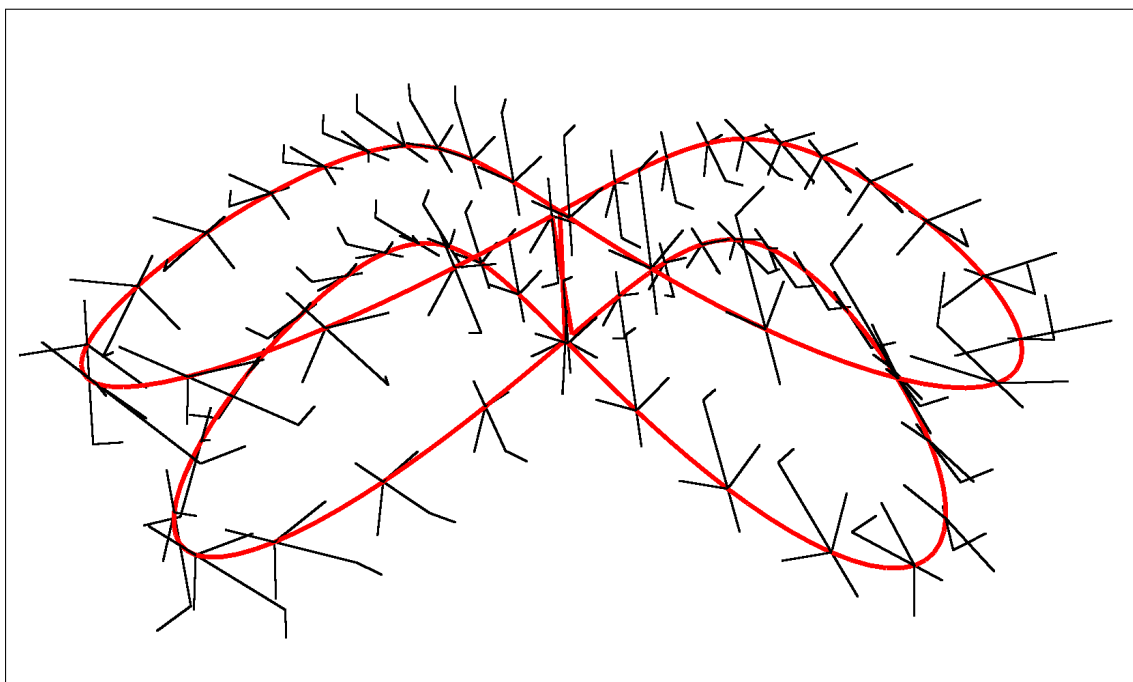
że uproszczona operacja splotu stosowana w macierzach rzadkich może być skutecznie wykorzystywana w zadaniach predykcji położenia i obrotu obiektów.

4.5 Badania wstępne – dane trójwymiarowe Plane3D

4.5.1 Sformułowanie problemu i generowanie danych

W drugim etapie badań wstępnych przetestowano działanie rzadkich sieci splotowych dla danych trójwymiarowych. W tym celu stworzono syntetyczny zbiór przedstawiający prosty model samolotu (rysunek 4.6). Uzyskane trójwymiarowe obrazy były nieznacznie zaszumiane, a następnie poddawane wokselizacji – rozmiar woksela ustalono na 0.01 jednostki, a siatka wokseli miała rozmiar $300 \times 300 \times 300$. Zadanie predykcji polegało na ustaleniu położenia obiektu (x, y, z) i kąta obrotu w trzech osiach: roll, pitch, yaw, oznaczonych $(\alpha_x, \alpha_y, \alpha_z)$. Położenie obiektu zmieniało się w zakresie $[0, 3]$ jednostki, a kąty obrotu roll i pitch zmieniały się w zakresie $[-\frac{\pi}{2}, \frac{\pi}{2}]$ radiana. Kąt yaw zmieniał się natomiast w zakresie $[-\pi, \pi]$ radiana.

Podobnie jak w poprzednim eksperymencie, dane generowano losowo w trakcie uczenia. Przygotowano również oddzielny, niezmienny zbiór testowy – składał się on z 300



Rysunek 4.7: Nałożenie obrazów stanowiących zbiór testowy dla zbioru danych Plane3D. (Źródło: opracowanie własne)

próbek, w których obiekt przemieszcza się po kształcie ósemki, zmieniając wszystkie trzy kąty obrotu (rysunek 4.7).

4.5.2 Wykorzystany model

Do rozwiązania zadania wykorzystano model SlicingCNN3D składający się z warstw rzadkich oraz gęstych. Model ten został opisany szczegółowo w rozdziale 3. W trakcie treningu dane w formie wokseli zaburzano dodając dwa rodzaje szumu:

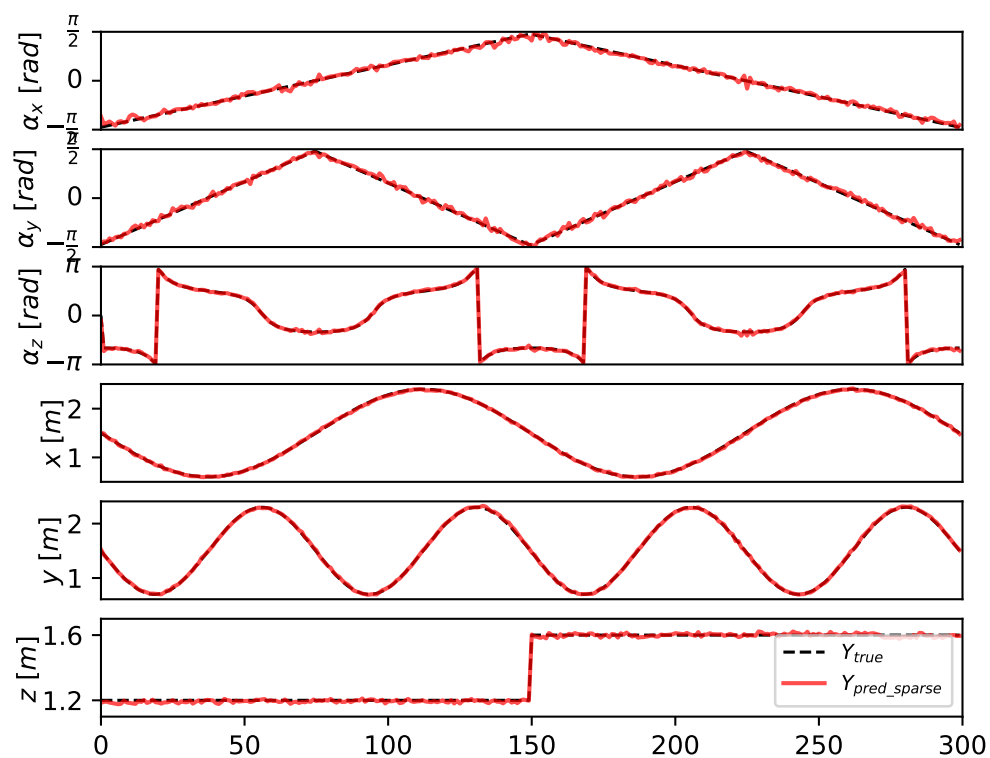
- dodanie pewnej losowej liczby wokseli tła,
- dodanie lub zmiana losowej liczby wokseli wokół obiektu.

Trening obu modeli: rzadkiego i gęstego prowadzono na wydziałowym serwerze, na maszynie wirtualnej z jedną kartą graficzną A100 i 16 procesorami. Predykcję prowadzono na komputerze w laboratorium z kartą graficzną RTX 3090. W obu przypadkach badania prowadzono posługując się system operacyjnym Ubuntu 22, środowiskiem analogicznym jak w poprzednim eksperymencie.

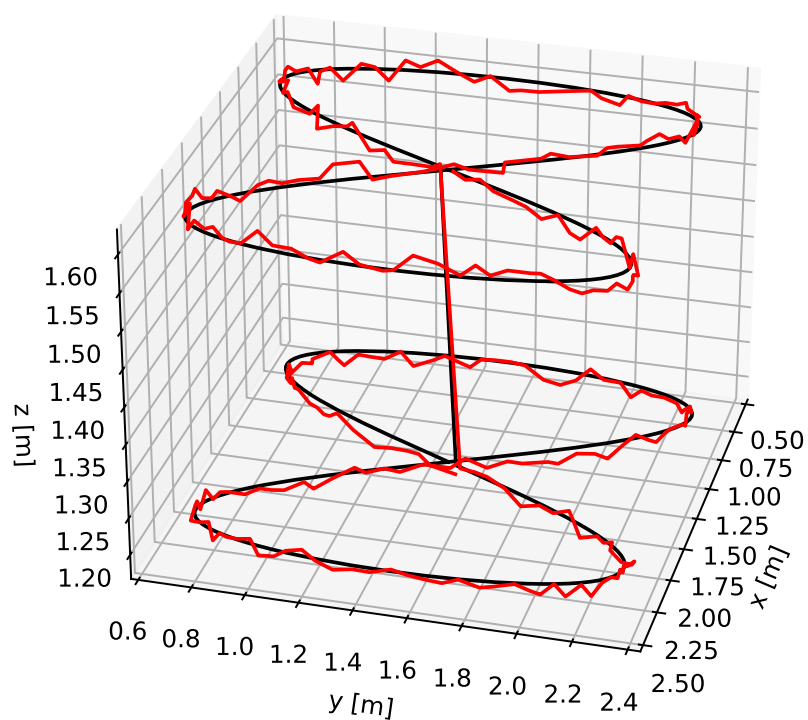
4.5.3 Wyniki

Wyniki ilościowe przedstawiono w tabeli 4.5 dla modelu rzadkiego. Modelu gęstego nie udało się nastroić w rozsądnym czasie. Na rysunku 4.8 przedstawiono przebieg trajektorii zadanej i obserwowanej w rozbiciu na poszczególne współrzędne, natomiast na rysunku 4.9 przedstawiono trajektorię w układzie XYZ.

W tabelach 4.6 i 4.7 przedstawiono informacje dotyczące czasów przetwarzania dla modeli rzadkiego i gęstego w zadaniach uczenia i predykcji. W obu przypadkach liczony



Rysunek 4.8: Porównanie rozpoznawanych zmiennych (pozycja i kąt obrotu) dla danych Plane3D. (źródło: opracowanie własne)



Rysunek 4.9: Wizualizacja trajektorii zadanej i obserwowanej dla danych Plane3D. (źródło: opracowanie własne)

Tabela 4.5: Jakość predykcji na zbiorze testowym dla danych Plane3D dla modelu rzadkiego SlicingCNN3D.

Model	α_X [rad]	α_Y [rad]	α_Z [rad]	x [m]	y [m]	z [m]
SlicingCNN3D	0.047	0.048	0.039	0.010	0.010	0.008
Zakres wartości	$[-\frac{\pi}{2}, \frac{\pi}{2}]$	$[-\frac{\pi}{2}, \frac{\pi}{2}]$	$[-\pi, \pi]$	[0, 3]	[0, 3]	[0, 3]

Tabela 4.6: Porównanie parametrów uczenia na karcie graficznej NVIDIA A100 dla zbioru liczącego 5000 próbek (zbiór Plane3D).

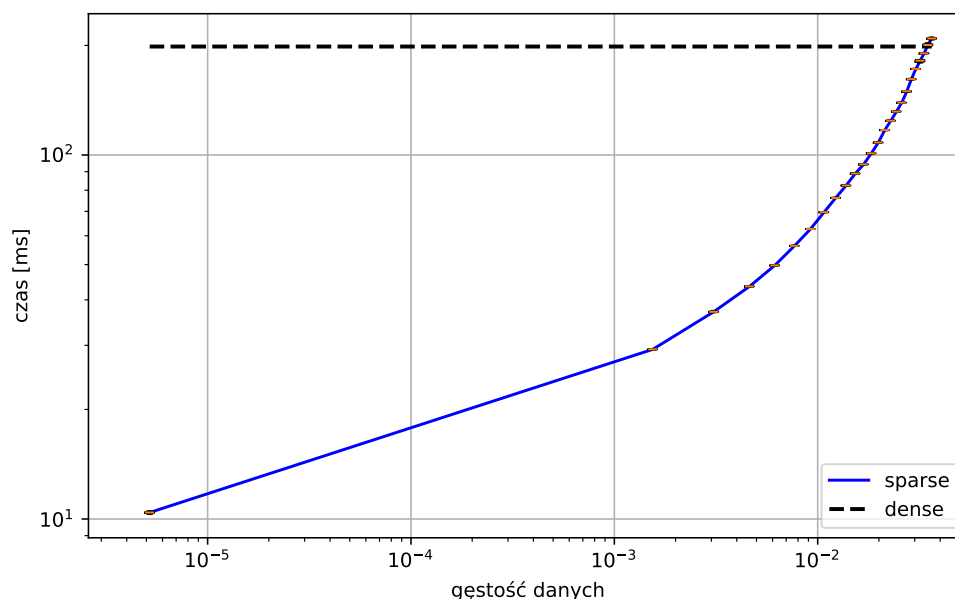
Model	Czas treningu – 1 epoka [s]	Maks. rozmiar batcha [liczba próbek]
SlicingCNN3D	8.12	3240
SlicingCNN3D_Dense	10633.34	4

Tabela 4.7: Porównanie parametrów predykcji na karcie graficznej NVIDIA RTX 3090 dla zbioru Plane3D).

Model	Czas predykcji – pojedyncza próbka [ms]	Maks. rozmiar batcha [liczba próbek]
SlicingCNN3D	10.42	12250
SlicingCNN3D_Dense	198.60	6

był czas przetwarzania pojedynczej epoki zawierającej 5 000 próbek. W tabeli 4.6 widać bardzo wyraźną różnicę w czasie uczenia – o czynnik 10^4 . Ponieważ dane rzadkie zajmują mniej miejsca, stąd rozmiar pojedynczego wsadu może być większy. Przy uczeniu modelu obsługującego dane rzadkie mamy zysk zarówno w sposobie realizacji operacji na danych rzadkich o małym stopniu wypełnienia, jak i na przesyłaniu danych do karty. W zadaniu predykcji różnica jest mniejsza, ponieważ dotyczy wyznaczenia parametrów pojedynczej próbki/zdjęcia i w tym przypadku zysk jest jedynie w sposobie realizacji operacji na danych rzadkich.

rysunek 4.10 przedstawia wpływ stopnia wypełnienia macierzy danych na czas predykcji. Dane trójwymiarowe w ogólności charakteryzują się większą rzadkością niż dane dwuwymiarowe. Jak można zaobserwować dla danych bardzo rzadkich, występuje około dwudziestokrotne przyspieszenie czasu predykcji modelu, natomiast zysk ten utrzymuje się do wypełnienia rzędu 10^{-2} , gdzie czas predykcji jest ponad dwukrotnie szybszy. Czasy obliczeń zrównują się przy wypełnieniu około 0.03.



Rysunek 4.10: Porównanie czasu predykcji dla pojedynczej próbki w przypadku modelu rzadkiego i gęstego dla danych (Plane3D) (Źródło: opracowanie własne).

4.6 Sterowanie robotami mobilnymi iRobot 650

4.6.1 Stanowisko badawcze

Kolejny eksperyment został przeprowadzony z wykorzystaniem wyposażenia laboratorium. Jego przedmiotem było sterowanie robotami mobilnymi iRobot z wykorzystaniem sprzężenia wizualnego z kamery RGB. Na stanowisko badawcze składały się cztery roboty iRobot 650, obserwowane przez kamerę szerokokątną. rysunek 4.11 przedstawia widok obszaru roboczego z góry. Powierzchnia, po której mogły się przemieszczać roboty, ograniczona była przez brzegi klatki.

Głównym celem eksperymentu było stworzenie pętli sterowania z wizualnym sprzężeniem zwrotnym, która wykorzystuje model sieci splotowej dostosowany do pracy z obrazami RGB. Zadaniem sieci neuronowej była detekcja stanu robotów (pozycji i kątów). Każdy z robotów był oznaczonych jednym z unikalnych symboli (rysunek 4.12). Zadaniem sieci było znalezienia pozycji robota oraz pozycji symbolu. Wyznaczenie środka robota oraz pozycji symbolu pozwalało dodatkowo na wyznaczenie kąta obrotu robota. Wykorzystywany model iRobota nie posiada modułu Wi-Fi, stąd konieczne było opracowanie modułu umożliwiającego komunikację z robotami.

Podsumowanie zadań związanych ze sterowaniem robotem mobilnym (zestaw danych iRobot):

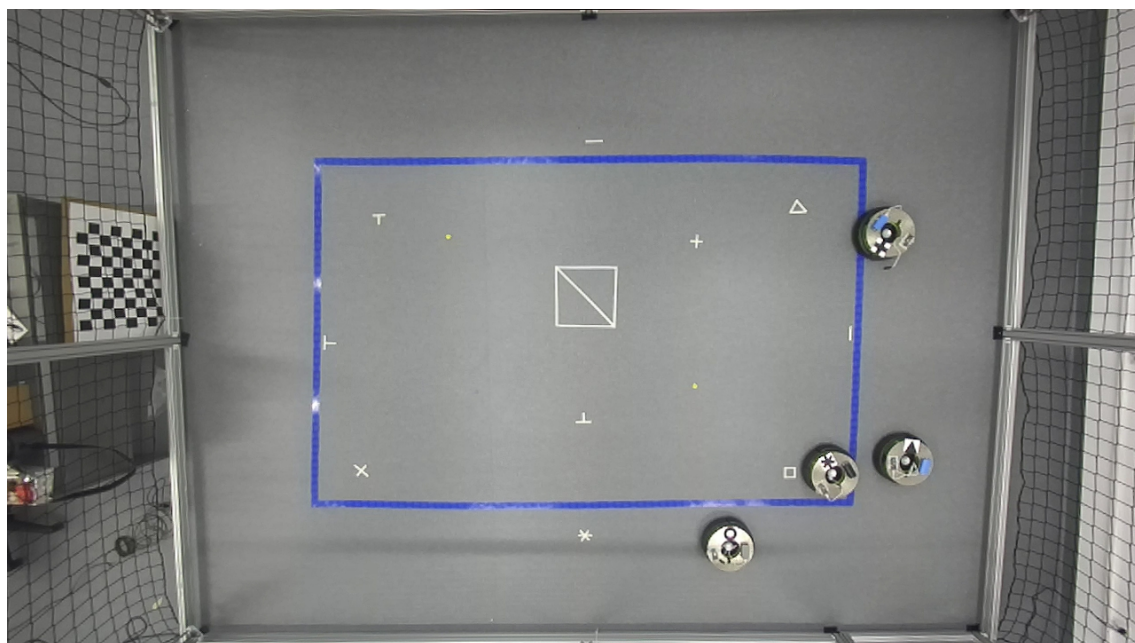
Dane wejściowe:	Obraz z kamery RGB $1280 \times 720 \times 3$
Zadanie główne:	Sterowanie ruchem 4 robotów mobilnych jednocześnie wzdłuż zadanej trajektorii.
Zadania uczenia maszynowego:	Detekcja i identyfikacja 4 robotów (oznaczonych różnymi symbolami) oraz czterech symboli, predykcja pozycji (x, y) we współrzędnych kamery, ustalenie orientacji (kąt α).
Zadania pomocnicze:	Komunikacja, moduł komunikacji bezprzewodowej, obsługa komunikatów MQTT.

4.6.2 Charakterystyka sterowanego obiektu iRobot 650

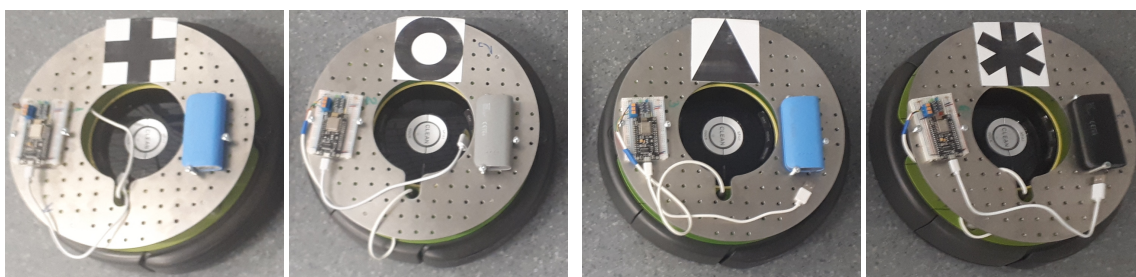
Wykorzystany w badaniach iRobot 650 typowo stosowany jest jako platforma autonomicznego odkurzacza. Platforma ta posiada otwarte API i stanowi wygodne narzędzie do testowania zadań z zakresu robotyki mobilnej. Średnica robota wynosi 348.5 mm, a jego wysokość to 92.25 mm [50]. Pojazd waży około 3.5 kilograma. Do poruszania się wykorzystuje dwa koła o średnicy 72 mm i rozstawie 235 mm. Za pomocą API dostępne są trzy tryby sterowania:

1. Sterowanie kołami niezależnie – niezależnie sterowanie prędkością każdego z kół, wartość $V \in [-500, 500]$ [mm/s],
2. Sterowanie kołami niezależnie – niezależnie sterowanie PWM każdego z kół, wartość w zakresie $[-255, 255]$,
3. Jazda po łuku o zadanym promieniu – niezależne określenie promienia skrętu $R \in [-2000, 2000]$ mm i prędkości $V \in [-500, 500]$ mm/s.

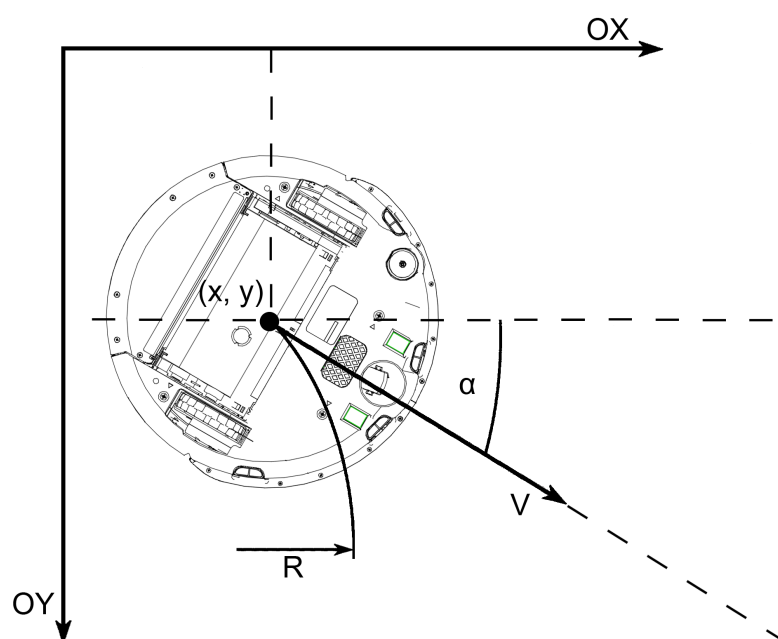
Na potrzeby eksperymentu wykorzystany został trzeci tryb sterowania – za pomocą



Rysunek 4.11: Widok z kamery obserwującej przestrzeń roboczą robotów. (Źródło: opracowanie własne)



Rysunek 4.12: Cztery iRoboty 650 wraz z rozróżniającymi je symbolami, widoczne umieszczenie układu ESP oraz akumulatora. (Źródło: opracowanie własne, publikacja [106])



Rysunek 4.13: Ilustracja sposobu sterowania robotem iRobot650, polegającego na ustaleniu promienia skrętu i prędkości. Dodatkowo zaznaczono układ współrzędnych, w którym przemieszcza się robot oraz kąt, pod jakim jest obrócony. (Źródło: własna publikacja [106])

prędkości i promienia skrętu, którego zasada działania została zilustrowana na rysunku 4.13.

Komunikacja z robotem, wykorzystująca API producenta, realizowana jest za pośrednictwem portu szeregowego. Roboty dostępne w laboratorium posiadają platformę umożliwiającą łatwy montaż dodatkowego wyposażenia pojazdu. Dla celów eksperymentów na pojeździe zamontowano układ do komunikacji oraz zasilający go akumulator. Układ bazuje na płytce ESP 8266 i służy jako odbiornik i przekaźnik komunikatów sterujących pomiędzy komputerem sterującym, a robotem mobilnym (rysunek 4.12). Komunikacja pomiędzy układem ESP 8266, a robotem odbywa się przewodowo, z wykorzystaniem portu szeregowego robota.

Na układ ESP 8266 WiFi wgrano prosty program odbierający i przekazujący

Tabela 4.8: Porównanie modeli YOLO.

Model	Liczba parametrów (mln)
YOLOv5l6 – large	76.8
YOLOv5m6 – medium	35.7
YOLOv5s6 – small	12.6
YOLOv5n6 – nano	3.2

komendy oraz wysyłający potwierdzenie odbioru. Komunikacja wykorzystuje protokół MQTT z wykorzystaniem biblioteki języka Python `paho-mqtt` po stronie komputera oraz głównej pętli sterującej. Każdy z robotów subskrybował dedykowany dla niego kanał.

4.6.3 Przygotowanie zbioru danych

Na potrzeby trenowania i testowania modelu detekcyjnego przygotowane zostały dwa zbiory danych (ozn. dalej *iRobot4*):

1. **Uczący** – zawierający około 100 000 klatek z filmów przedstawiających ruch robotów, zebranych z różnych kamer umieszczonych w różnych miejscach sceny.
2. **Testowy** – oddzielny zbiór złożony z 1 000 ujęć, przedstawiających przestrzeń roboczą robotów w wymagających warunkach oświetleniowych.

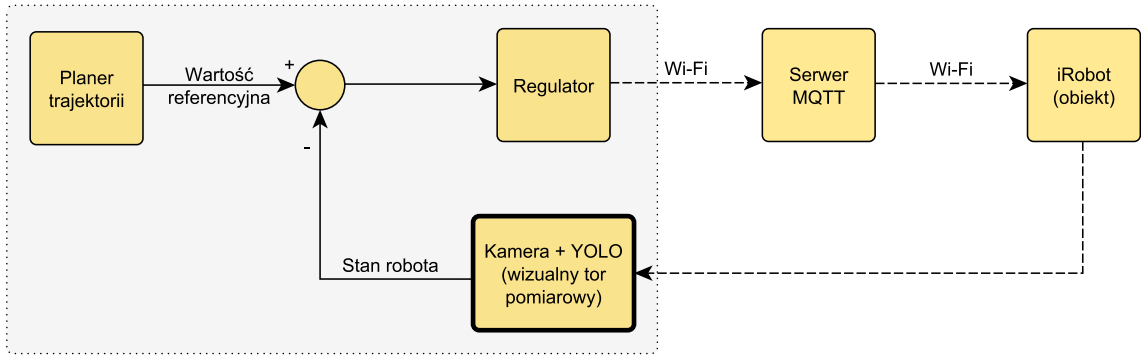
Na pojedynczą próbkę składa się obraz RGB oraz zestaw ośmiu par współrzędnych (x,y) , opisujących położenie 4 środków robotów oraz środków 4 symboli wyróżniających roboty w układzie współrzędnych obrazu – wartości te są więc wyrażone w pikselach.

Zebrane dane pochodzą z zarejestrowanych filmów, które przedstawiają roboty przemieszczające się zgodnie z wbudowanym w nie algorytmem, uwzględniającym odbicia od przeszkód. Ręcznie oznaczona została jedynie niewielka część klatek – zaledwie około 1.5% – przy użyciu których wytrenowano detektor obiektów. W kolejnym kroku posłużono się tym modelem do wykonania tagowania coraz większego zbioru danych, który ponownie wymagał wykonania pewnej małej liczby korekt dla obrazów błędnie oznaczonych. W ten sposób po kilku iteracjach w sposób półautomatyczny otagowano cały zbiór danych uczących, natomiast zbiór testowy był tagowany ręcznie.

4.6.4 Wykorzystany model sieci splotowej

W ramach eksperymentu jako narzędzie do detekcji wykorzystano różne warianty architektury YOLOv5 [56], dostosowane do pracy z obrazami o rozdzielczości 1280×720 pikseli. Rozpatrywano cztery warianty modelu, przedstawione w tabeli 4.8.

W badaniach wykorzystano modele wstępnie wytrenowane z użyciem zbioru COCO [74], które zostały następnie douczone na własnym zbiorze *iRobot4*. Wagi sieci optymalizowano z użyciem metody SGD. Trening wykonano na serwerze wydziałowym (z użyciem jednej karty graficznej NVIDIA A100), a predykcja oraz główna pętla sterująca wykonywane były na komputerze w laboratorium (z kartą graficzną NVIDIA RTX 3090).



Rysunek 4.14: Schemat układu sterowania robotami mobilnymi iRobot650. (Źródło: opracowanie własne)

4.6.5 Sterowanie

Wytrenowany detektor obiektów YOLO został wykorzystany w pętli wizualnego sprzężenia zwrotnego (rysunek 4.14) w układzie regulacji nadążnej, której celem było jednocześnie sterowanie ruchem robotów tak, aby podążały za zadanymi trajektoriami. Pomiar pozycji obiektów wykonywano za pomocą sieci spłotowej YOLO, zasilanej obrazami dostarczającymi przez kamerę. Stan układu obejmował 12 zmiennych: pozycje i kąty wszystkich czterech robotów. Oprócz detekcji za pomocą sieci wymagane były dodatkowe przeliczenia kątów obrotu robotów, korzystając z prostych zależności geometrycznych.

Algorytm sterowania wykorzystuje różnicę (oznaczoną jako e) pomiędzy aktualnym stanem robota a stanem zadanym i wyrażoną jako:

$$e(t) = \begin{bmatrix} e_x \\ e_y \\ e_\alpha \end{bmatrix} = \begin{bmatrix} x_{ref}(t) - x(t) \\ y_{ref}(t) - y(t) \\ \text{mod}_{2\pi}(\alpha_{ref}(t) - \alpha(t)) \end{bmatrix}, \quad (4.1)$$

gdzie $\text{mod}_{2\pi}$ to symetryczna funkcja modulo 2π , której wartości należą do przedziału $(-\pi, \pi]$. Z kolei wartości $x_{ref}(t)$ i $y_{ref}(t)$, $\alpha_{ref}(t)$ są to zadane w danej chwili współrzędne dla robota, a $x(t)$, $y(t)$, $\alpha(t)$ to wyznaczone z pomiarów wartości współrzędnych i kąta robota.

Kąt mierzony był pomiędzy dodatnią częścią osi OX , a wektorem łączącym środek robota oraz środek symbolu umieszczonego na robocie.

Posiadając informację o błędzie pozycji, regulator generował sygnał sterujący używając następującej reguły:

$$V = \min \left(k_1 \sqrt{e_x^2 + e_y^2}, 300 \right) \quad [\text{mm/s}] \quad (4.2)$$

$$R = \text{sign}(e_\alpha) * \frac{2000}{k_2 |e_\alpha|^{k_3} + 1} \quad [\text{mm}], \quad (4.3)$$

gdzie k_1 , k_2 i k_3 to pewne współczynniki wyznaczone doświadczalnie.

W ramach testowania – biorąc pod uwagę fakt, że standardowym zastosowaniem iRobota jest odkurzanie powierzchni płaskich – zaplanowano trajektorię, która z zadanym

promieniem pokrywa prostokątny obszar. Trajektorię zilustrowano na rysunku 4.16. Do jej stworzenia wykorzystano iteracyjny algorytm wykorzystujący generowaną w sposób losowy krzywą typu Peano, szczegółowo omówiony w pracy [96]. Trajektorię wygładzono używając krzywych interpolacyjnych Béziera – do tego celu skorzystano z funkcji `BSpline` pakietu `scipy.interpolate` [129]. Wygenerowana trajektoria została podzielona na cztery części o równej długości – po jednej dla każdego z robotów. Te cztery części stanowiły trajektorie zadane dla czterech robotów. Zadajnik wartości nadążnej zadawał kolejne pozycje należące do trajektorii w tempie dostosowanym do prędkości robotów.

4.6.6 Wyniki

Przed rozpoczęciem eksperymentu ze sterowaniem, dokonano wyboru wersji modelu YOLOv5 spośród czterech wariantów modeli dostosowanych do pracy z obrazami w rozdzielczości 1280×720 : YOLOv5n6, YOLOv5s6, YOLOv5m6, YOLOv5l6. Na podstawie uzyskanych wyników – czasu predykcji oraz jakości detekcji – wybrano model L, realizujący detekcję w czasie zbliżonym do odstępu pomiędzy rejestracją poszczególnych obrazów przez kamerę – obrazy rejestrowano z prędkością 30 klatek na sekundę, a czas odpowiedzi modelu to około 29.8 ms. Wybrany model charakteryzuje się jednocześnie wysoką jakością predykcji. Szczegółowe porównanie modeli zawarto w tabeli 4.9.

W tabeli 4.10 zamieszczono informacje o jakości predykcji modelu L dla poszczególnych klas. Obie tabele przedstawiają wyniki uzyskane na zbiorze testowym.

We wstępnej fazie uczenia modeli YOLO, gdy zbiór danych nie był wystarczająco zróżnicowany, napotymano na różnego rodzaju błędy w detekcji. Przykładowe błędne rozpoznania przedstawiono na rysunku 4.15. Błędy te są spowodowane głównie przez nietypowe oświetlenie sceny i w warunkach jednolitego oświetlenia nie występują.

Na rysunku 4.16 przedstawiono trajektorię przejechaną przez roboty oraz zadaną. Początkowe rozmieszczenie robotów było losowe, musiały więc dojechać do punktu rozpoczynającego trajektorię. Ruch robotów ustawał po osiągnięciu końcowego punktu trajektorii. Z kolei rysunek 4.17 przedstawia porównanie pokrycia powierzchni dla wbudowanego (losowego) algorytmu iRobota oraz dla prezentowanego rozwiązania. Warto podkreślić, że wbudowany algorytm pracował przez 100 sekund i zawiera wiele fragmentów nakładających się obszarów, natomiast proponowane rozwiązanie przez czas znacznie krótszy, wynoszący około 60 sekund.

Na rysunku 4.18 przedstawiono wartości zadane oraz zmienne stanu dla poszczególnych robotów. Widoczne na wykresach opóźnienie (około 0.5 sekundy) jest efektem zastosowania sterowania nadążnego. Brak wartości zadanej dla kąta wynika z tego, że sterowano robotem bez ustalania kąta obrotu. Konstrukcja fizyczna robota umożliwia poruszanie się tylko w ten sposób, że wektor kierunkowy robota jest styczny do toru ruchu – przykładowo robot nie może poruszać się w bok.

Dokładność odwzorowania trajektorii wyniosła 6.41 piksela w osi OX oraz 6.67 piksela w osi OY . Błąd ten wynosi około 10% średnicy robota na obrazie, wynoszącej około 60 pikseli. W trakcie eksperymentu testowego detektor YOLO nie popełnił żadnego błędu w detekcji robotów i znaczników – wszystkie roboty i symbole zidentyfikowano prawidłowo na wszystkich klatkach w trakcie przejazdu testowego.

W tabeli 4.11 przedstawiono dokładność odwzorowania trajektorii – uśrednioną po wszystkich próbkach różnicę między trajektorią zadaną a mierzoną. Pominięto początkowy

Tabela 4.9: Porównanie wariantów modeli YOLO z uwzględnieniem jakości i czasu predykcji.

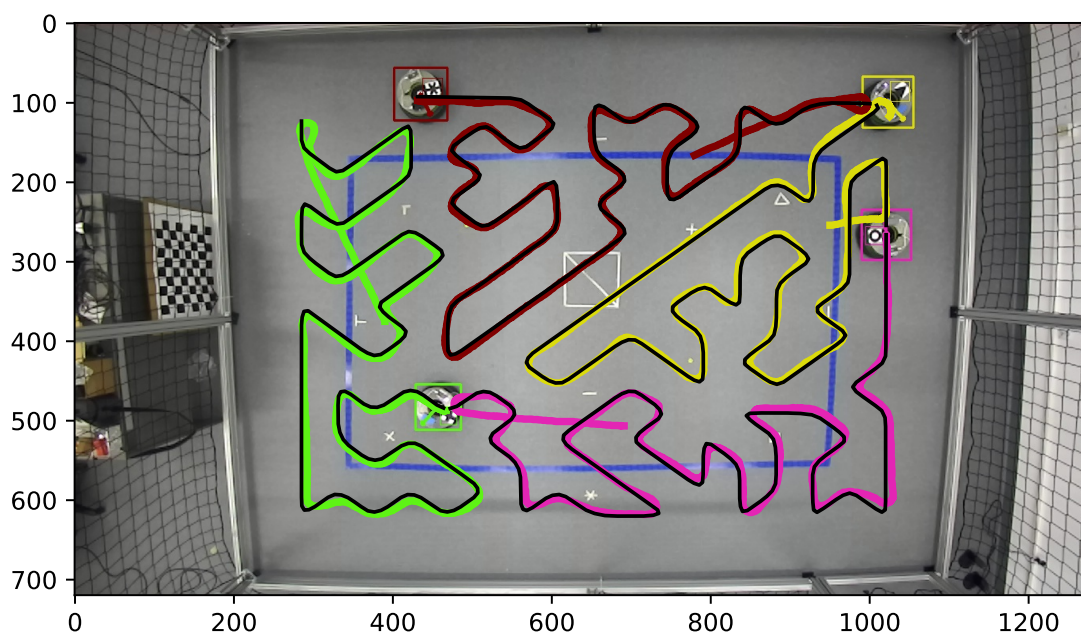
Model	Precyzja	Przywołanie	W pełni poprawnie rozpoznane obrazy	Czas predykcji na obraz
YOLOv5n6	0.980	0.936	68.4 %	17.69 ms
YOLOv5s6	0.938	0.946	59.9 %	15.53 ms
YOLOv5m6	0.977	0.951	77.3 %	28.03 ms
YOLOv5l6	0.985	0.966	81.5 %	42.22 ms

Tabela 4.10: Szczegółowe wyniki dla modelu YOLOv5l6 – przedstawiono precyzję, przywołanie, oraz średnią uśrednioną precyzję (mAP) przy progach 0.5 oraz 0.95.

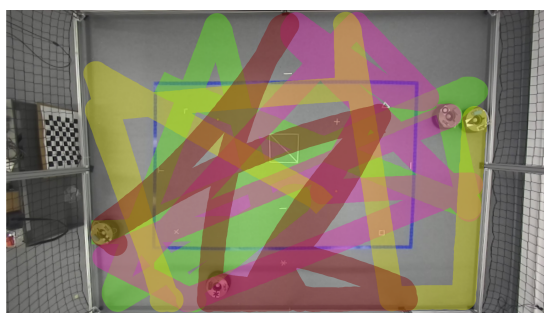
Klasa	Precyzja	Przywołanie	mAP@.5	mAP@.5:.95
cross	0.997	0.991	0.995	0.854
cross_center	0.901	1	0.99	0.936
circle	0.991	0.851	0.883	0.755
circle_center	0.957	0.931	0.983	0.904
triangle	0.992	0.88	0.912	0.762
triangle_center	0.99	0.897	0.973	0.861
star	0.996	1	0.995	0.87
star_center	0.957	1	0.995	0.956
łącznie	0.973	0.944	0.966	0.862



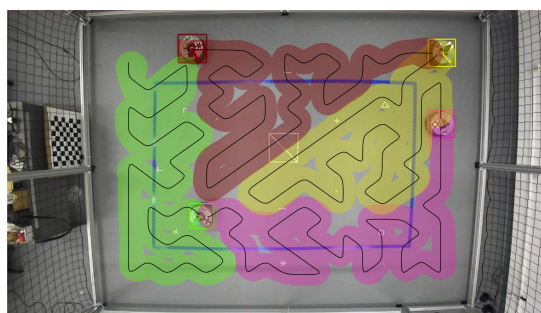
Rysunek 4.15: Przykłady błędnych detekcji modelu YOLO: po lewej – podwójna detekcja robota oznaczonego symbolem gwiazdy, na środku – fałszywie pozytywna detekcja: pętla kabla została rozpoznana jako trójkąt, po prawej fałszywy negatyw – nie rozpoznano trójkąta ani robota nim oznaczonego. (Źródło: opracowanie własne, publikacja [106])



Rysunek 4.16: Zadane (czarne line) oraz przejechane (kolorowe linie) trajektorie. Na rysunku roboty znajdują się w pozycjach docelowych. (Źródło: opracownie własne, publikacja [106])



(a)



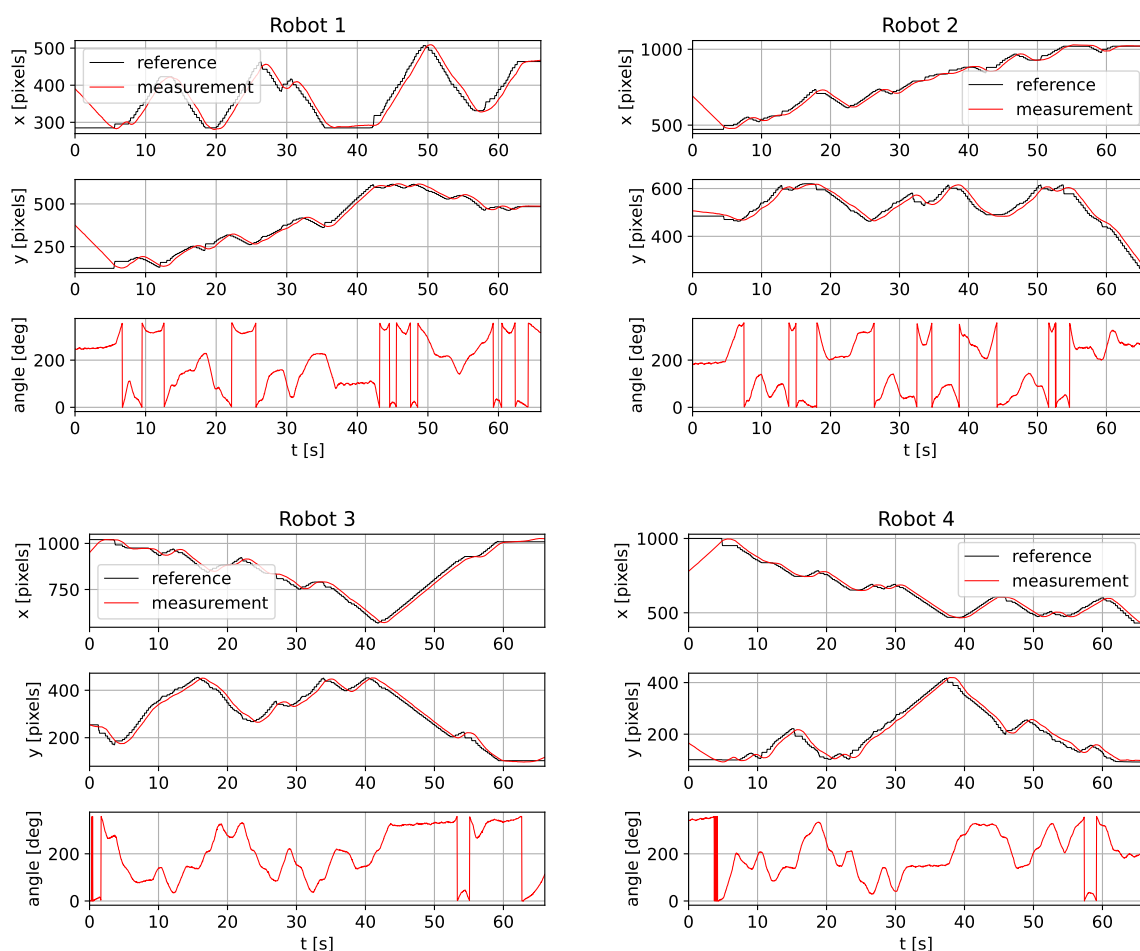
(b)

Rysunek 4.17: Porównanie trajektorii uzyskanych z użyciem (a) wbudowanego algorytmu iRota oraz (b) zaproponowanego rozwiązania. (Źródło: opracownie własne, publikacja [106])

fragment o długości około 5 sekund, gdy roboty dojeżdżały do początku trajektorii zadanej, startując od swoich położenia początkowych (fragment ten jest widoczny na rysunku 4.16).

Eksperyment pokazał, że głębokiej sieci splotowej można użyć nie tylko do śledzenia, ale i do sterowania robotami mobilnymi. Zastosowany model był wystarczająco szybki i dokładny do realizacji obu tych celów. Prezentowane wyniki dotyczą jakości pomiaru w oparciu wyłącznie o wyjście z modelu, bez wygładzania uzyskanych pomiarów. Wyniki można byłoby jeszcze poprawić, używając np. filtru Kalmana.

W eksperymencie wykorzystano kamerę szeroką kątem umieszczoną centralnie nad sceną, która miała wbudowaną korekcję efektu rybiego oka, aczkolwiek model YOLO

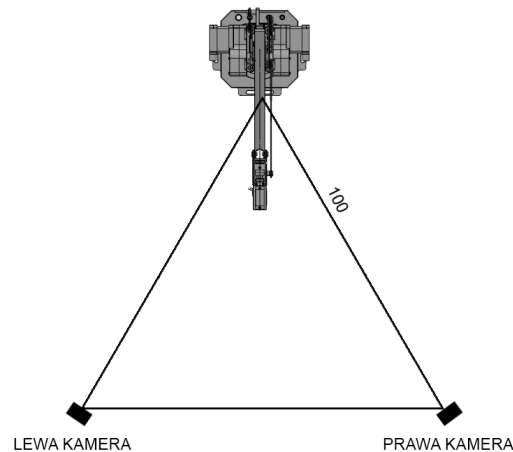


Rysunek 4.18: Trajektorie opisujące stan układu (pozycje i kąty) w rozbiu na poszczególne składowe dla wszystkich robotów w trakcie eksperymentu, reference (czarne linie) – trajektorie zadana, measurement (czerwone linie) – wyniki pomiarów pozycji za pomocą sprzężenia wizualnego. (Źródło: opracowanie własne, publikacja [106])

Tabela 4.11: Różnica w położeniu robotów między wartością zadaną i mierzoną w rozbiu na współrzędne, a także różnica w odległości.

Robot	błąd w osi X (e_x)	błąd w osi Y (e_y)	$e_r = \sqrt{e_x^2 + e_y^2}$
1: Cross	5.96	6.75	9.01
2: Circle	6.85	7.40	10.09
3: Triangle	6.46	5.93	8.77
4: Star	6.37	6.60	9.17
średnia	6.41	6.67	9.26

zachowuje się bardzo dobrze w przypadku obrazów bez korekcji. Model był również testowany dla kamery ustawianej pod kątem. W takich przypadkach konieczna byłaby korekcja między pozycją robota na obrazie a pozycją w przestrzeni fizycznej.



Rysunek 4.19: Rozmieszczenie kamer i ramienia robota. (Źródło: opracowanie własne)

4.7 Śledzenie ramienia robota uArm Swift Pro

4.7.1 Stanowisko badawcze

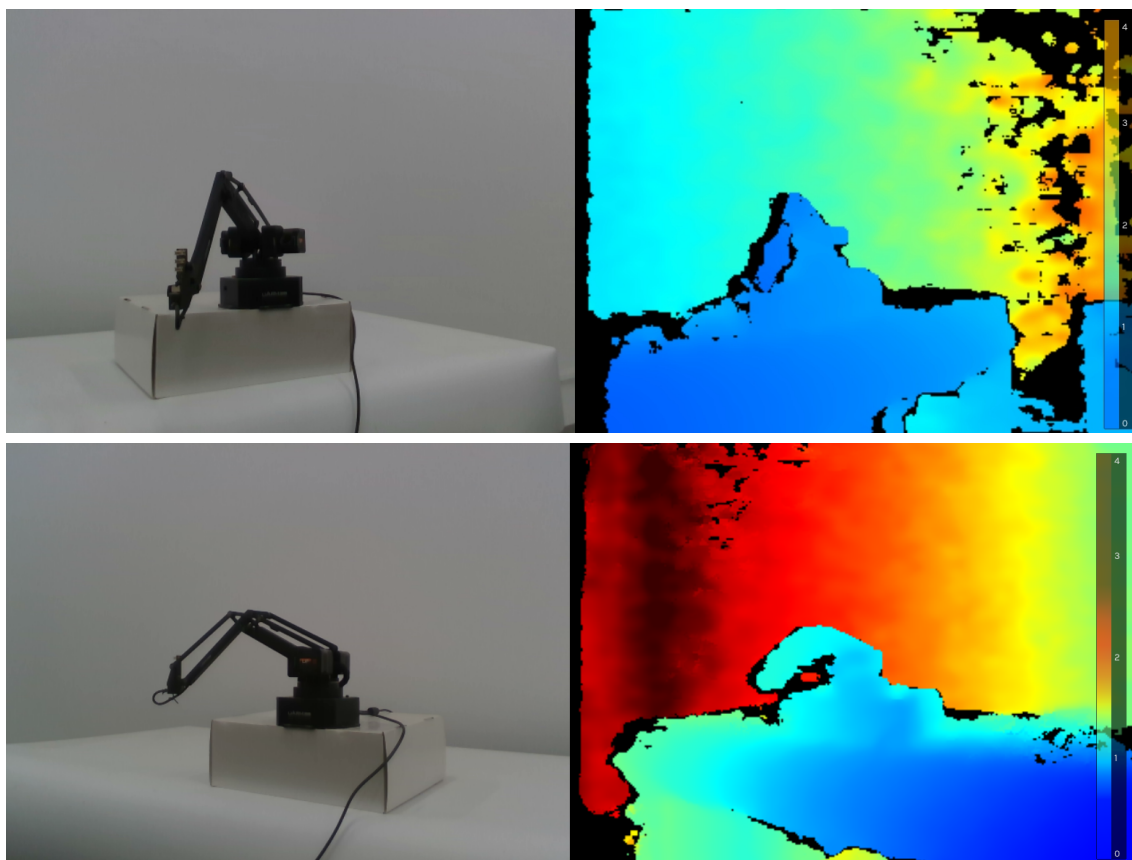
Przedmiotem kolejnego eksperymentu jest ramię robota uArm Swift Pro. Ramię umieszczone jest na podstawie i obserwowane przez układ dwóch kamery głębi Intel RealSense D435, umieszczonych w odległości metra od siebie oraz w odległości metra od podstawy robota. Kąt pomiędzy kamerami wynosi 60° . Poglądową ilustrację rozmieszczenia robota i kamer przedstawiono na rysunku 4.19. Z kolei na rysunku 4.20 zaprezentowano przykładowe ujęcia z lewej i z prawej kamery dla tej samej klatki.

W eksperymencie tym wykorzystano nowocześniejszy sensor wizyjny niż we wcześniejszym – a więc kamerę głębi – oraz ograniczono się tylko do śledzenia obiektu i nie zrealizowano sterowania. Przedstawione w tej sekcji wyniki zaprezentowano w pracy [107].

Poniżej przedstawiono podsumowanie problemu badawczego w zadaniu obserwacji ramienia robota:

Dane wejściowe	Obraz z kamery RGB-D o wymiarze: $640 \times 480 \times 4$ Chmura punktów o wymiarze: $750 \times 750 \times 1375$ Liczba klatek: $2 \cdot 5\,220$
Zadanie główne	Śledzenie ramienia robota predykcja trzech kątów skręcenia ramienia na podstawie danych z kamery głębi Intel RealSense
Zadania	Predykcja kątów α_0 (obrót robota) oraz kątów α_1, α_2 (kąty pomiędzy członami) Ustalenie pozycji efektora robota (x, y, z) – zadanie proste kinematyki

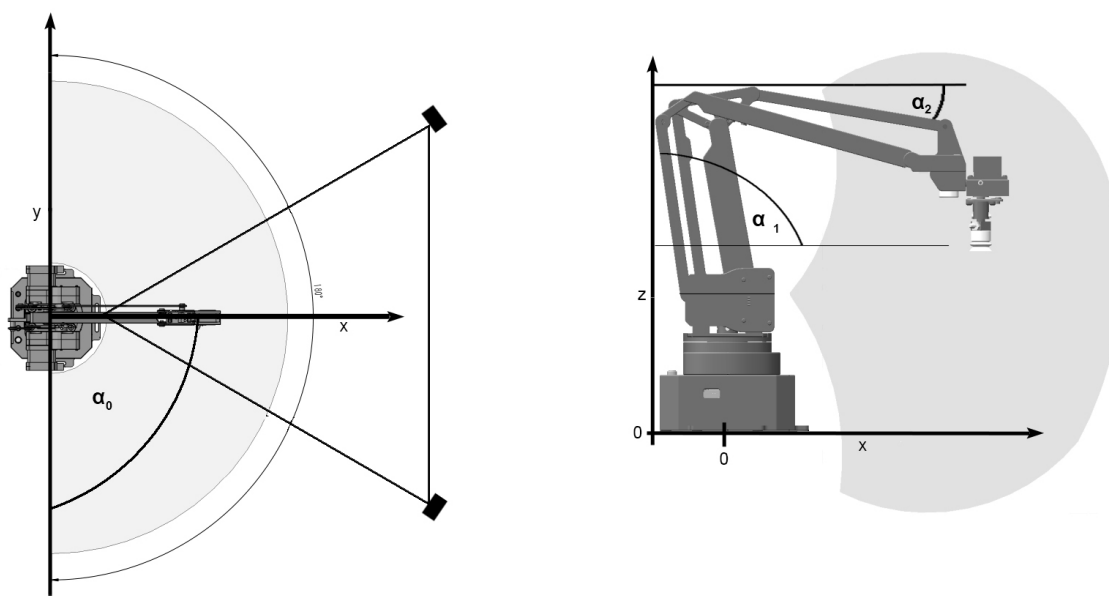
Kamera Intel RealSense D435 Kamera używa układu dwóch soczewek do stworzenia obrazu głębi [49]. Obraz ten może być dodatkowo poprawiony z użyciem projektora



Rysunek 4.20: Widok z lewej (góra) i prawej (dół) kamery. Po lewej stronie przedstawiono widok z kamery RGB, po prawej mapę głębi. (Źródło: opracowanie własne)

Tabela 4.12: Podstawowe parametry kamery głębi Intel RealSense D435 w oparciu o materiały dostępne na stronie internetowej producenta [49].

Parametr	Wartość
Wymiary	60 × 25 × 25 mm
Waga	72 g
Kąt widzenia dla głębi (FOV)	87° × 58° (pole widzenia w osi poziomej i pionowej)
Tryby nagrywania głębi	720p@6fps, 848p@10fps, VGA@30fps, 480p@60fps, 256p@90fps
Zakres głębi	0.3 m do 3 m (najlepsze obrazowanie w tym zakresie)
Dokładność głębi	< 2% dla odległości 2 m
Format zapisu	BAG (zawiera timestamp dla każdej klatki)
API w językach	Python, C, C++, C#, Matlab
Komunikacja	USB 3.1
Wymaga GPU	NIE



Rysunek 4.21: Oznaczenie kątów α_0 , α_1 , α_2 robota. (Źródło: opracowanie własne w oparciu o [124])

podczerwieni, co może być przydatne dla scen z małą ilością tekstur. Główne parametry kamery zebrano w tabeli 4.12.

Ramię robota uArm Swift Pro Zgodnie ze specyfikacją producenta [124], ramię robota uArm Swift Pro waży 2.2 kg i ma wymiary $150 \times 140 \times 281$ mm. Ramię posiada cztery stopnie swobody (rysunek 4.21) – przy czym ostatni dotyczy obrotu końcówki robota i nie był wykorzystywany w eksperymentach. Robot zasilany jest z użyciem dedykowanego zasilacza. Komunikacja z nim możliwa jest za pośrednictwem interfejsu USB 2.0 oraz Bluetooth 4.0. Po wykonaniu podłączenia możliwa jest komunikacja z robotem za pośrednictwem biblioteki uArm-Python-SDK [125].

4.7.2 Przygotowanie zbioru danych

Dane na potrzeby eksperymentu zbierano z dwóch źródeł: 1) z robota poprzez API producenta oraz 2) obraz z kamer poprzez API producenta kamery – bibliotekę pyreal-sense2 [123]. Oba źródła danych rejestrowały czasy pomiarów, jednak nie były one w żaden sposób synchronizowane. Z tego powodu do przygotowania otagowanego zbioru danych uczących konieczne było uwspólnienie czasów pomiarów pozycji robota z danymi z kamer. Efekt ujednolicenia czasu osiągnięto poprzez reinterpolację pomiarów kątów i pozycji robota do czasów próbkowania klatek – niezależnie dla każdej z kamer. Było to konieczne, ponieważ podstawa czasu oraz częstotliwość pomiaru dla obu urządzeń była różna. Odczyt z kamery pozyskiwano co około 0.167 sekundy (rejestrowano obraz w trybie 6 FPS w rozdzielczości 640×480), a odczyt z API robota odbywał co około 0.2 s. Obraz w formacie RGB-D transformowany do chmury punktów za pomocą biblioteki pyrealsense2. Dodatkowo wykonywano transformację (obrót i przesunięcie) widoku

z obu kamer do tego samego układu współrzędnych, związanego z układem ramienia (rysunek 4.21).

Warto podkreślić, że mimo transformacji chmury punktów do wspólnego układu obrazy rejestrowane przez obie kamery różnią się od siebie. Spowodowane jest to głównie tym, że chmury punktów zawierają jedynie punkty znajdujące się od strony patrzenia kamery.

W ramach eksperymentów przygotowane zostały dwa zbiory danych:

- **Uczący** – film przedstawiający robota pokrywającego siatkę możliwych kątów (4 240 ujęć z każdej z kamer),
- **Testowy** – oddzielny film, w trakcie którego robot przemieszcza się po trajektorii złożonej z 5 kwadratów (980 ujęć z każdej z kamer).

Link do zarejestrowanych filmów umieszczono w repozytorium GitHub [104]. Dodatkowo udostępniono tam modele i kod źródłowy, co umożliwia odtworzenie części predykcyjnej eksperymentu.

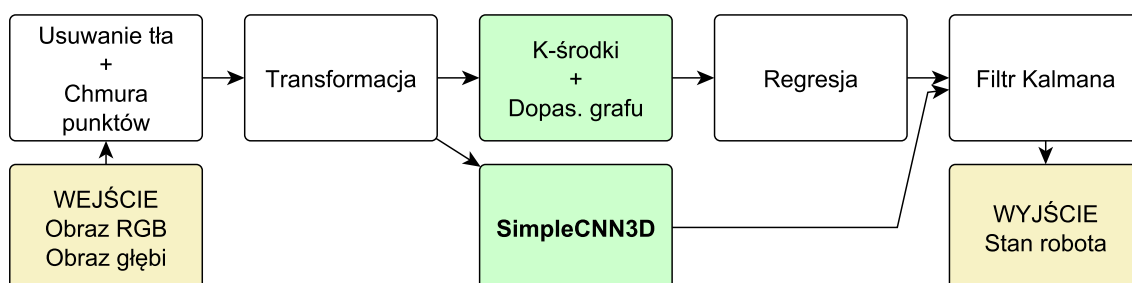
4.7.3 Wykorzystane procedury pomiaru kątów

W zagadnieniu pomiaru kątów skręcenia robota porównano dwa podejścia:

1. Predykcję kątów za pomocą sieci splotowej zawierającej rzadkie warstwy. Wykorzystano do tego celu rzadką sieć splotową SimpleCNN3D;
2. Dopasowanie grafu do chmury punktów pozyskanej z kamery głębi, a następnie predykcja kątów bezpośrednio z grafu (model GraphFitting).

Wspólną częścią obu podejść był preprocessing polegający: na usunięciu tła, wygenerowaniu chmury punktów na podstawie obrazu głębi oraz transformacji danych do układu współrzędnych niezależnego od kamery. Cały potok przetwarzania przedstawiono na rysunku 4.22.

Model sieci głębokiej Model SimpleCNN3D został szczegółowo omówiony w rozdziale 3. Model ten na wejście przyjmuje chmurę punktów po usunięciu tła, natomiast jego wyjściem są kąty skręcenia ramienia. Model ten zaimplementowano z użyciem frameworków PyTorch [93] oraz SpConv [116].



Rysunek 4.22: Potok przetwarzania dla zadania śledzenia ramienia robota uArm Swift Pro. (Źródło: opracowanie własne)

Dopasowanie grafu – model GraphFitting Drugie podejście polegało na dopasowaniu grafu (w tym przypadku bardzo prostego, będącego układem dwóch odcinków) do chmury punktów pomiarowych, a następnie wyznaczeniu kątów z zależności geometrycznych. Procedura dopasowania grafu (GraphFitting) składała się z następujących etapów:

1. Wykonanie algorytmu k -środków na chmurze punktów w celu redukcji liczby punktów;
2. Właściwa procedura dopasowania grafu;
3. Podniesienie wymiarowości danych oraz regresja kątów na podstawie rozmieszczenia węzłów grafu.

Pierwszym etapem obróbki danych było usunięcie statycznego tła. W tym celu wykorzystano algorytm k -środków, który dla losowej sekwencji wektorów-obrazów znajdował dwa centra: obraz reprezentujący tło oraz pierwszy plan. Kolejnym krokiem była transformacja pikseli należących do pierwszego planu do chmury punktów. Wizualizację tych dwóch początkowych etapów przetwarzania wstępnego danych przedstawiono na rysunku 4.23.

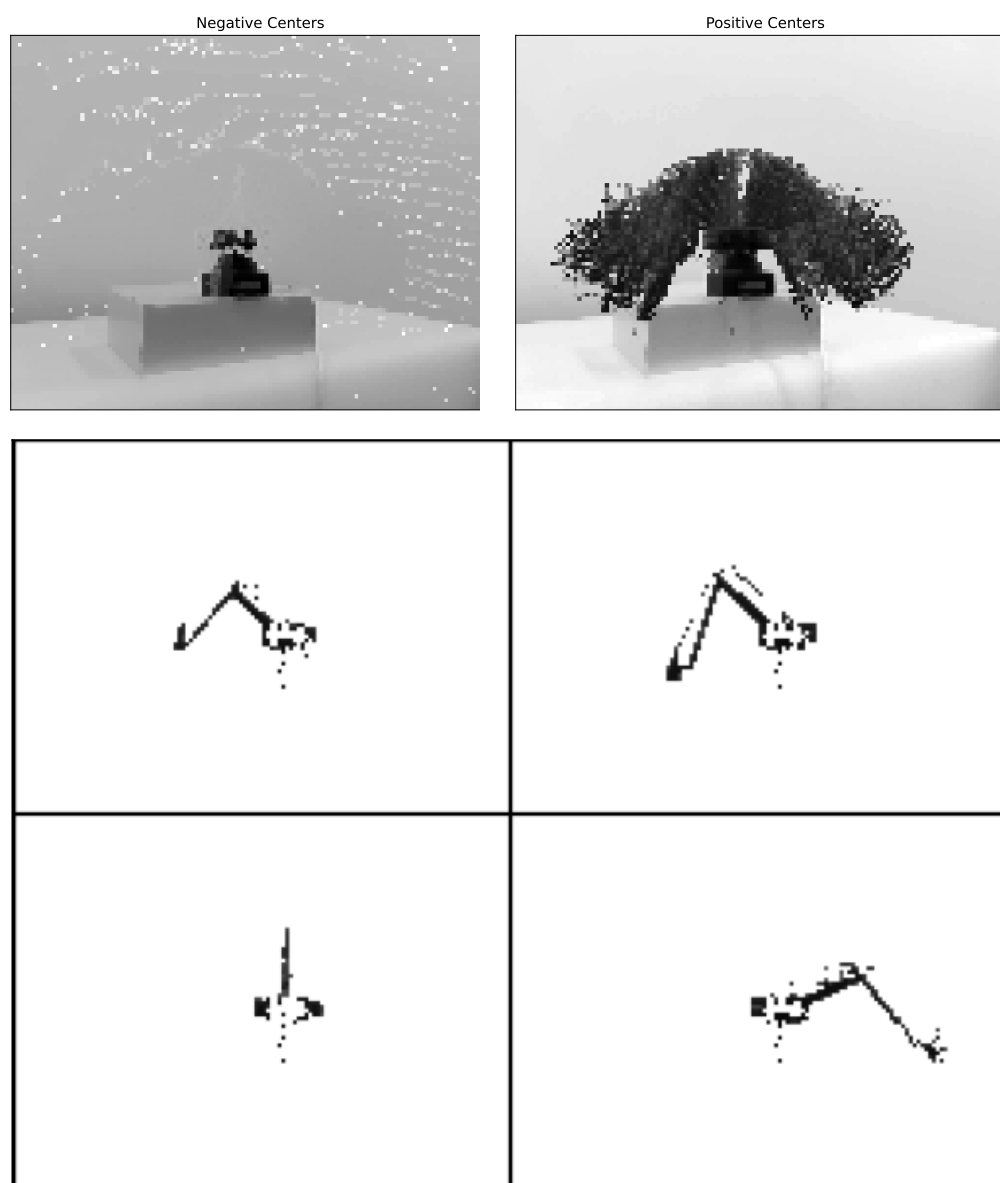
W kolejnym etapie chmura punktów po redukcji liczebności (do 20 – 80 punktów) stanowiła dane wejściowe do procedury dopasowujące położenie wierzchołków grafu C . Celem optymalizacji jest minimalizacja odległości pomiędzy punktami a grafem o ustalonej strukturze. Omówiona procedura jest możliwa do zastosowania dla dowolnego grafu, natomiast w badaniach wykorzystano graf, będący układem dwóch odcinków, co ma przypominać budowę ramienia robota (rysunek 4.24b). Możliwe jest uproszczenie zadania optymalizacji poprzez ustalenie pewnych wartości brzegowych grafu, np. pozycji początkowej oraz długości ramion, jednak w eksperymentach tego nie robiono. Graf, oprócz trzech najbardziej charakterystycznych węzłów, posiada także węzły pośrednie (w liczbie od 3 do 6) rozmieszczone równomiernie (ozn. C_{ext}). Węzły te mają za zadanie lepsze określenie wartości funkcji dopasowania grafu do punktów.

Optymalizacja dopasowania grafu do punktów wykonywana jest w oparciu o metrykę zdefiniowaną poniższym wzorem:

$$\text{dist}(C_{ext}, X) = \left\{ \sum_{x \in X} \min_{c \in C_{ext}} d_2(c, x) + \sum_{c \in C_{ext}} \min_{x \in X} d_2(c, x) \right\}. \quad (4.4)$$

Miara ta jest koncepcyjnie zbliżona do metryki Hausdorffa (użyto $\sum$ zamiast $\max$). Odległość grafu od chmury punktów jest sumą minimalnych odległości euklidesowych pomiędzy węzłami grafu, a chmurą punktów oraz chmurą punktów, a węzłami grafu. Ilustrację zastosowania metryki do przedstawiono na rysunku 4.24, natomiast w tabeli 4.13 wyniki eksperymentów dotyczących wyboru liczby centrów oraz liczby punktów grafu na czas i jakość predykcji.

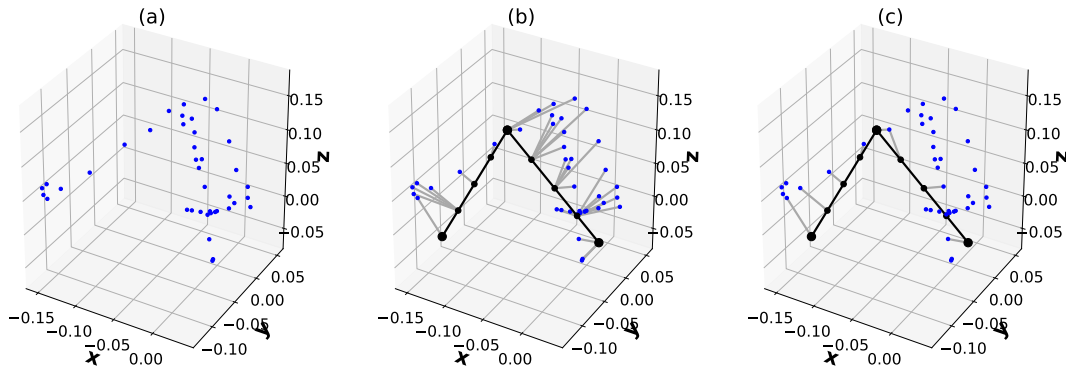
Uzyskane dzięki optymalizacji położenie węzłów grafu może być użyte bezpośrednio do predykcji kątów skręcenia ramienia. Ponieważ jednak wyniki takiego podejścia nie były zadowalające, wykorzystano nieliniowy model regresyjny, który na podstawie uzyskanych współrzędnych węzłów grafu przewidywał wartości kątów.



Rysunek 4.23: Wizualizacja centrów: reprezentujących tło (po lewej u góry) oraz pierwszy plan (po prawej u góry), oraz przykładowe klatki przedstawiające chmury punktów po usunięciu tła (obrazy poniżej). (Źródło: opracowanie własne, publikacja [107])

Przetestowano trzy metody o strukturze dwuwarstwowej dostępne w pakiecie scikit-learn [94]:

1. Dwuwarstwową sieć neuronową (MLP) wykorzystującą procedurę MLPRegressor z funkcją aktywacji $\tanh$ w warstwie ukrytej oraz liniową w warstwie wyjściowej.
2. Metodę SGD wykorzystującą przekształcenie jądrowe typu RBF wraz z aproksymacją Nyströma (funkcja `kernel_approximation.Nystroem` z pakietu `sklearn`) [140] z warstwą liniową strojona metodą SGDRegressor w oparciu o kryterium błędu MSE.



Rysunek 4.24: Miara jakości dopasowania grafu. Czarną linią oznaczono ramię robota, czarnymi punktami oznaczają węzły grafu C oraz punkty pośrednie C_{ext} , a niebieskimi punktami oznaczono chmurę punktów. Miara dopasowania $\text{dist}(C_{ext}, X)$ określona przez (4.4) jest sumą odcinków przedstawionych na wykresach (b) (łączyjących dla każdego punktu najbliższe węzły grafu) i (c) (łączyjących dla każdego węzła grafu najbliższe punkty chmury punktów). (Źródło: opracowanie własne, publikacja [107])

3. Metodę Huber wykorzystującą jak powyżej przekształcenie jądrowe typu RBF wraz z aproksymacją Nyströma, ale z warstwą liniową strojona metodą Hubera (funkcja `HuberRegressor` [48]) w oparciu o aproksymację błędu MAE.

Tabela 4.14 przedstawia porównanie jakości predykcji trzech testowanych modeli regresyjnych na danych uczących (na walidacyjnym podzbiórze zbioru uczącego), a przebiegi predykcji dla tych modeli na zbiorze uczącym przedstawiono na rysunku 4.25. Na podstawie wyników zaprezentowanych w tabelach 4.13, 4.14 oraz na rysunku 4.25 do dalszych badań wybrano model Huber. Ustalono też liczbę centrów w algorytmie k -środków na 40, a liczbę punktów rozszerzających graf C_{ext} na 4. Eksperyment wykazał, że czas predykcji wzrasta nieznacznie wraz ze wzrostem k , natomiast nie był widoczny związek między liczbą węzłów grafu a czasem. Oba parametry miały niewielki wpływ na jakość predykcji, co świadczy w pewnym stopniu o odporności metody.

4.7.4 Wyniki

W tabeli 4.15 przedstawiono szczegółowe zestawienie dokładności porównywanych modeli: `SimpleCNN3D` oraz `GraphFitting` na zbiorze uczącym i testowym. Przebiegi trajektorii zadanych oraz predykcje modeli przedstawione są na rysunkach: 4.26, 4.27, 4.28 i 4.29. Porównano tam dokładność predykcji kątów dla dopasowania grafu oraz rzadkiej splotowej sieci na danych uczących i testowych oraz dla kamer lewej i prawej. rysunki 4.30 i 4.31 przedstawiają predykcje końcówki ramienia na zbiorze testowym. Dokładność przewidywania pozycji efektora robota wykonano rozwiązując zadanie proste kinematyki, w oparciu o wartości kątów będące wynikiem predykcji.

W eksperymentach wykorzystano niezależne dwie sekwencje wideo z dwóch niezależnych kamer umieszczonych w odległości około 1 metra od ramienia robota i w odległości około 60 stopni między nimi. Trajektoria testowa została przygotowana w taki sposób, że na określonej części trajektorii ramię robota znajdowało się bezpośrednio przed lewą kamerą (patrz rysunek 4.21). Jest to przypadek trudniejszy do zmierzenia z ujęcia z lewej

Tabela 4.13: Wpływ liczby punktów używanych w algorytmie k –środków (N_{kmeans}) oraz liczby węzłów pośrednich grafu (N_{graph}) na czas i jakość obliczeń. (err_{α_i} oznacza błąd MAE dla każdego z kątów, $\overline{err}_{\alpha}$ to średni błąd trzech kątów na zbiorze testowym (Źródło: opracowanie własne).

N_{kmeans}	N_{graph}	czas [ms]	err_{α} [stopnie]			$\overline{err}_{\alpha}$ [deg]
			α_0	α_1	α_2	
20	3	28.67	1.25	4.05	2.45	2.58
	4	28.32	1.24	3.99	2.35	2.53
	5	28.48	1.20	4.08	2.37	2.55
	6	28.47	1.24	4.05	2.43	2.57
40	3	31.66	1.19	3.66	2.22	2.36
	4	31.91	1.21	3.61	2.26	2.36
	5	31.80	1.20	3.72	2.17	2.36
	6	31.86	1.14	3.65	2.27	2.35
60	3	34.98	1.20	3.52	2.24	2.32
	4	35.12	1.15	3.69	2.22	2.35
	5	35.01	1.17	3.55	2.23	2.32
	6	34.99	1.19	3.49	2.24	2.31

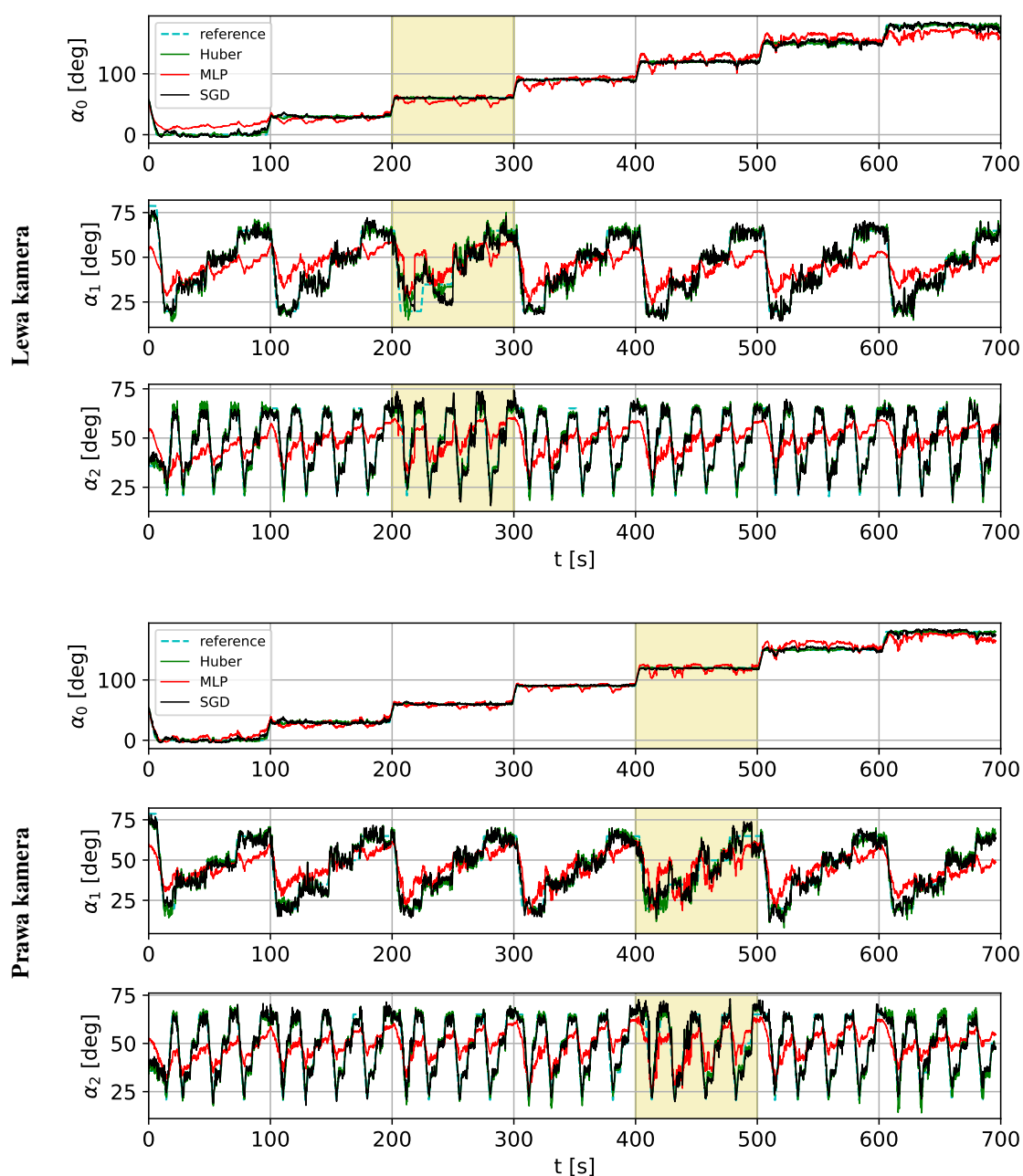
Tabela 4.14: Porównanie błędów modeli regresyjnych.

Model	Lewa kamera				Prawa kamera			
	α_0	α_1	α_2	$\overline{err}_{\alpha}$ [stopnie]	α_0	α_1	α_2	$\overline{err}_{\alpha}$
MLP	8.19	10.12	10.13	9.48	5.08	8.22	8.75	7.35
SGD	2.18	3.10	2.26	2.51	1.89	3.50	2.37	2.59
Huber	1.18	2.83	1.91	1.97	0.98	3.13	2.08	2.06

kamery, jednak ustalony celowo, aby zaobserwować różnice w jakości pomiaru w tych dwóch wariantach.

Jak oczekiwano, dokładność zestawu testowego dla lewej kamery (rysunek 4.30) jest nieco niższa niż dla prawej kamery (rysunek 4.31). Gdy kamera patrzy z boku, identyfikacja kątów jest względnie prosta, natomiast w pewnych ułożeniach – gdy kamera patrzy na wprost – występują błędy pomiaru. Na pierwszym rysunku zaznaczony jest fragment, gdy kamera znajduje się przed ramieniem robota. W drugim przypadku prawa kamera patrzy na ramię z boku. Średnia dokładność oszacowania kąta na podstawie widoku z kamery głębi wynosi mniej niż 3 stopnie, a błąd dokładności szacowania położenia robota jest mniejszy niż 20 mm.

W ramach badań wykorzystujących ramię robota uArm Swift Pro sprawdzono także czas predykcji dla obu metod. Wyniki uzyskanych pomiarów zaprezentowane zostały w



Rysunek 4.25: Porównanie jakości predykcji kątów modeli regresyjnych (MLP, SGD i Huber) wspomagających procedurę GraphFitting dla danych zebranych z lewej oraz prawej kamery. Przedstawiono wyniki na połączeniu danych uczących i walidujących (podział 70:30). Zacienione obszary oznaczają kąty, dla których ramię robota umiejscowione jest na wprost kamery. (Źródło: opracowanie własne)

tabeli 4.16. W tym eksperymencie trening sieci splotowej wykonano na komputerze z 6-rdzeniowym procesorem i7-8700 (3.20GHz) oraz 16 GB RAM, bez użycia GPU. Predykcję (testowanie) dla obu metod wykonywano na tym samym komputerze, wymuszając dodatkowo pracę w jednym wątku.

Tabela 4.15: Zestawienie dokładności porównywanych modeli: SimpleCNN3D oraz Graph-Fitting na zbiorze uczącym i testowym. α_0 , α_1 , α_2 , x , y , z oznaczają błędy bezwzględne dla kątów i pozycji w odniesieniu do rzeczywistego stanu robota; err_α to średnia wartość, a $err_{pos} = \sqrt{x^2 + y^2 + z^2}$ to błąd pozycji.

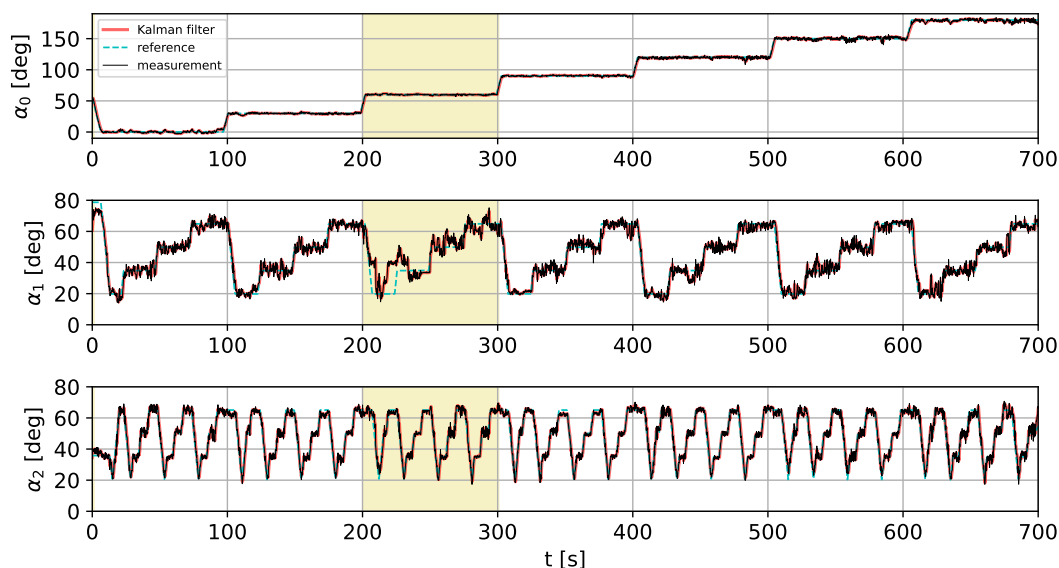
Kamera	Model	Zbiór	α_0	α_1	α_2	x	y	z	err_α	err_{pos}
			[degree]			[mm]			[deg]	[mm]
lewa	CNN	train	1.14	1.58	1.90	5.80	6.18	6.14	1.54	10.46
		test	2.68	3.95	3.06	12.54	14.98	9.75	3.23	21.84
	Graph	train	1.01	2.52	2.01	6.83	6.09	8.38	1.85	12.41
		test	1.49	4.42	2.00	9.38	9.27	12.73	2.64	18.32
prawa	CNN	train	0.98	1.63	2.16	6.09	5.64	6.88	1.59	10.78
		test	2.06	2.25	2.23	9.77	8.48	6.85	2.18	14.64
	Graph	train	0.87	2.79	2.14	6.91	5.61	8.79	1.93	12.51
		test	1.08	2.58	1.68	6.61	6.48	7.22	1.78	11.74

Tabela 4.16: Czasy przetwarzania poszczególnych operacji zgodnie z potokiem przetwarzania przedstawionym na rysunku 4.22.

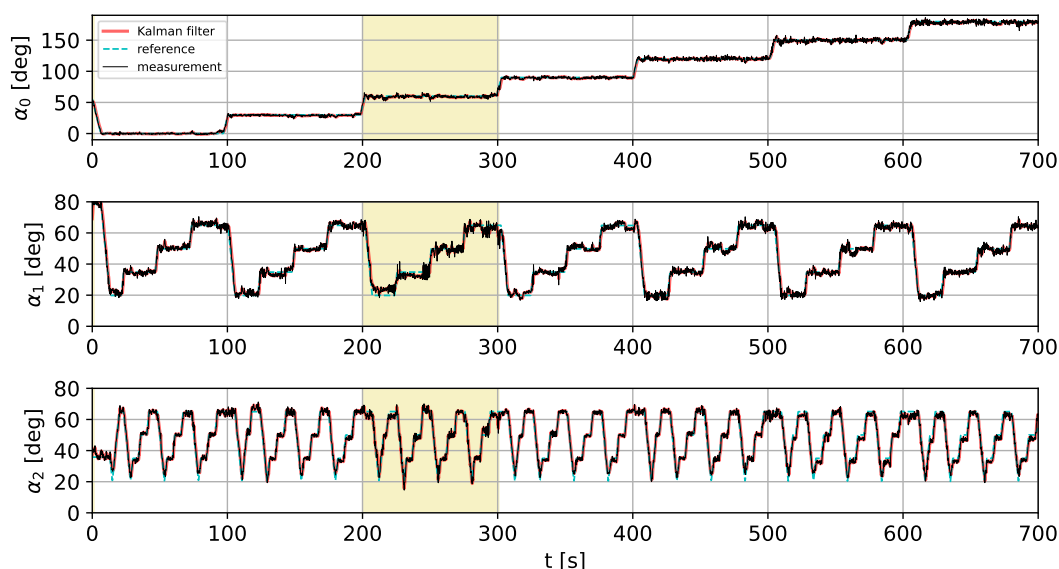
Model	Zadanie	Średni czas na klatkę [ms]
Preprocessing (wspólny)	Pobr. klatki z kamery	7.07
	Usuwanie tła	1.20
	Chmura punktów	4.46
	Czas preprocesingu:	12.73
GraphFitting	k -środki	8.55
	Graph fitting	10.47
	Transformacja + Regresja	$\ll$ 1.00
	Czas całkowity:	31.76
SimpleCNN3D	Predykcja	7.03
	Czas całkowity:	21.94

Jak pokazuje pomiar czasu, sieć SimpleCNN3D pozwala na uzyskanie znacznie krótszego czasu predykcji – całkowity czas przewidywania kątów opisujących ułożenie ramienia robota (z preprocessingiem wspólnym dla obu metod) jest o około 30% krótszy niż dla dopasowania grafu i wynosi 21.94 ms. Jest to szczególnie istotny wynik, ponieważ uzyskany został dla sieci spłotowej uruchamianej w jednym wątku procesora, bez użycia karty graficznej i technik kwantyzacji modelu. Sugeruje to, że przedstawioną strukturę można rozbudować, poprawiając jakość jej predykcji, przy zachowaniu konkurencyjnego czasu obliczeń. Uzyskanie tak wysokiej szybkości predykcji możliwe jest dzięki zastosowaniu warstw spłotowych przystosowanych do pracy z danymi rzadkimi.

Zbiór uczący, Lewa kamera, Kąty, GraphFitting

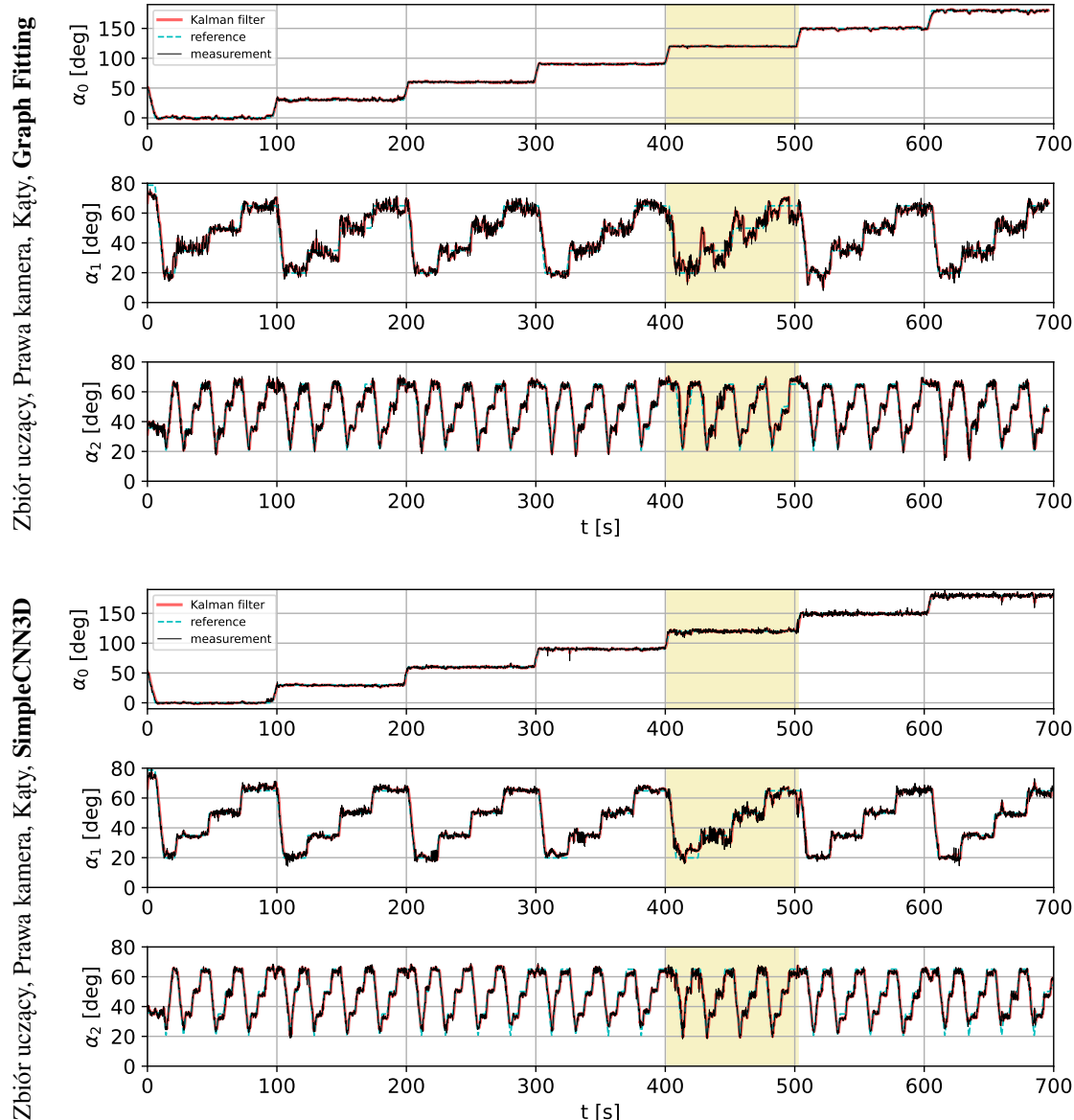


Zbiór uczący, Lewa kamera, Kąty, SimpleCNN3D



Rysunek 4.26: Porównanie dokładności przewidywania kątów dla GraphFitting i SimpleCNN3D na zbiorze uczącym (lewa kamera). Zacieniowanym obszarem oznaczono fragmenty trajektorii, w których ramię robota jest skierowane na wprost kamery. (Źródło: opracowanie własne, publikacja [107])

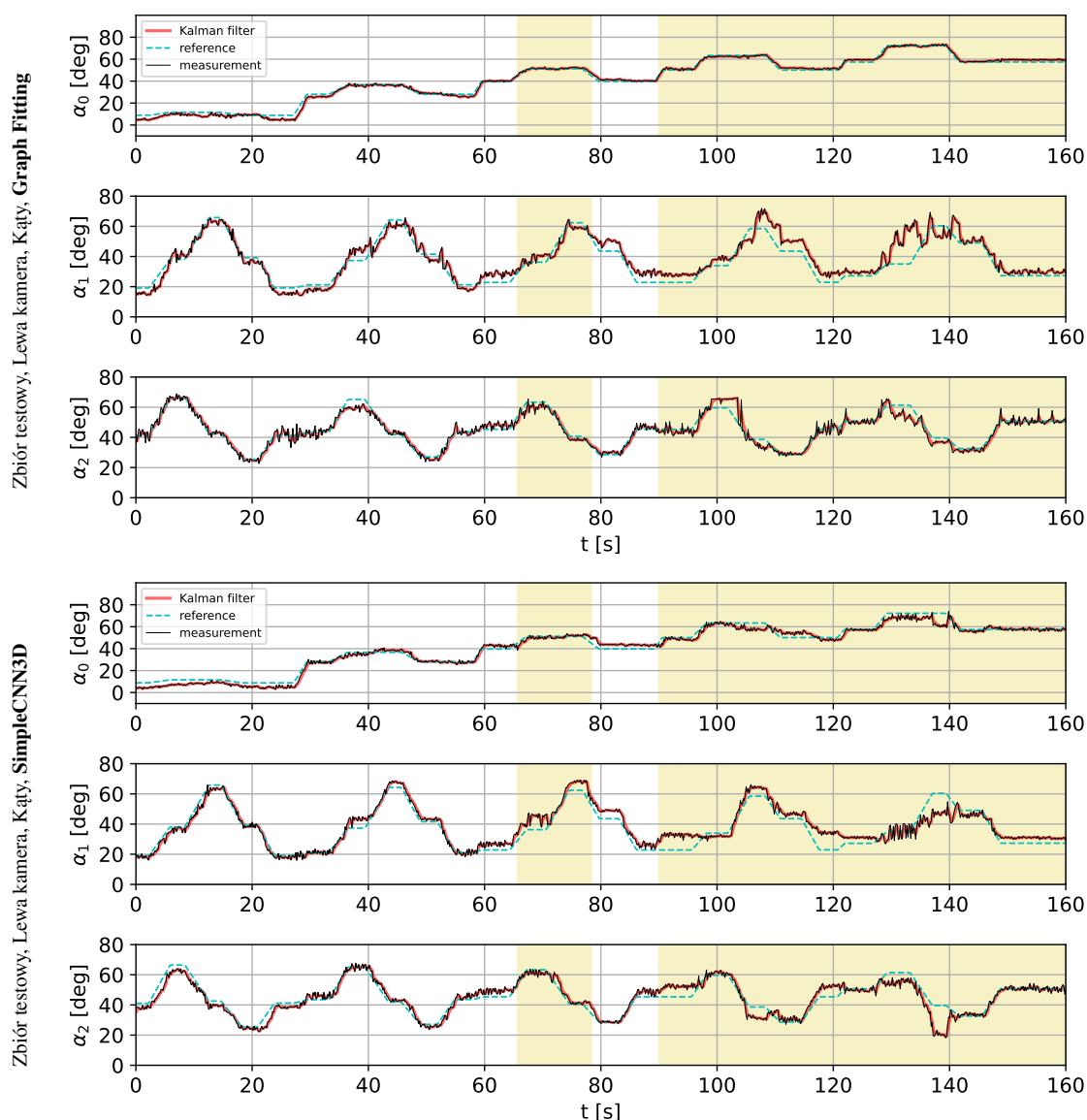
Podsumowanie wyników Eksperymenty pokazują, że model GraphFitting miał nieco lepszą dokładność niż sieć splotowa z warstwami rzadkimi. Przedstawiona dokładność predykcji kątów wahała się w granicach 1.8° – 3.3° dla modelu GraphFitting oraz około 2.6° – 3.3° dla modelu SimpleCNN3D. Dokładność pomiaru odległości dla czujnika głębokości RealSense D435 wynosi około 2% zmierzonej odległości. W skali prezentowanego eksperymentu dokładność ta była więc na poziomie 1 – 2 cm. Zgodnie z deklaracją producenta ramienia robota uArm Swift Pro, dokładność pozycjonowania wy-



Rysunek 4.27: Porównanie dokładności przewidywania kątów dla GraphFitting i SimpleCNN3D na zbiorze uczącym (prawa kamera). Zacieniowanym obszarem oznaczono fragmenty trajektorii, w których ramię robota jest skierowane na wprost kamery – może to powodować obniżenie dokładności. (Źródło: opracowanie własne, publikacja [107])

nosi około 0.2 mm, czyli jest o rząd wielkości wyższa niż dokładność kamery RGB-D. W przypadku obu modeli dokładność pomiaru położenia robota mieści się w zakresie 10 – 20 mm, czyli jest na poziomie dokładności czujnika D435, lub nawet nieco poniżej. Uzyskano więc wysoką jakość predykcji, na poziomie zbliżonym do błędów wprowadzanych przez system pomiarowy, przy bardzo krótkim czasie przetwarzania.

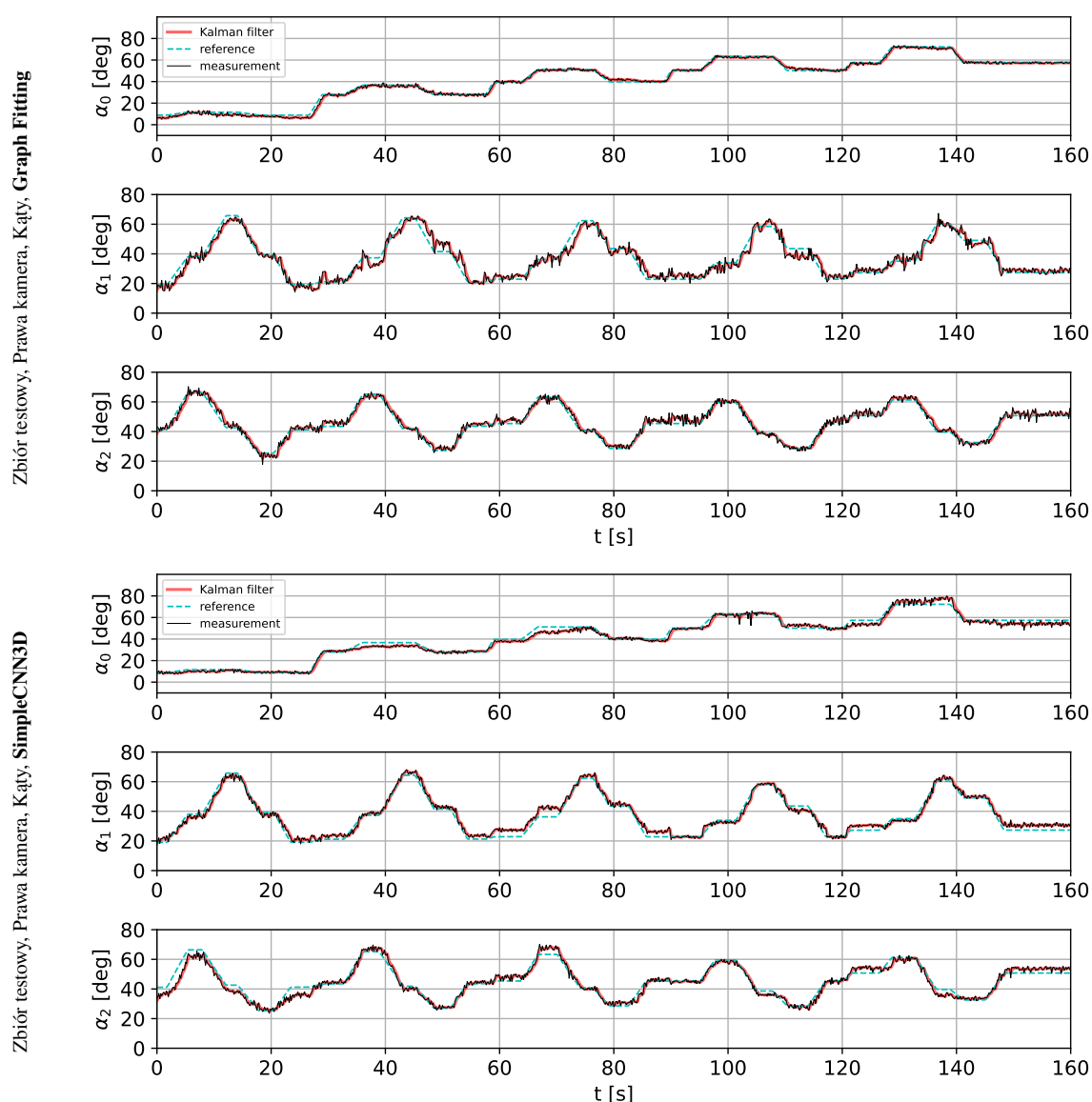
Wyniki pokazują, że oba przedstawione rozwiązania działają w czasie rzeczywistym z szybkością śledzenia od około 30 FPS (dla podejścia GraphFitting) do 50 FPS (dla splo-



Rysunek 4.28: Porównanie dokładności przewidywania kątów dla GraphFitting i SimpleCNN3D na zbiorze testowym (lewa kamera). Zacieniowanym obszarem oznaczono fragmenty trajektorii, w których ramię robota jest skierowane na wprost kamery. (Źródło: własna publikacja [107])

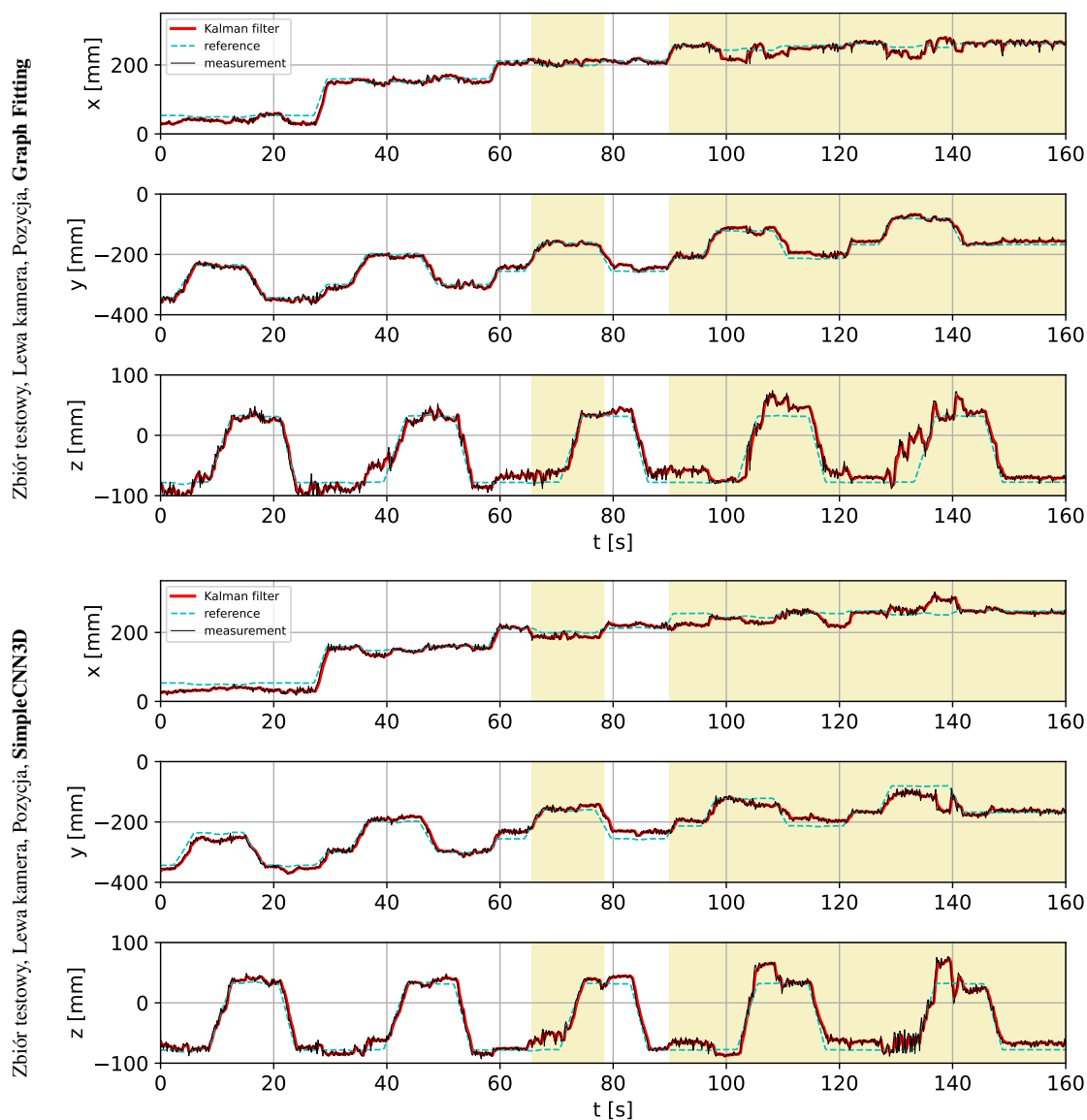
towego modelu SimpleCNN3D) dla obliczeń na procesorze w modelu jednowątkowym. Daje to duży potencjał na przyspieszenie obliczeń, tym bardziej, że model sieci splotowej jest znacznie lepiej dostosowany do środowisk obliczeniowych wykorzystujących GPU. Rozbudowa struktury sieci o kolejne warstwy nie musiałaby wydłużyć czasu predykcji, pod warunkiem zmiany urządzenia obliczeniowego na kartę graficzną.

Jest także jasne, że przedstawione podejście nie jest konkurencyjne pod względem szybkości czy dokładności w stosunku do czujników pomiarowych stosowanych w robocie. Jak jednak pokazują eksperymenty, za pomocą kamery głębi możliwa jest obserwacja pozycji i kątów skręcenia z dokładnością na poziomie dokładności sensora wizyjnego

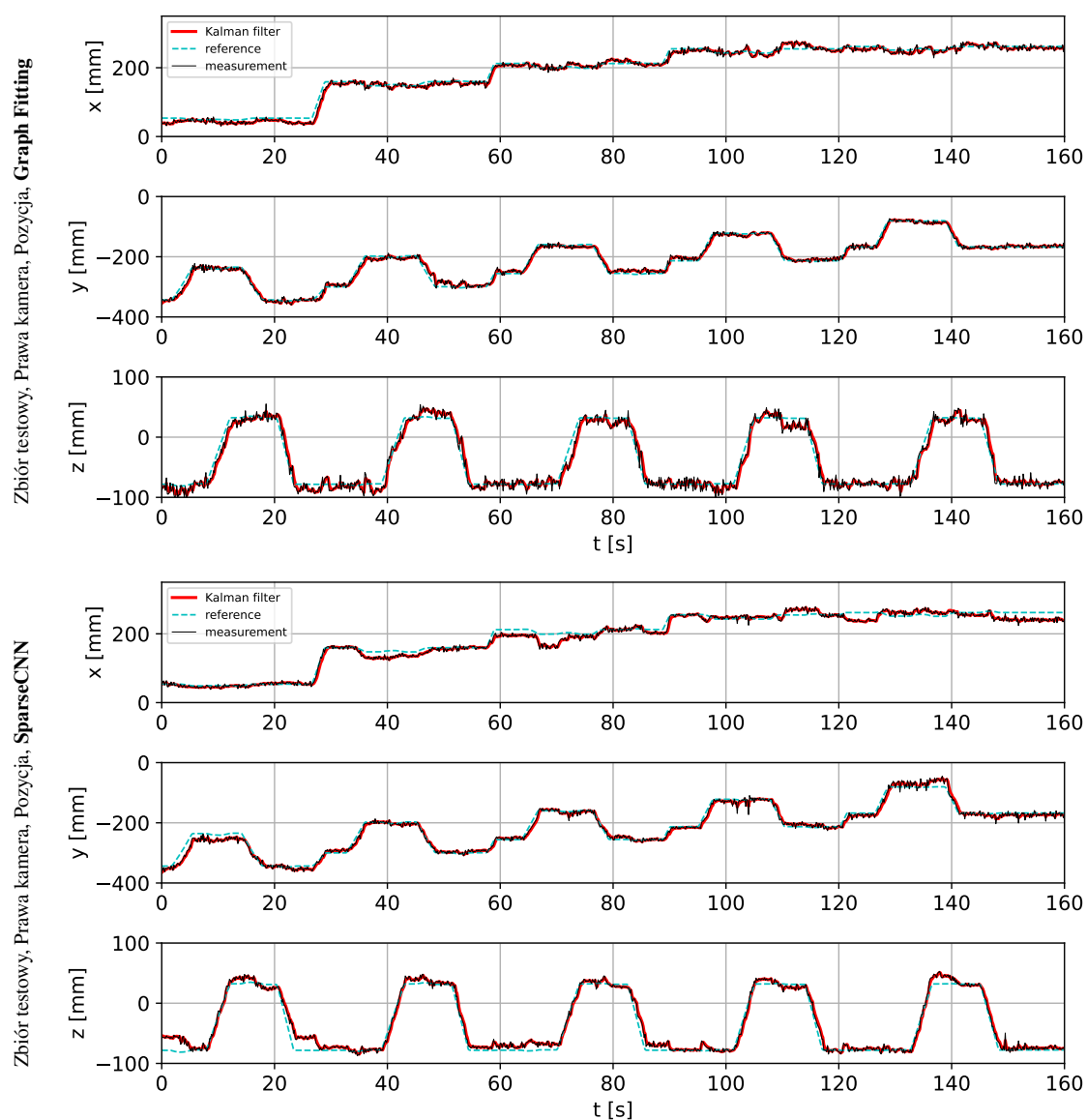


Rysunek 4.29: Porównanie jakości przewidywania kątów dla GraphFitting i SimpleCNN3D na zbiorze testowym (prawa kamera). (Źródło: własna publikacja [107])

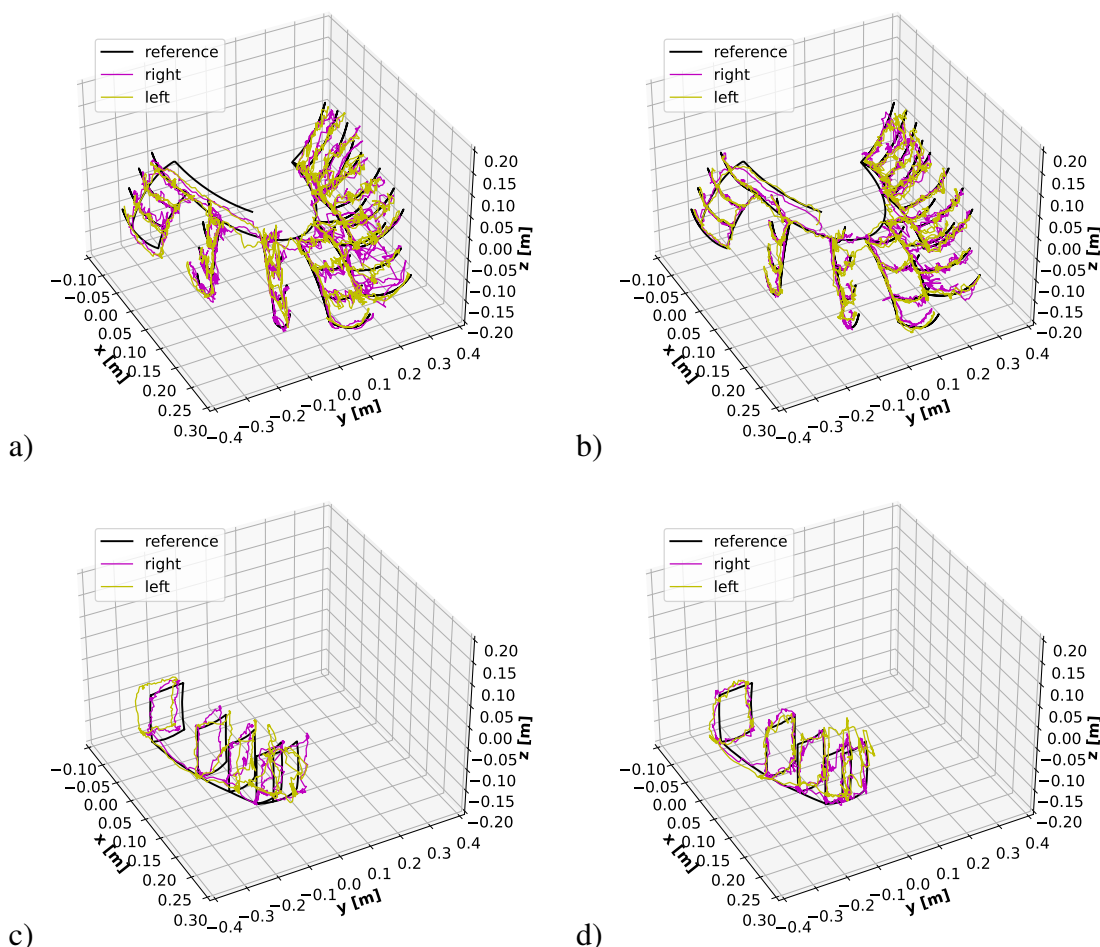
Model GraphFitting jest co do zasady podejściem nienadzorowanym i przy dokładnym usunięciu tła możemy go używać nawet wtedy, gdy nie mamy dokładnego pomiaru kątów. Pewnym ograniczeniem tej metody jest czas obliczeń dla bardziej skomplikowanych struktur grafów. Rzadka sieć splotowa jest modelem nadzorowanym i ma również pewne ograniczenia, na przykład w takich zadaniach, jak śledzenie ludzkiej ręki lub ułożenia całego ciała człowieka. W takich sytuacjach etykietowanie danych jest dodatkowym wyzwaniem.



Rysunek 4.30: Porównanie dokładności przewidywania pozycji dla GraphFitting i SimpleCNN3D na zbiorze testowym (lewa kamera). Zacielowanym obszarem oznaczono fragmenty trajektorii, w których ramię robota jest skierowane na wprost kamery. (Źródło: opracowanie własne, publikacja [107])



Rysunek 4.31: Porównanie dokładności przewidywania pozycji końcówki robota dla modeli GraphFitting i SparseCNN na zbiorze testowym (prawa kamera) . (Źródło: opracowanie własne, publikacja [107])



Rysunek 4.32: Trajektorie testowe w układzie kartezjańskim dla danych uczących i modeli: a) GraphFitting, b) SimpleCNN3D, a także dla danych testowych: c) GraphFitting, d) SimpleCNN3D Czarną linią zaznaczono trajektorię zadaną, z kolei kolorowymi predykcje na podstawie obrazu z lewej kamery (żółty) oraz prawej (fioletowy). (Źródło: opracowanie własne, publikacja [107])

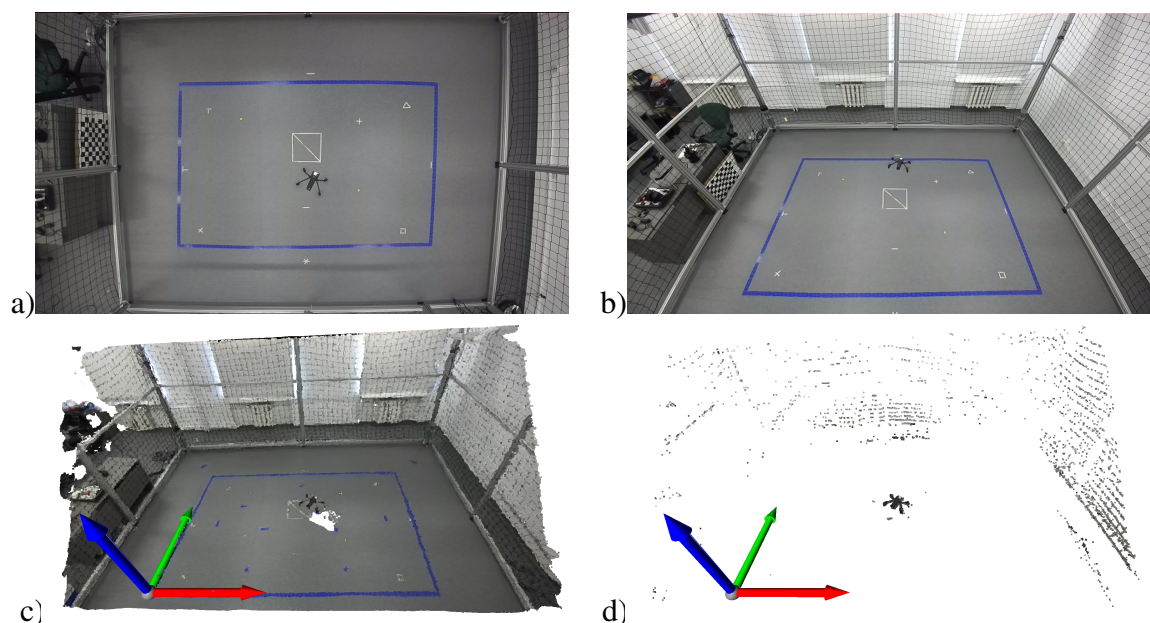
4.8 Sterowanie dronem Parrot ANAFI

4.8.1 Stanowisko badawcze

Ostatni eksperyment jest bardziej złożony niż dwa poprzednie i stanowi niejako ich połączenie. Zrealizowano w nim bowiem zarówno śledzenie obiektu z wykorzystaniem kamery głębi, jak i sterowanie z wizualnym sprzężeniem zwrotnym. Obiektem badań jest dron Parrot ANAFI – opisany dokładniej w kolejnym podrozdziale.

Przestrzenią roboczą dla drona jest klatka o wymiarach $5.4 \times 4.1 \times 3$ m (przedstawiona na rysunku 4.33), wykorzystywana we wcześniejszym eksperymencie z robotami mobilnymi. Istotnym elementem wyposażenia klatki jest siatka ochronna, której zadaniem jest przechwycenie drona w przypadku utraty kontroli nad nim.

Scena jest obserwowana przez dwie kamery głębi ZED 2i w ujęciach przedstawionych



Rysunek 4.33: Widok sceny z obu kamer: a) umieszczonej centralnie oraz b) obserwującej scenę pod kątem. Przestrzeń robocza w widoku z kamery głębi (c) oraz widok z kamery głębi po usunięciu tła (d), na rysunkach zaznaczono układ współrzędnych RGB/XYZ. (Źródło: opracowanie własne)

na rysunku 4.33a,b. Dane zbierano dla obu widoków, natomiast dalsze przetwarzanie oraz algorytmy uczenia maszynowego zrealizowano wyłącznie dla kamery obserwującej scenę pod kątem (rysunek 4.33b) – głównie z tego powodu, że z tego widoku uzyskiwany był pełniejszy obraz obserwowanej sceny.

Kamera głębi ZED 2i Użyta kamera charakteryzuje się lepszymi parametrami niż wcześniej stosowana Intel RealSense D435. Zgodnie z danymi producenta [117], kamera ZED 2i wykazuje się dokładnością głębi na poziomie $< 1\%$ dla odległości < 3 m oraz $< 5\%$ dla odległości poniżej 15 m. Podstawowe parametry przedstawiono w tabeli 4.17.

Dron Parrot ANAFI Badany dron – przedstawiony na rysunku 4.34 – jest stosunkowo niewielkim obiektem latającym. Po rozłożeniu ma wymiary $175 \times 240 \times 65$ mm oraz waży 320 g. Wyposażony jest w szereg czujników - jak informuje producent w oficjalnej dokumentacji [92] są to:

- **IMU** Invensense MPU-6000, (inercyjny układ pomiarowy zawierający: 3-osiowy żyroskop, 3-osiowy akcelerometr),
- **magnetometr** AKM AK8963,
- **GPS** U-BLOX UBX-M8030,
- **czujnik ultradźwiękowy** do pomiaru wysokości,
- **barometr** ST Microelectronics LPS22HB,

Tabela 4.17: Główne parametry techniczne kamery ZED 2i.

Parametr	Wartość
Wymiary	175.25 × 30.25 × 43.10 mm
Waga	230 g
Kąty widzenia dla głębi (FOV)	110°(H) x 70°(V) x 120°(D)
Tryby nagrywania głębi	2K@15fps, 1080p@30fps, 720p@60fps, VGA@100
Zakres głębi	0,3 m do 20 m (najlepsze obrazowanie w tym zakresie)
Dokładność pomiaru głębi	< 1% dla odległości < 3 m oraz < 5% dla odległości poniżej 15 m
Format zapisu	SVO (rejestruje timestamp dla każdej klatki)
API w językach	Python, C, C++, C#
Komunikacja	USB 3.1
Wymaga GPU	TAK (API ZED SDK)
Dodatkowe wyposażenie	IMU, barometr i magnetometr



Rysunek 4.34: Zdjęcie drona Parrot ANAFI. (Źródło: opracowanie własne)

- **kamera** umieszczona na spodzie drona, wspomagająca pomiar prędkości poziomej i wysokości,

Wbudowane oprogramowanie drona wykorzystuje odczyty z czujników do estymacji wysokości, pozycji i prędkości drona. Bezpośredni dostęp do pomiarów z większości sensorów nie jest jednak możliwy. W ramach komunikacji z dronem (za pośrednictwem biblioteki Pythona `pyparrot`) dostępny jest natomiast odczyt pozycji w oparciu o sygnał GPS oraz odczyt kątów roll, pitch i yaw, a także m. in. wysokości i prędkości drona

czy też poziomu naładowania baterii. Podkreślić należy, że w pomieszczeniu – a więc w warunkach niniejszego eksperymentu – sygnał GPS nie jest dostępny. Co za tym idzie, nie można ustalić położenia drona bez dokonywania dodatkowych pomiarów. Istotnym problemem jest też dokładność pomiaru kąta – jak podano w [36], IMU Invensense MPU-6000 wspierany sygnałem GPS w warunkach zewnętrznych, zapewnia dokładność pomiaru rzędu 3 – 10 stopni (0.05 – 0.17 radiana). Innym ograniczeniem jest częstość komunikacji z dronem – odczyty możliwe są co około 0.1 sekundy.

Badanym dronem można sterować na dwa sposoby:

- z użyciem kontrolera oraz aplikacji przeznaczonej dla smartfonów,
- za pośrednictwem API.

W związku ze znacznie większymi możliwościami wybrany został drugi sposób. Sterowanie za pomocą biblioteki `pyparrot` języka Python pozwala na dwukierunkową komunikację z dronem. Biblioteka umożliwia trzy tryby sterowania lotem:

1. Użycie metody `move_relative(x,y,z, $\alpha_z$ )`. Po wysłaniu komendy dron przemieszcza się o wektor $[x, y, -z]$ względem swojego obecnego położenia oraz obraca względem osi Z o kąt α_z . Wszystkie ruchy odbywają się jednocześnie, a rozwiązania zadania odwrotnego kinematyki i dynamiki (tj. sterowań pędnikami) jest wykonywane po stronie oprogramowania drona.
2. Użycie metody `fly_direct( $\alpha_{roll}$ ,  $\alpha_{pitch}$ ,  $\alpha_{yaw}$ ,  $z$ ,  $t$ )`, której wykonanie ustawia drona pod zadanymi kątami: roll, pitch, yaw oraz ustala zadaną prędkość lotu pionowego przez czas t , wyrażony w sekundach. Ustalając pochylenia roll i pitch wymusza się ruch w bok, w przód, lub ich złożenie. Nadmienić należy, że argumentami metody odpowiadającymi za kąty oraz prędkość lotu góra/dół są liczby całkowite z zakresu $[-100, 100]$. Liczby te odnoszą się do domyślnych maksymalnych wartości kątów oraz prędkości. Domyślne wartości można zmienić używając odpowiednich poleceń.
3. Trzeci sposób to procedura prywatna, niedostępna bezpośrednio przez API, i jest wykorzystywana przez metody powyższe do generowania ruchu drona. Działa podobnie jak `fly_direct`, z tym że czas działania komendy wynosi jeden cykl sterowania. Aby dron był w ruchu, komenda ta cały czas musi być wymuszana, a uzyskanie odpowiedniej trajektorii wymaga rozwiązania zadania odwrotnego dynamiki.

4.8.2 Przygotowanie zbioru danych

Podsumowanie problemu badawczego w zadaniu lokalizacji oraz sterowania dronem:

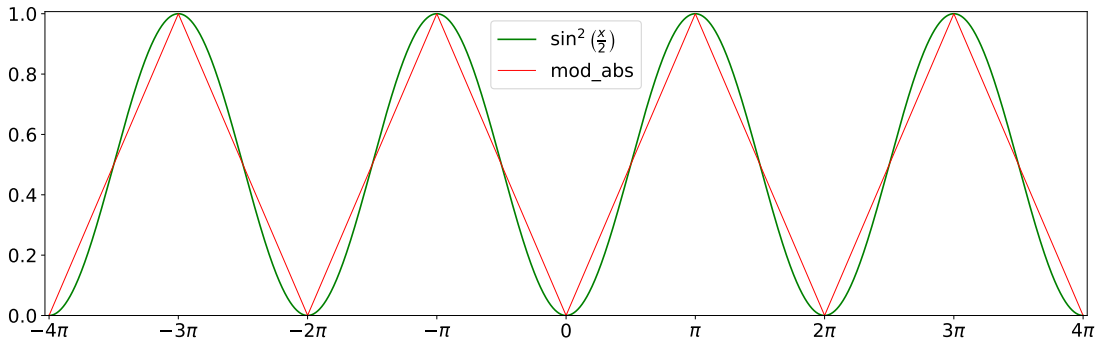
Dane wejściowe	Obraz z kamery RGB-D w formacie $1280 \times 720 \times 4$ Chmura punktów o wymiarach: $545 \times 545 \times 545 \times 3$ Liczba próbek uczących: 776 795 Liczba próbek walidacyjnych: 26 945
Zadanie główne	Lot dronem wzdłuż zadanej trajektorii.
Zadania	1. Predykcja stanu drona tj. jednoczesna predykcja pozycji i orientacji drona (x, y, z, α) . 2. Predykcja pozycji drona (x, y, z) , za pomocą sieci splotowej i detektora YOLO. 3. Predykcja kąta α_z (obrót drona w osi Z – yaw) na obrazach wyciętych i skadowanych. 4. Sterowanie dronem wykorzystując wizyjny tor pomiarowy oraz pętlę sprzężenia zwrotnego.

W ramach przygotowywania zbioru wykonano 68 rejestrowanych lotów trwających od około pół minuty do kilku minut. Podczas tych lotów sterowano dronem za pośrednictwem biblioteki pyparrot. Sterowania dobierano tak, aby trajektorie były jak bardziej różnorodne pod względem pozycji oraz kątów. Zebrane filmy przedstawiają drona latającego zarówno po zaplanowanych wcześniej sekwencjach (sterowanie metodą `move_relative`) oraz po bardziej złożonych trajektoriach (użycie metody `fly_direct`).

W trakcie każdego lotu rejestrowano odczyty udostępniane przez API drona (z częstotnością około 0.1 s) wraz z czasem uzyskania odczytu. Jednocześnie nagrywano przelot z użyciem kamer ZED 2i (60 klatek na sekundę), która oprócz informacji wizyjnych przechowuje także informacje o czasie (timestamp) zarejestrowania klatki. Wykorzystanie danych czasowych, podobnie jak w eksperymencie z ramieniem robota, pozwoliło to na wykonanie interpolacji i ujednolicenia odczytów z drona do klatek zarejestrowanych przez kamerę. Skupiono się przede wszystkim na danych istotnych z punktu widzenia proponowanej metody sterowania – a więc pozycji oraz kącie obrotu w osi Z. Obrazy rejestrowano z dwóch kamer – obserwującej scenę pod pewnym kątem oraz przedstawiającej widok z góry. Na potrzeby eksperymentu wykorzystano ostatecznie tylko dane z kamery obserwującej scenę z boku, jednak dane te po przekształceniu do chmury punktów były transformowane z układu odniesienia związanego z kamerą, do układu odniesienia związanego ze sceną (patrz rysunek 4.33).

Współrzędne x, y i z drona oraz x, y i z kamery drona ustalono za pomocą tagowania ręcznego wspomaganego wyuczonym wcześniej detektorem YOLO. Detektor został wyuczony się do znalezienia drona oraz kamery drona. Na podstawie obszaru detekcji znajdowano współrzędne centralnych punktów obszarów detekcji (ang. bounding boxów) w przestrzeni trójwymiarowej. Zastosowano tutaj podejście takie jak dla eksperymentu dotyczącego iRobotów - ręcznie oznaczono niewielką część danych, a następnie kilkakrotnie douczano model na predykcjach z naniesionymi poprawkami. Zaznaczyć należy, że tym razem liczba koniecznych korekt była większa. Wynika to z tego, że kamera drona jest koloru białego i niejednokrotnie występowała na jasnym tle – i w związku z tym oraz jej małym rozmiarem pojawiały się trudności z jej wykryciem.

Wykorzystując powyższe techniki stworzone trzy zbiory danych:



Rysunek 4.35: Wizualizacja dwóch rozważanych funkcji wykorzystywanych do wyznaczania błędów zmiennych okresowych (kątów): funkcja wartość bezwzględna modulo 2π (linia czerwona) oraz $\sin^2(\frac{x}{2})$. (Źródło: opracowanie własne)

- **Uczący** – zawierający dane z 66 filmów z różnych lotów. Zarejestrowane filmy zawierały od 2 000 do 30 000 klatek, co przekłada się na rozmiar zbioru uczącego wynoszący 776 795 różnych klatek.
- **Walidujący** – zawierający dane z dwóch oddzielnych filmów (o łącznej długości 26 945 klatek).
- **Testowy** – na zbiór ten składa się 100 ręcznie oznaczonych próbek. Adnotacje wykonano bezpośrednio na chmurach punktów.

4.8.3 Modele wykorzystywane w eksperymentach oraz przygotowanie danych

W eksperymentach wykorzystano dwa autorskie modele sieci spłotowych:

1. YOLO+DroneCropCNN3D,
2. ProjectionCNN3D oraz porównawczo gęsty ProjectionCNN3D_Dense.

Pierwszy z nich wykonuje predykcję pozycji i orientacji drona dwuetapowo. W pierwszym kroku (za pomocą modelu YOLO) wykonywana jest detekcja drona na obrazie dwuwymiarowym i na podstawie obszaru detekcji określano położenie drona. Następnie z obrazu 2D oraz obrazu głębi wycinany jest fragment obszaru sceny zawierający drona.

W oparciu o ten fragment obrazu tworzona jest chmura punktów, która po wokselizacji stanowi wejście do sieci DroneCropCNN3D. Zadaniem tej sieci jest tylko predykcja kąta obrotu drona w osi Z.

Z kolei drugi model – ProjectionCNN3D – pracuje na chmurze punktów przedstawiającej całą scenę i nie wymaga dodatkowej sieci do ustalenia położenia drona. Model ten przewiduje jednocześnie pozycję i orientację obiektu.

Na danych pobieranych z kamery RGB-D przed podaniem na wejście sieci ProjectionCNN3D wykonywano kolejno: usuwanie statycznego tła, zamianę na chmurę punktów oraz wokselizację do rozmiaru pojedynczej klatki $545 \times 545 \times 545 \times 3$, gdzie 3 oznacza trzy kanały kolorów (RGB). Dla sieci DroneCropCNN3D rozmiar wejścia wynosił $38 \times 40 \times 40 \times 3$, ponieważ obraz wejściowy przedstawiał tylko niewielki wycinek sceny.

Do treningu modeli użyto funkcji straty opartej na kryterium MSE (rysunek 4.35):

$$\sin^2\left(\frac{1}{2}(\alpha_{pred} - \alpha_{true})\right) + (xyz_{pred} - xyz_{true})^2. \quad (4.5)$$

Funkcja ta składa się z dwóch części: składnika dotyczącego kąta oraz składnika dotyczącego pozycji (błąd średniokwadratowy, składnik używany tylko dla modelu ProjectionCNN3D). Obliczanie błędu dla kąta jest realizowane z użyciem symetrycznej i okresowej funkcji $\text{mod}2\pi_sym$ (rysunek 4.35) – funkcja ta zachowuje się w podobny sposób jak MSE, a jednocześnie uwzględnia okresowość zmiennej α , tj. gdy dwa kąty różnią się o 2π , wskazania błędów powinny być jednakowe. Przykładowo, gdy jeden kąt jest zbliżony do $-\pi$ a drugi do π to tak naprawdę oznacza to ten sam kąt. Brak operacji modulo powodowałby, że w tym przypadku błąd byłby pozornie duży. W trakcie badań rozważano również okresową funkcję wartości bezwzględnej (funkcję mod_abs), której pewną wadą jest brak gładkości w otoczeniu zera.

Opracowane modele były porównane z istniejącymi architekturami sieci zaprojektowanymi do rozwiązywania zadań z danymi w postaci chmur punktów. Wykorzystano następujące modele referencyjne:

- sieć PointNet [142],
- sieć grafową DGCNN [122],
- transformer pracujący na chmurach punktów – model PCT [100].

Wszystkie wykorzystane modele były zaimplementowane we framweworku PyTorch.

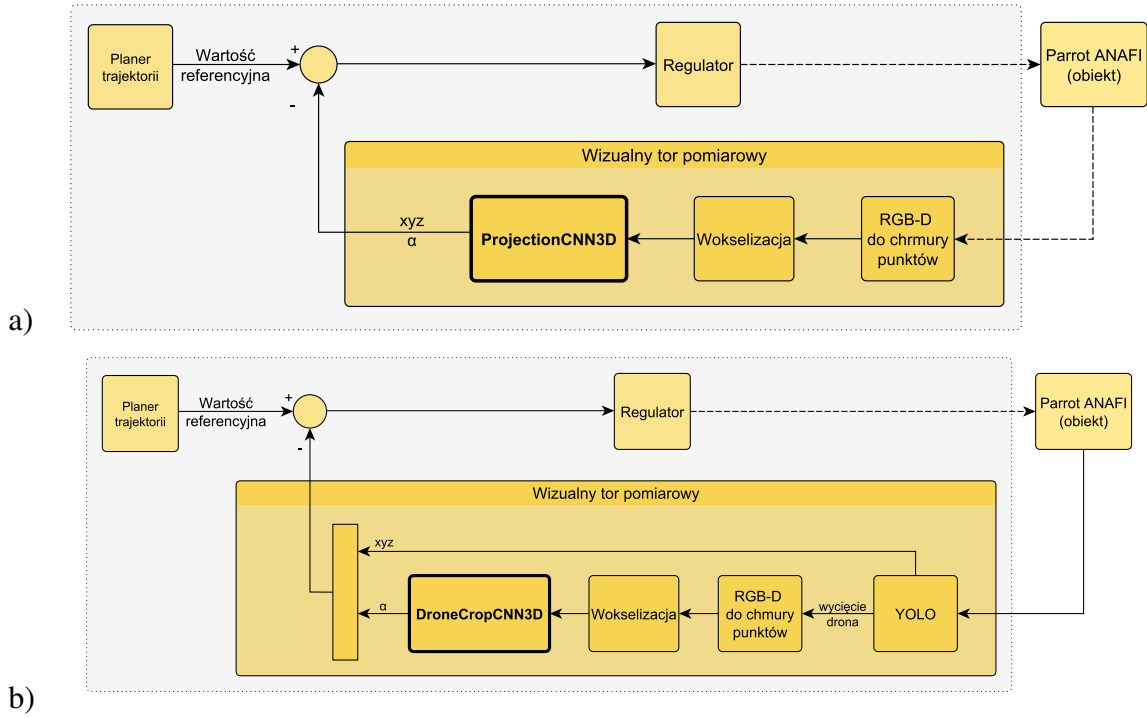
4.8.4 Pętla sterująca

Do lokalizacji drona wykorzystano dwa wspomniane wcześniej modele. Model ProjectionCNN3D wraz z pętlą sterującą przedstawiono na rysunku 4.36a. Drugi model, a w zasadzie kombinację dwóch modeli: YOLO i sieci DroneCropCNN3D (YOLO/CNN), wraz z pętlą sterującą przedstawia rysunek 4.36b. Zasadnicza różnica jest taka, że pierwszy model pracuje na pełnej scenie i jednocześnie wyznacza predykcje wszystkich zmiennych, natomiast drugi model do lokalizacji używa detektora YOLO, oraz używa dodatkowej sieci rzadkiej do predykcji kąta na przyciętych i skadrowanych obrazach 3D drona.

Planer trajektorii zadawał kolejne punkty na przygotowanej wcześniej trajektorii testowej, a celem – podobnie jak w przypadku robotów mobilnych – było nadążanie za zadaną trajektorią. Różnica pomiędzy dronem a robotem mobilnym (poza oczywistą wysokością) polegała na tym, że robot mobilny nie może poruszać się bezpośrednio w bok, natomiast dron może wykonać ruch o wektor w dowolnym kierunku niezależnie od obrotu.

Do sterowania dronem za pośrednictwem API, wybrano metodę `move_relative`, ponieważ oferowała najwygodniejszy sposób sterowania z zadaniem ruchu wzdłuż zadanej trajektorii. Celem przedstawienia reguł sterujących ruchem drona przyjmijmy następujące oznaczenia: niech $[x, y, z, \alpha]$ będą współrzędnymi określającymi aktualny stan drona (tj. aktualne położenie drona i kąt obrotu wokół osi Z), niech wektor $[x_0, y_0, z_0, \alpha_0]$ oznacza zadany stan docelowy. Oba wektory są zdefiniowane w globalnym układzie współrzędnych. Wówczas różnica pomiędzy aktualnym położeniem drona, a stanem docelowym (błąd) wyrażona jest jako:

$$\begin{aligned}\Delta_x &= x_0 - x \\ \Delta_y &= y_0 - y \\ \Delta_z &= z_0 - z \\ \Delta_\alpha &= \alpha_{z_0} - \alpha_z.\end{aligned}\tag{4.6}$$



Rysunek 4.36: Schemat układu sterowania dronem Parrot ANAFI: a) użycie sieci ProjectionCNN3D oraz b) metody referencyjnej YOLO/CNN3D. (Źródło: opracowanie własne)

W oparciu o błąd, regulator wyznacza wektor sterujący dla drona zgodnie z następującą regułą:

$$\begin{aligned}
 c_x &= \Delta_x \cos(\Delta_\alpha) + \Delta_y \sin(\Delta_\alpha) \\
 c_y &= \Delta_x \sin(\Delta_\alpha) - \Delta_y \cos(\Delta_\alpha) \\
 c_z &= -\Delta_z \\
 c_\alpha &= -\text{modulo}2\pi(\Delta_\alpha)
 \end{aligned}
 \tag{4.7}$$

Ponieważ dron kontrolowany jest metodą `move_relative`, do współrzędnych x oraz y konieczne jest wyznaczenie względnej zmiany pozycji względem lokalnego układu współrzędnych (względem drona). Z kolei znak minus w równaniu wyznaczającym c_z oraz c_α spowodowany jest tym, że układ drona posiada oś Z skierowaną przeciwnie do osi przyjętego nieruchomego układu, tj. oś Z w układzie drona skierowana jest w dół, natomiast oś Z globalnego układu skierowana jest w górę. Skutkuje to też tym, że obrót wokół osi Z w obu układach realizowany jest w przeciwną stronę.

Pomiary dostarczane przez modele zostały dodatkowo wygładzone za pomocą filtru Kalmana. Wykorzystano do tego celu bibliotekę `filterpy` [57, 65].

Konieczny jest dodatkowy komentarz dotyczący wyznaczania wartości zadanej dla kąta oraz obliczenia błędów kątowych. Formalnie i praktycznie możliwe jest ustalenie wartości zadanej dla kąta na wartość wykraczającą poza pełen obrót, np. $\alpha_z = 4\pi$. Przykładowo, gdy dron jest w pozycji $\alpha_0 = 0$, taka wartość oznaczałaby konieczność wykonania dwóch pełnych obrotów. Natomiast istnieje trudność w wyznaczaniu wartości

kąta z dokładnością do liczby obrotów na podstawie pojedynczej klatki, gdyż wszystkie obroty o wielokrotność obrotu pełnego wyglądają bardzo podobnie. Z tego powodu w eksperymencie zaniebujemy liczbę pełnych obrotów, a kąt mierzymy jedynie w zakresie pełnego obrotu (tj. modulo 2π , oraz mierzony w zakresie $(-\pi, \pi]$). Taki sposób powoduje powstanie pozornej nieciągłości w otoczeniu kąta o wartościach zbliżonych do π .

Bezpośrednie zastosowanie uchybu regulacji jako różnicy wartości zadanej i mierzonej jest problematyczne dla sygnałów okresowych, jakim jest kąt obrotu. Podobnie jak przy dopasowywaniu modelu CNN lub estymacji filtru Kalmana wprowadzałoby to fałszywe błędy. Z tego powodu przy pomiarze błędu dla kąta wprowadzono następującą okresową funkcję `mod_abs`:

$$\text{mod_abs}(x, T = 2\pi) = \min(x \pmod{T}, T - (x \pmod{T})). \quad (4.8)$$

Z podobnych powodów do sterowania używamy symetrycznej funkcji modulo 2π , która przyjmuje wartości w przedziale $(-\pi, \pi]$.

Podobnie, aby filtr Kalmana działał prawidłowo dla sygnału okresowego, w bibliotece `filterpy` [57, 65] konieczne było (we fragmencie do estymacji stanu kąta) dodanie jako opcji poprawki tak, aby prawidłowo obsługiwać pozorną nieciągłość przy zmianie kąta z $-\pi$ do π (lub na odwrót). Brak tej poprawki powoduje, że niewielka zmiana kąta w na granicy $-\pi$ i π powodowałaby duży błąd. Prawidłowe działanie filtra dla sygnałów okresowych osiągnięto dzięki zastosowaniu powyższej okresowej funkcji modulo 2π .

4.8.5 Wyniki

Model YOLO W pierwszej kolejności podjęto się wyboru wariantu modelu YOLO, który będzie stosowany w architekturze YOLO + DroneCropCNN3D. W wariantcie tym model YOLO uczony był do detekcji drona na obrazie RGB, a następnie na podstawie obszaru wskazanego przez detektor wycinano fragment sceny zawierający drona. Samo wycięcie chmury punktów odbywało się za pomocą API biblioteki `pyzed`. W tabeli 4.18 przedstawiono wyniki (czasy predykcji oraz dokładności) różnych struktur detektora YOLO w zadaniu predykcji pozycji drona. Na podstawie wyników przedstawionych w tabeli wybrano najbardziej złożony model YOLOv5l, liczący 46.5 miliona parametrów.

Porównanie modeli na zbiorze walidującym Następnie działanie modeli `ProjectionCNN3D` oraz `YOLO+DroneCropCNN3D` zostało sprawdzone na dwóch zbiorach: walidującym i testowym. Wyniki dla zbioru walidującego przedstawiono w tabeli 4.19. Zbiór walidujący obejmował dane z dwóch przelotów niezależnych od zbioru uczącego. Wartości referencyjne dla kąta odczytywane były bezpośrednio z drona poprzez API biblioteki `pyparrot`, natomiast wartości referencyjne dla pozycji były odczytywane półautomatycznie z wspomaganie modelu YOLO. Zbiór testowy zawierał ręcznie etykietowane pozycje dla 100 losowych klatek ze zbioru walidującego.

Na rysunku 4.37 przedstawiono porównanie pomiaru pozycji i kąta dla obu modeli. W przypadku pomiaru pozycji wyniki te nie przedstawiają tak naprawdę porównania względem obu modeli. Jak można zaobserwować, rozbieżność pomiędzy modelem `ProjectionCNN3D`, a pomiarem pozycji z użyciem procedury `YOLO+DroneCropCNN3D` jest niewielka – około 1 cm, tj. na poziomie rozmiaru woksela. Błąd działania modeli można

Tabela 4.18: Porównanie czasów oraz jakości predykcji dla różnych wariantów modelu YOLOv5 na zbiorze walidującym, dla obrazów w rozdzielczości 1280×720 pikseli. @.5 oznacza mAP@.5, z kolei @.5:.95 oznacza mAP@.5:.95. (Źródło: opracowanie własne)

Model	Czas [ms] predykcji	Klasa: dron				Klasa: kamera drona			
		P	R	@.5	@.5:.95	P	R	@.5	@.5:.95
L	32.32	1	1	0.995	0.995	0.998	0.989	0.995	0.884
M	23.38	1	1	0.995	0.993	0.998	0.987	0.995	0.869
S	19.73	1	1	0.995	0.992	0.998	0.987	0.995	0.872
N	19.65	1	1	0.995	0.98	0.995	0.978	0.994	0.861

Tabela 4.19: Porównanie modeli ProjectionCNN3D i YOLO+DroneCropCNN3D w predykcji pozycji drona oraz kamery wbudowanej w drona na zbiorze walidującym. Dokładność dla kąta liczona jest w oparciu o pomiary uzyskane przez API drona, natomiast dla pozycji w odniesieniu do wyjścia z modelu YOLO.

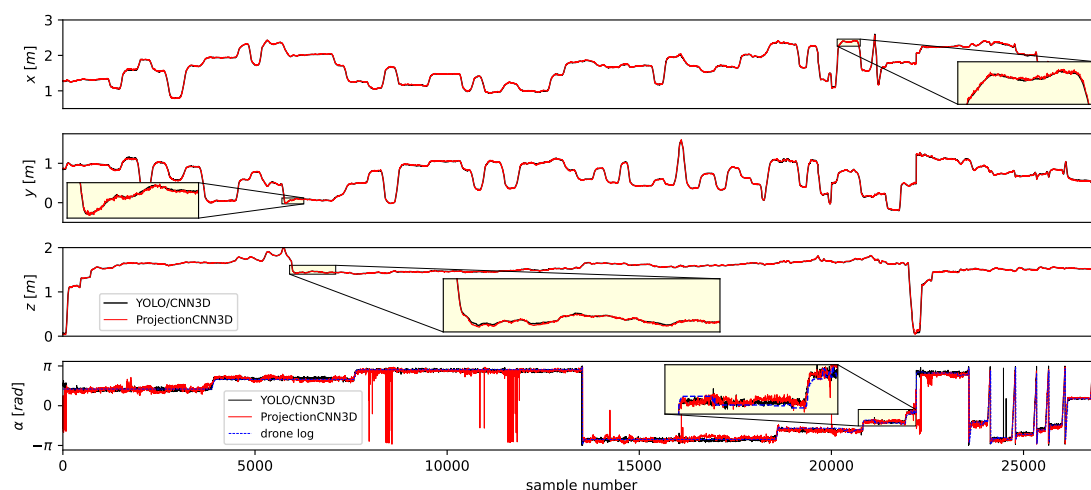
Model – obiekt	x [cm]	y [cm]	z [cm]	α_z [rad]
ProjectionCNN3D – dron	0.52	0.60	0.43	0.14
ProjectionCNN3D – kamera	0.92	1.36	1.08	-
YOLO + DroneCropCNN3D – dron	-	-	-	0.09

natomiast wyznaczyć dla kąta α_z . W przypadku pomiaru kąta obrotu (α) model pracujący na wyciętych i skadowanych danych jest nieznacznie dokładniejszy (0.09 rad względem 0.14 rad.). Należy przy tym pamiętać, że wartością referencyjną dla kąta jest odczyt z sensora drona. Odczyt ten jest obarczony szumem spowodowanym zakłóceniami magnetometru w pomieszczeniu oraz jego ograniczoną dokładnością pomiaru. Zadanie predykcji kąta lepiej rozwiązuje architektura YOLO+DroneCropCNN3D, w której jedna z sieci jest przeznaczona wyłącznie do tego problemu.

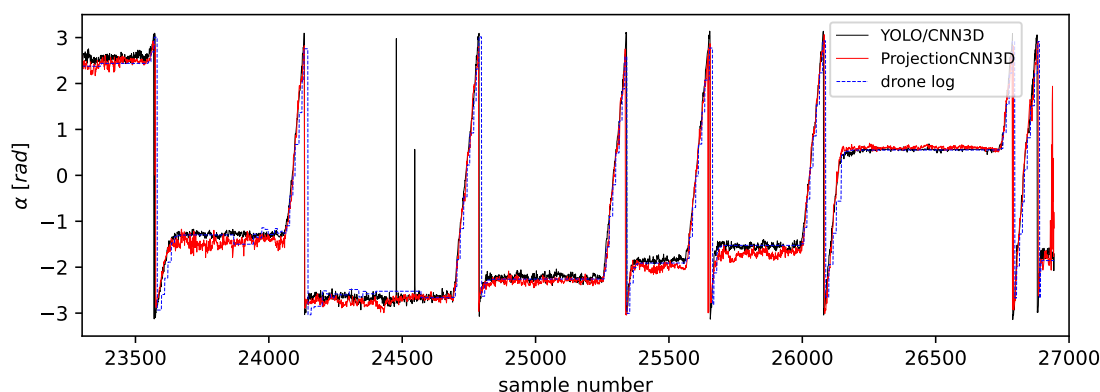
Na rysunku 4.38 przedstawiono fragment trajektorii walidującej, w którym dron wykonuje obroty w osi Z w różnych miejscach sceny. Widoczny w okolicach klatek numer 7 000 i 12 000 spadek jakości predykcji jest tylko pozorny. Spowodowane jest to tym, że mierzony kąt ma wartość bliską π oraz równocześnie są to wartości bliskie $-\pi$ – jest to wskazane wcześniej zjawisko pozornej nieciągłości.

Istotną zaletą modelu ProjectionCNN3D jest jego ujednolicona struktura do przewidywania stanu drona, umożliwiającą pominięcie detektora YOLO. Ta pojedyncza struktura sieci spłotowej skutkuje krótszymi czasami przewidywania.

Porównując dwa zadania przewidywania położenia drona i jego kąta obrotu, zadanie rozpoznawania kąta okazuje się być trudniejsze. Lokalizacja wymaga jedynie filtrowania szumów i można ją skutecznie osiągnąć z dużą dokładnością, używając chmur punktów. W ogólności kąt obrotu obiektu powinien być niezmienniczy względem translacji. Natomiast chmura punktów zarejestrowana przez kamery głębi zależy od położenia obiektu względem kamery oraz również od kąta, pod którym kamera patrzy na scenę. Tak więc widok z kamery głębi nie jest w pełni niezmienny względem translacji. Przykładowo dostajemy różny obraz obiektu przy takim samym kącie obrotu, ale umiejscowionego w



Rysunek 4.37: Trajektoria walidująca w rozbiciu na poszczególne współrzędne, liniami **czerwonymi** oznaczono wyjście z modelu ProjectionCNN3D, czarnymi liniami oznaczono wartości referencyjne z modelu YOLO. Ostatni wykres przedstawia porównanie predykcji kąta modelu ProjectionCNN3D oraz DroneCropCNN3D względem odczytów z drona (linia **niebieska**). (Źródło: opracowanie własne: publikacja [108])



Rysunek 4.38: Porównanie jakości predykcji kąta – powiększono część trajektorii, w trakcie której dron wykonywał obroty. (Źródło: opracowanie własne: publikacja [108])

lewej i w prawej części sceny. Sieć DroneCropCNN3D, wchodząca w skład metody YOLO+DroneCropCNN3D, działa z przyciętymi i wyrównanymi obrazami drona i rozwiązuje łatwiejsze zadanie niż sieć ProjectionCNN3D, przewidując kąt obrotu drona na podstawie skadrowanego i wyrównanego fragmentu sceny.

Przed rozpoczęciem eksperymentów ze sterowaniem zbadano dodatkowo dokładność przewidywania pozycji na zbiorze testowym, którego próbki zostały oznaczone manualnie, w oparciu o chmury punktów. Jest to metoda znacznie dokładniejsza niż stosowana w zbiorze uczącym i walidacyjnym (oznaczenia tworzone przez YOLO na podstawie obrazów dwuwymiarowych). Wyniki przedstawiono w tabeli 4.20. Widoczne jest to, że dla osi *Y* oraz *Z* architektura ProjectionCNN3D popełnia znacznie mniejszy błąd predykcji, a dla osi *X* nieznacznie większy.

Powyższe wyniki pokazują, że przy użyciu architektury opartej o Submanifold Sparse Convolution, przewidującej jednocześnie pozycję i kąt obrotu, można uzyskać dokład-

Tabela 4.20: Porównanie dokładności modeli ProjectionCNN3D oraz YOLO + DroneCropCNN3D na ręcznie etykietowanym zbiorze testowym.

Model	x [cm]	y [cm]	z [cm]
ProjectionCNN3D – dron	1.16	1.97	0.92
ProjectionCNN3D – kamera	1.05	1.19	0.94
YOLO + DroneCropCNN3D – dron	0.96	3.10	1.02
YOLO + DroneCropCNN3D – kamera	0.94	1.24	1.27

ność predykcji wyższą niż w przypadku referencyjnego rozwiązania. Dla przewidywania pozycji możliwe jest osiągnięcie dokładności na poziomie woksela.

Analiza czasów działania i porównanie z modelem gęstym Badane podejścia ocenione zostały również pod kątem czasu predykcji dla pojedynczej próbki. Porównanie modeli przedstawiono w Tabeli 4.21. W tabeli zamieszczono również czas przetwarzania dla modelu ProjectionCNN3D dla danych bez usuwania tła oraz dla odpowiednika tego modelu, który pracuje na danych w formie gęstej i stosuje gęste sploty.

Model ProjectionCNN3D, pracujący na chmurach punktów po usunięciu tła, pozwala na przewidywanie pozycji i orientacji drona ze zdecydowanie najwyższą częstotliwością wynoszącą 18 Hz. Jest to możliwe dzięki dostosowaniu operacji splotu do danych rzadkich, znacznej redukcji liczebności punktów, a także przewidywania stanu robota z użyciem jednego modelu. Metoda referencyjna, wymagająca YOLO, umożliwia wykonywanie pomiarów z częstotliwością ponad 12 Hz. Znacznie dłuższy czas predykcji spowodowany jest użyciem detektora. Co prawda pozwala to na przyspieszenie usuwania tła, generowania chmury punktów oraz predykcji kąta, ale zysk nie jest wystarczający, by skompensować długi czas działania modelu YOLO.

Kolejne analizowane rozwiązanie – sieć ProjectionCNN3D dla pełnej chmury punktów – eliminuje niezbyt kosztowny czasowo proces usuwania tła. Skutkuje to jedną ponad dwukrotnym wydłużeniem generowania chmury punktów i predykcji, co przekłada się na częstotliwość pomiaru około 8 Hz. Nadal jednak jest to znacznie lepszy rezultat niż dla modelu gęstego. Uzyskana z jego użyciem częstotliwość pomiaru wynosząca niespełna 2 Hz nie pozwala na kontrolowania mogącego szybko się przemieszczać drona.

Porównanie względem modeli referencyjnych Opracowane modele zostały porównane z istniejącymi architekturami sieci zaprojektowanymi do rozwiązywania zadań z danymi w postaci chmur punktów. W eksperymentach porównawczych wykorzystano modele referencyjne: Sieć PointNet [142] (PointNet), Sieć grafową Dynamic Graph CNN [122] (DGCNN), transformer pracujący na chmurach punktów [100] (PCT).

Architektury referencyjne wybrano głównie dlatego, że: są one dostosowane do danych rzadkich, są szeroko cytowane w literaturze oraz posiadały implementacje w pakiecie PyTorch dostępne do celów badawczych. Wskazane w porównaniu sieci PointNet, DGCNN oraz PCT to struktury działające bezpośrednio na chmurze punktów. Jest to zasadni-

Tabela 4.21: Czasy przetwarzania dla badanych modeli z podziałem na poszczególne operacje (wyrażony w milisekundach).

Operacja	Projection CNN3D_Dense	Projection CNN3D (bez us. tła)	Projection CNN3D	YOLO/ CNN3D
Pobranie klatki	4.77	4.76	4.75	4.75
Predykcja YOLO	—	—	—	35.42
Usuwanie tła	3.93	—	3.95	2.42
Chmura punktów	19.87	47.82	19.22	17.65
Wokselizacja	1.03	6.13	0.92	0.64
Predykcja	576.82	67.90	26.58	18.64
Czas łączny	606.42	126.61	55.42	79.52

Tabela 4.22: Porównanie różnych metod pracujących na chmurach punktów na zbiorze walidującym względem predykcji modelu YOLO. Podano błędy dla poszczególnych zmiennych oraz łączny błąd pozycji $err_r = \sqrt{x^2 + y^2 + z^2}$.

Model	Średni błąd predykcji				
	x [cm]	y [cm]	z [cm]	err_r [cm]	α_z [rad]
ProjectionCNN3D	0.52	0.60	0.43	0.91	0.141
PointNet	6.12	6.35	3.08	9.34	0.337
DGCNN ($k = 40$)	3.48	3.28	2.16	5.24	0.114
DGCNN ($k = 20$)	4.49	2.65	2.53	5.80	0.103
DGCNN ($k = 10$)	4.02	3.51	1.97	5.69	0.104
PCT	2.05	1.73	1.27	2.97	0.10

Tabela 4.23: Porównanie różnych metod pracujących na chmurach punktów na ręcznie etykietowanym zbiorze testowym.

Model	Błąd predykcji			
	x [cm]	y [cm]	z [cm]	err_{xyz} [cm]
ProjectionCNN3D	1.16	1.97	0.92	2.46
PointNet	5.89	7.97	2.48	10.21
DGCNN ($k = 40$)	4.42	4.17	2.05	6.41
DGCNN ($k = 20$)	6.11	4.44	2.64	8.00
DGCNN ($k = 10$)	5.03	4.41	2.35	7.09
PCT	2.57	2.58	1.45	3.92

cza różnica w stosunku do proponowanych struktur, które wymagają dodatkowego kroku (wokselizacji), mimo że również pracują na danych rzadkich. Badane modele nauczono na takim samym zbiorze uczącym i z użyciem tego samego kryterium jak w przypadku mo-

delu ProjectionCNN3D, a następnie przetestowano na tym samym zbiorze walidacyjnym oraz zbiorze testowym.

Porównanie trajektorii na zbiorze walidacyjnym dla wszystkich analizowanych modeli przedstawiono na rysunku 4.39, natomiast podsumowanie ilościowe tych wyników zebrano w tabeli 4.22. Analogiczne wyniki na ręcznie etykietowanym zbiorze testowym przedstawiono w tabeli 4.23. Jak można zaobserwować, najlepszym z analizowanych modeli predykcyjnych był model PCT (Point Cloud Transformer). Uzyskał on około dwukrotnie gorszą dokładność przy pomiarze pozycji oraz nieco lepsza dokładność przy pomiarze kąta (0.1 rad względem 0.14). Najgorszym modelem okazała się sieć PointNet, przy czym pewnym usprawiedliwieniem może być tu fakt, że była ona pierwszą architekturą zdolną do pracy bezpośrednio na chmurze punktów.

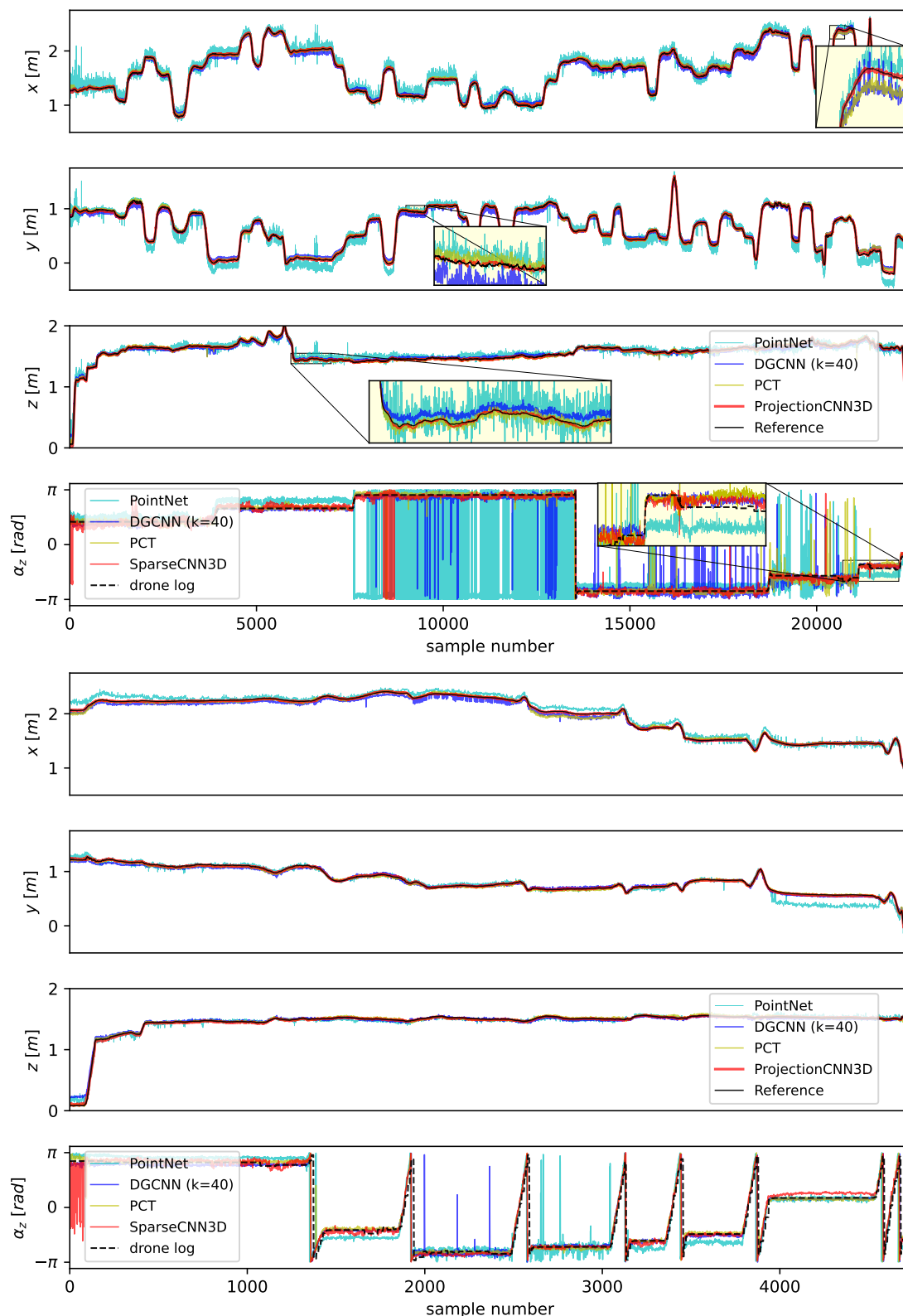
Eksperymenty te pokazały również, że zaproponowana architektura ProjectionCNN3D pozwala na osiągnięcie wyraźnie wyższej jakości predykcji pozycji drona, mimo że pracuje na danych przedstawiających scenę z niższą dokładnością niż oryginalna chmura punktów (ze względu na proces wokselizacji).

W tabelach 4.22 i 4.23 umieszczono wyniki sieci grafowej DGCNN dla różnych wartości parametru k (liczba centrów w algorytmie k środków). Jak widać uzyskiwana jest zbliżona jakość predykcji. Dodatkowo na rysunku 4.40 przedstawiono porównanie trajektorii dla modeli DGCNN, przy różnych wartościach parametru k , na wybranym fragmencie zbioru walidującego.

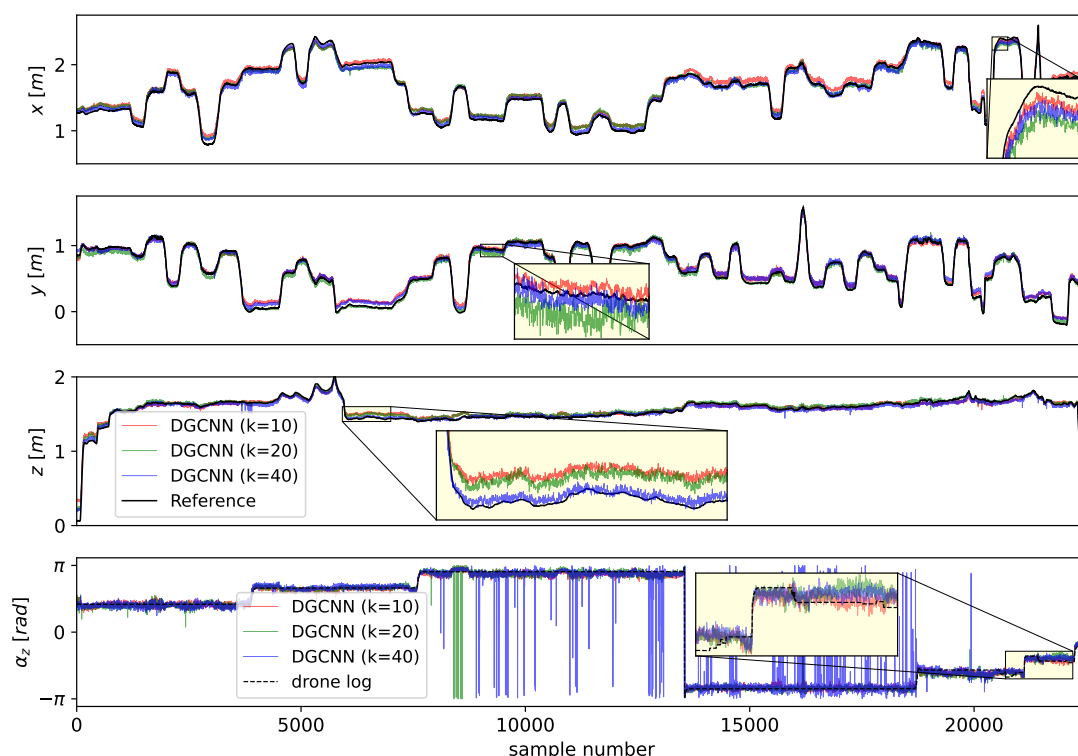
Ocena jakości wizualnego sprzężenia zwrotnego Po gruntownym przetestowaniu modeli wykonano eksperyment obejmujący analizę całego układu regulacji wyposażonego w wizualny tor pomiarowy. Eksperyment polegał na przelocie drona po zadanej trajektorii z użyciem wcześniej omówionego toru pomiarowego. Sterowanie wzdłuż trajektorii odbywało się za pomocą API pyparot, zgodnie z zaprezentowanym algorytmem regulacji.

Jako trasę testową wybrano trajektorię w kształcie ósemki, wykonywanej dwukrotnie na wysokości 1.2 oraz 1.6 metra. W trakcie lotu kamera drona miała być zawsze skierowana do środka trajektorii. W pętli sterującej pomiary były wygładzane filtrem Kalmana. Pomiary wykonywano z częstością przetwarzania modeli, natomiast sygnały sterujące generowane są w interwałach ok. 1 – 2 sekundowych. W chwili, gdy dron osiągnie jeden z punktów zadanych trajektorii, podawany jest kolejny punkt zadany.

Na rysunku 4.41a,b przedstawiono trajektorię zadaną oraz trajektorie uzyskane dla obu modeli: ProjectionCNN3D oraz YOLO + DroneCropCNN3D. Dodatkowo na platformie Zenodo udostępniono filmy przedstawiające oba przeloty [105]. Z kolei na rysunku 4.41c,d przedstawiono te same trajektorie w rozbiciu na poszczególne zmienne stanu oraz dodatkowo przedstawiono sygnały przed i po użyciu filtru Kalmana. Dla wysokości oraz kąta zaznaczono również odczyty pochodzące z API drona. Widoczne jest kilkucentymetrowe przesunięcie pomiędzy pozycją mierzona przez API oraz pozycją mierzoną przez proponowany model. Spowodowane jest to prawdopodobnie tym, że wbudowany sensor jest skalibrowany do poziomu podłogi, natomiast model przewiduje współrzędną w pobliżu środka ciężkości drona. rysunek 4.41e,f przedstawia nałożone klatki z przelotami w ujęciu z kamery wraz z zaznaczoną trajektorią.



Rysunek 4.39: Porównanie różnych sieci dostosowanych do pracy z chmurami punktów: PointNet, DGCNN, PCT oraz ProjectionCNN3D względem sygnałów referencyjnych. Górny i dolny wykres przedstawiają dwie różne części zbioru walidującego. (Źródło: opracowanie własne)



Rysunek 4.40: Porównanie wariantów modelu sieci grafowej DGCNN dla różnych wartości parametru k , wyniki na II części zbioru walidującego. (Źródło: opracowanie własne)

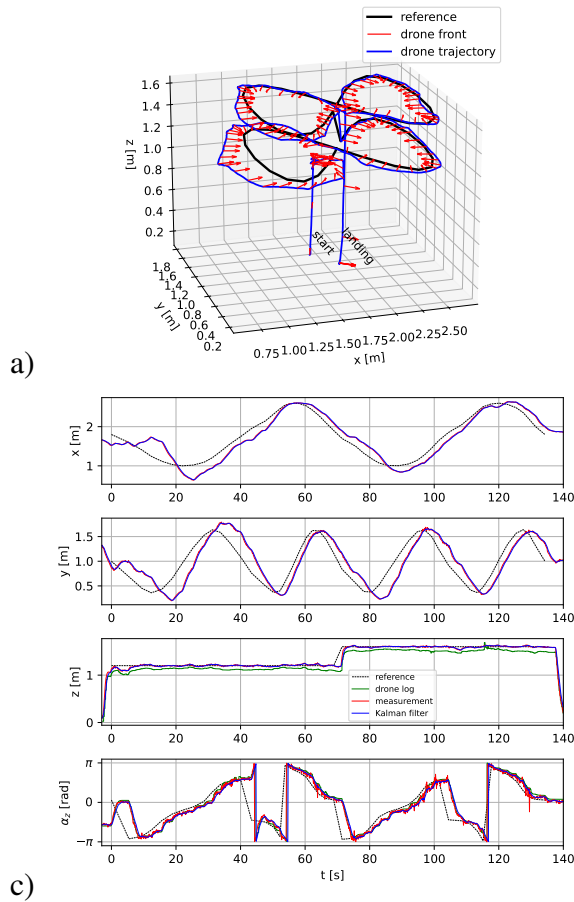
Tabela 4.24: Dokładność odwzorowania trajektorii przez drona z użyciem modeli YOLO + DroneCropCNN3D i ProjectionCNN3D, z filtrem Kalmana oraz bez filtra.

Model	x [cm]	y [cm]	z [cm]	α_z [rad]
YOLO+DroneCropCNN3D	8.14	7.06	1.49	0.14
YOLO+DroneCropCNN3D/Kalman	8.11	6.67	1.54	0.10
ProjectionCNN3D	4.86	5.39	1.46	0.30
ProjectionCNN3D/Kalman	5.59	3.89	1.38	0.18

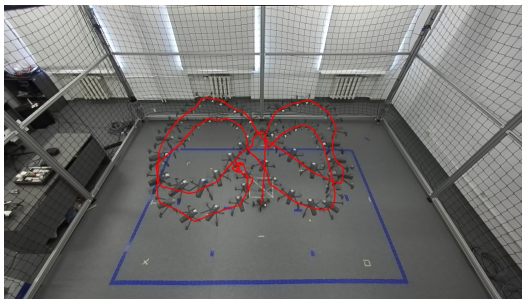
Eksperyment podsumowany został w tabeli 4.24. Widoczna jest znacznie lepsza dokładność pozycji dla sieci pracującej na całym obrazie sceny (model ProjectionCNN3D). Architektura YOLO + DroneCropCNN3D wyróżniała się tylko w zadaniu predykcji kąta. Wynika to głównie z tego, że posiadała ona sieć dedykowaną do predykcji kąta, która pracuje na niewielkim wycinku chmury punktów zawierającym drona. Architektura ProjectionCNN3D przyjmująca na wejście całą scenę i rozwiązuje jednocześnie oba zadania (predykcję pozycji i predykcję kąta) realizowane osobno przez obie części drugiego modelu, tj. YOLO oraz DroneCropCNN3D. Ponadto należy podkreślić, że filtr Kalmana w obu przypadkach poprawia jakość predykcji kąta obrotu, natomiast w przypadku pomiaru pozycji poprawa następowała w większości przypadków, chociaż nie zawsze.

Wcześniejsze etapy eksperymentu wyznaczały dokładność samego toru pomiarowego, natomiast w tym etapie badana jest pod uwagę dokładność całego układu regulacji. Na

model YOLO + DroneCropCNN3D

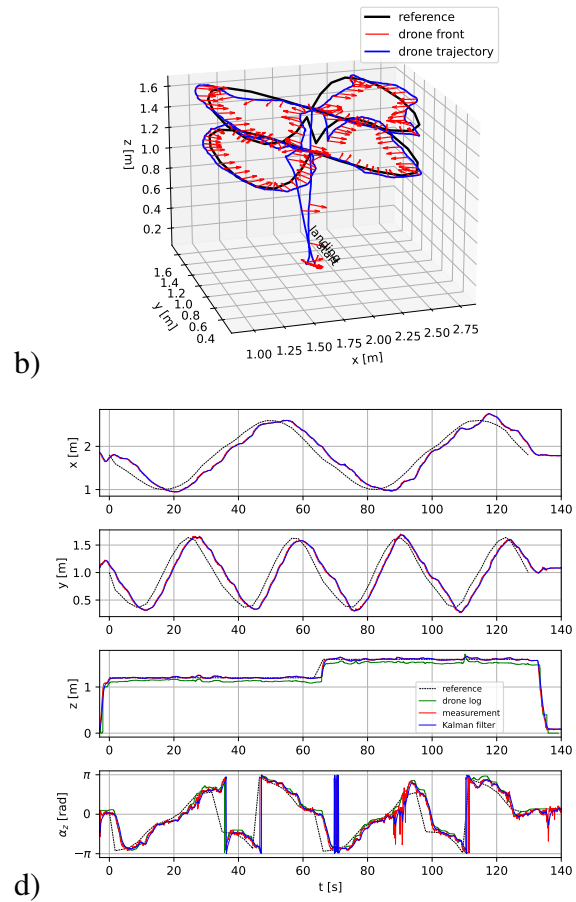


c)

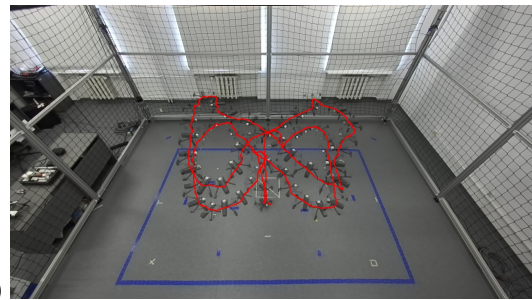


e)

model ProjectionCNN3D



d)



f)

Rysunek 4.41: a, b) Trajektorie w przestrzeni 3D uzyskane podczas lotów testowych (niebieskie linie) wraz z informacją o obrocie drona (czerwone strzałki) oraz zadanymi trajektoriami (czarne linie). c, d) Trajektorie dla z lotów testowych w rozbiciu na poszczególne współrzędne: x , y , z i α_z . e, f) Trajektorie podczas lotów testowych w rzucie widoku kamery RGB-D. Lewa kolumna: model YOLO + DroneCropCNN3D, prawa: model ProjectionCNN3D pracujący na całej scenie. (Źródło: opracowanie własne, publikacja [108])

dokładność układu regulacji, oprócz toru pomiarowego, składają się również jakość algorytmu regulacji, dokładności elementów wykonawczych czy obecność zakłóceń. Należy podkreślić, że głównym źródłem znacznej różnicy w dokładności pomiędzy wcześniejszymi etapami eksperymentu, a obecnym, jest ograniczona dokładność układów wykonaw-

czych i kontrolera drona. Ponadto istotny jest tu też wpływ zakłóceń jak ruchy powietrza, czy np. bliskość siatki.

Celem oszacowania dokładności pozycjonowania kontrolera wbudowanego w drona przeprowadzono dodatkowe krótkie doświadczenie, w którym wydano dronowi polecenie wzniesienia się i pozostawania nieruchomo przez minutę. W takim eksperymencie dron jest utrzymywany w jednej pozycji przez wbudowany kontroler wykorzystujący informacje z wbudowanych sensorów (głównie IMU i czujnik ultradźwiękowy, oraz wspomaganie kamerą). W eksperymencie tym rejestrowano pozycje za pomocą układu wizyjnego. W takich warunkach dron wykonywał ruchy w zakresie 4–6 cm (do 6.06 cm w osi X , do 5.06 cm w osi Y i do 4.10 cm w osi Z). Przy dłuższym czasie trwania eksperymentu zaobserwowano zmiany pozycji drona rzędu 10 cm i więcej.

Podsumowanie wyników Zaproponowany model wykorzystujący dane z całej sceny był w stanie skutecznie zmierzyć położenie i orientację drona. Jak pokazały badania, dokładność pomiaru położenia drona za pomocą proponowanej procedury wizualnego sprzężenia zwrotnego jest stosunkowo wysoka, rzędu 1 cm (i mniej), co odpowiada rozmiarowi wokseli. Dokładność przewidywania orientacji drona mieści się w zakresie do 0.14 rad. Czas przetwarzania dla całego modelu sceny wynosił około 18.5 FPS co było wystarczające i dostosowane do interwału próbkowania API drona. Dokładność całej pętli sterującej, tj. dokładność, z jaką dron odwzorowuje zadaną trajektorię, okazała się porównywalna z dokładnością, z jaką wbudowany kontroler drona utrzymuje go w ustalonej pozycji tj. około 4 cm dla pozycji oraz 0.18 radiana dla kąta. Proponowana sieć ProjectionCNN3D, wykorzystująca trójwymiarowe dane wejściowe o całej scenie, uzyskała nieco lepszą dokładność przewidywania położenia niż predyktor lokalizacji oparty na YOLO.

Model YOLO przewidywał położenie (środką) drona i kamery i na tej podstawie podjęto próbę przewidywania kąta obrotu, korzystając z zależności geometrycznych, jednak uzyskane wyniki okazały się gorsze niż w przypadku modelu ProjectionCNN3D. Zastosowane projekcje do podprzestrzeni 2D i 1D wydają się naturalnym podejściem zwłaszcza do zadań pomiaru pozycji. Proponowane podejście jest szczególnie skuteczne w środowiskach wewnętrznych, w których jakość sygnału GPS jest zazwyczaj niska. Warto podkreślić, że opracowana metoda nie wykorzystywała dodatkowych znaczników.

W stosunku do analogicznego modelu gęstego czas predykcji modelu rzadkiego był ponad dziesięciokrotnie szybszy. W stosunku do metod referencyjnych pracujących na chmurach punktów opracowany model charakteryzował się znacznie wyższą dokładnością przewidywania pozycji. Najlepszy model referencyjny (PCT) posiadał około dwukrotnie większy błąd, a pozostałe 4 – 5-krotnie. W przypadku pomiaru kąta modele referencyjne dawały nieco lepsze wyniki. Najlepszy model PCT uzyskał na zbiorze walidacyjnym błąd 0.1 rad – zbliżony do modelu DroneCropCNN3D, dokonującego predykcji na skadrowanych danych, natomiast globalny model ProjectionCNN3D miał dokładność 0.14 rad.

Warto podkreślić, że modele grafowe w odniesieniu do proponowanej sieci pracującej na rzadkich obrazach w formie wokseli miały krótsze czasy predykcji, mimo wyraźnie gorszej jakości predykcji (szczególnie pozycji). Przyczyn może być kilka – po pierwsze istnieje pewien narzut czasowy z przekształceniem chmury punktów do listy wokseli.

Ponadto sposób wykonywania obliczeń – zwłaszcza operacji splotów – jest odmienny w sieciach grafowych oraz w sieci pracującej na wokselach. Sieć pracująca na wokselach ma sposób pracy zbliżony do gęstej sieci konwolucyjnej. Czas obliczeń zależy od sposobu implementacji operacji rzadkich oraz sposobu reprezentacji danych rzadkich.

Otrzymane wyniki porównano z przedstawionymi w literaturze dla dronów podobnej klasy oraz zadania sformułowanego w ten sam sposób tj. sterowania dronem w zamkniętym pomieszczeniu w oparciu o wizyjny tor pomiarowy. Przykładowo w pracy [111] przedstawiono układ sterowania dronem Parrot ANAFI. Dane z kamery RGB-D wykorzystano jako dodatkowy pomiar, wspomagający dane uzyskane z wbudowanego czujnika inercyjnego drona (IMU) oraz czujnika ultradźwiękowego. Autorzy raportują dokładność sterowania dronem na poziomie kilku centymetrów dla pomiaru pozycji oraz około 0.1 radiana (dla kąta) z częstotliwością pomiaru wynoszącą 30 Hz. W pracy [23] opisano stanowisko testowe, wyposażone w wiele zewnętrznych kamer RGB do rozpoznawania pozycji. Celem poprawy jakości predykcji na dronach umieszczono dodatkowe markery. Obiektem sterowania były dwa drony Parrot Bebop 2. Autorzy uzyskali dokładność lokalizacji do 5 cm w punktach testowych, przy częstotliwości pomiaru wynoszącej 50 Hz. W artykule [87] przedstawiono system wyposażony w zewnętrzną kamerę RGB-D do lokalizacji drona Bebop 2. Jako model predykcyjny wykorzystano detektor YOLO (v3). Autorzy raportują dokładność około 10 cm, przy częstotliwości pomiaru niższej niż 10 Hz.

4.8.6 Śledzenie drona Parrot ANAFI w środowisku symulowanym

Opis eksperymentu

Eksperyment dotyczący drona Parrot ANAFI stał się inspiracją do nowego kierunku badań, które rozpoczęto we współpracy z pracownikami Katedry Systemów Multimedialnych Wydziału Informatyki ZUT. W eksperymencie tym wykorzystano dane ze środowiska symulowanego przygotowanego przez prof. Radosławem Mantiuka oraz dr Marka Wernikowskiego, które jest stosunkowo wierną kopią laboratorium. Głównym założeniem przyjętym w trakcie badań jest to, że proces uczenia modelu do zastosowań praktycznych można zrealizować w oparciu o dane symulowane, odtwarzające rzeczywistą scenę.

Jest to szczególnie ważne ze względu na specyfikę zbierania danych rzeczywistych. Stworzenie zbioru uczącego sieć neuronową wymagało wykonanie aż 66 przelotów, co było zadaniem czasochłonnym. Osobną kwestią jest dokładność uzyskanych etykiet – dla kąta obrotu drona w osi Z posłużono się odczytem z API drona, którego dokładność była ograniczona. Problematiczne było również oznaczanie położenia drona – do etykietowania posłużono się detektorem YOLO, pracującym na obrazach dwuwymiarowych, co ograniczało precyzję informacji o lokalizacji. W środowisku symulowanym mamy natomiast pełną wiedzę o scenie oraz możemy uzyskać dowolnie wysoką dokładność.

Niedokładność danych uczących sprawiała, że potencjał modelu ProjectionCNN3D mógł być zbadany tylko w ograniczonym zakresie. W ramach eksperymentu stworzono (z użyciem programu Blender [20]) uproszczoną symulację sceny obserwowanej we wcześniejszym eksperymencie (rysunek 4.42 b). Na scenie umieszczono model CAD drona udostępniony przez producenta ¹.

¹<https://www.parrot.com/assets/s3fs-public/2022-04/ANAFI.zip>

Model nie posiadał kamery, którą odtworzono w uproszczonej formie. Odwzorowano również proces usuwania tła przy generowaniu chmury punktów (rysunek 4.42d) poprzez losowe wybieranie do 1 500 punktów z tła i siatki oraz podłogi. Odtworzono także szum wprowadzany przez kamerę głębi. Co ważne, środowisko symulacyjne umożliwiało wygenerowanie etykiet z bardzo wysoką dokładnością – zarówno dla kąta, jak i dla pozycji drona.

Wykonanie symulacji pozwoliło na łatwe generowanie danych uczących – korzystając ze wsparcia GPU stworzono w stosunkowo niedługim czasie 1 700 000 próbek uczących. Stworzono także zbiór testowy, odtwarzając dwa rzeczywiste zbiory wykorzystane do testowania modelu we wcześniejszym eksperymencie.

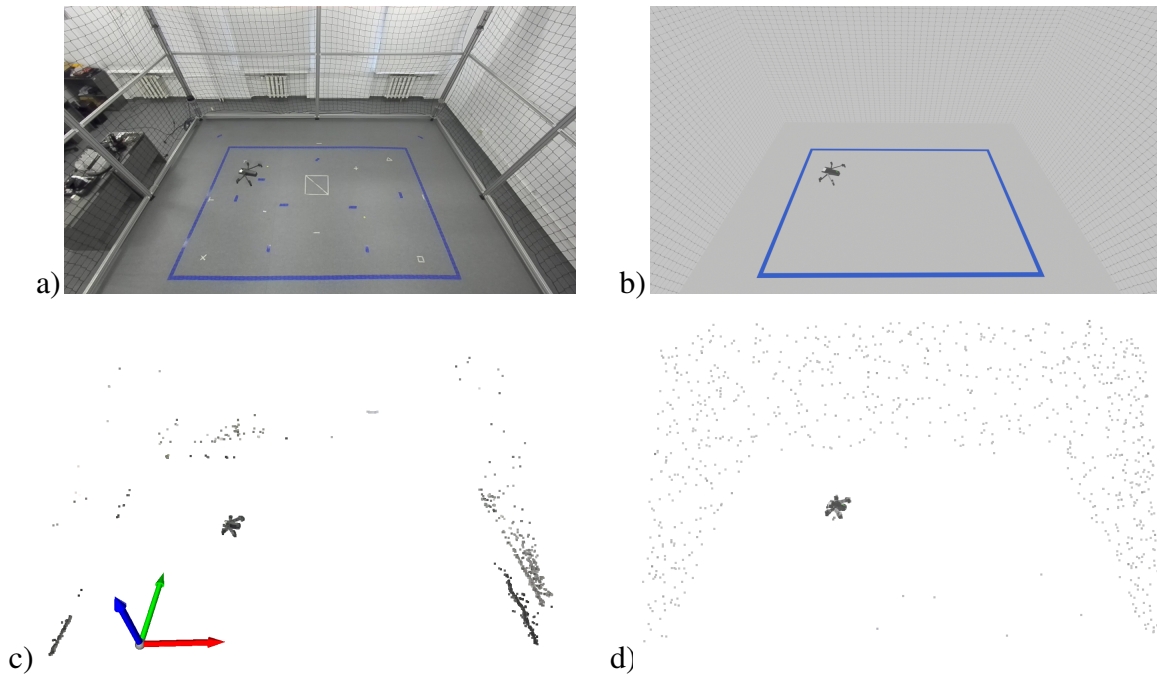
W ramach badań dotyczących symulacji drona wykorzystano trzy modele – ProjectionCNN3D, SlicingCNN3D oraz referencyjny DGCNN ($k = 40$). Wszystkie modele wytrenowano posługując się wyłącznie wygenerowanymi danymi, podzielonymi w proporcji 4:1. Model SlicingCNN3D, wykorzystany w badaniach wstępnych do lokalizacji szkieletu samolotu (obrazy w rozdzielczości $300 \times 300 \times 300$), wymagał przystosowania do pracy z obrazami o rozmiarze $545 \times 545 \times 545$. Cel ten osiągnięto rozszerzając architekturę o warstwę pooling w początkowej części modelu. Zmieniono też liczbę kanałów wejściowych w pierwszej warstwie, by móc obsługiwać obrazy z barwą zapisaną w formacie RGB.

Po nauczaniu modeli przetestowano ich działanie na wygenerowanych danych symulacyjnych, które odtwarzały rzeczywiste trajektorie walidacyjne. Następnie modele SlicingCNN3D oraz DGCNN przetestowano na danych rzeczywistych.

Zastosowanie zdolnego do przewidzenia trzech kątów modelu SlicingCNN3D pozwoliło na rozszerzenie badania o przewidywanie kątów roll oraz pitch. Rozpoczynając badania dotyczące drona, pracując na danych rzeczywistych, planowano przewidywać wszystkie trzy kąty opisujące jego orientację. Okazało się to jednak niewykonalne ze względu na trudności z zebraniem odpowiedniego zestawu danych uczących – dron przez zdecydowaną większość czasu ustawiony jest poziomo, a wyraźne zmiany kątów roll i pitch są krótkotrwałe. Uzyskanie długotrwałego pochylenia do przodu, do tyłu lub na bok wymagałoby długo trwających przelotów z wysoką prędkością, co jest niemożliwe do zrealizowania w bezpieczny sposób w kilkumetrowej klatce. Na dodatek maksymalne wartości kątów (co najwyżej $0.2 - 0.4$ rad) były małe w stosunku do szumu pomiarów uzyskanych z drona, co utrudniało strojenie modelu.

Wyniki

Na rysunku 4.43 oraz w tabeli 4.25 przedstawiono efekty działania badanych modeli na danych symulowanych, które odwzorowują rzeczywistą trajektorie walidacyjną. Dla modelu ProjectionCNN3D uzyskano znacznie wyższą dokładność predykcji kąta niż w przypadku uczenia i testowania modelu na danych rzeczywistych. Zaobserwować można ponad dwukrotne zmniejszenie błędu predykcji – 0.06 rad zamiast 0.14 rad. Zachowana została przy tym wysoka – wyraźnie poniżej rozmiaru woksela – dokładność przewidywania pozycji. Dzięki stworzeniu środowiska symulacyjnego można zaobserwować, że model ProjectionCNN3D umożliwia osiąganie wysokiej jakości predykcji kąta. W przypadku modelu SlicingCNN3D uzyskano wyraźnie niższą dokładność przewidywania pozycji niż dla modelu ProjectionCNN3D, aczkolwiek nadal jest ona zbliżona do rozmiaru woksela – a więc w pełni wystarczająca do kontrolowania drona. Znaczącym zyskiem z użycia modelu Sli-



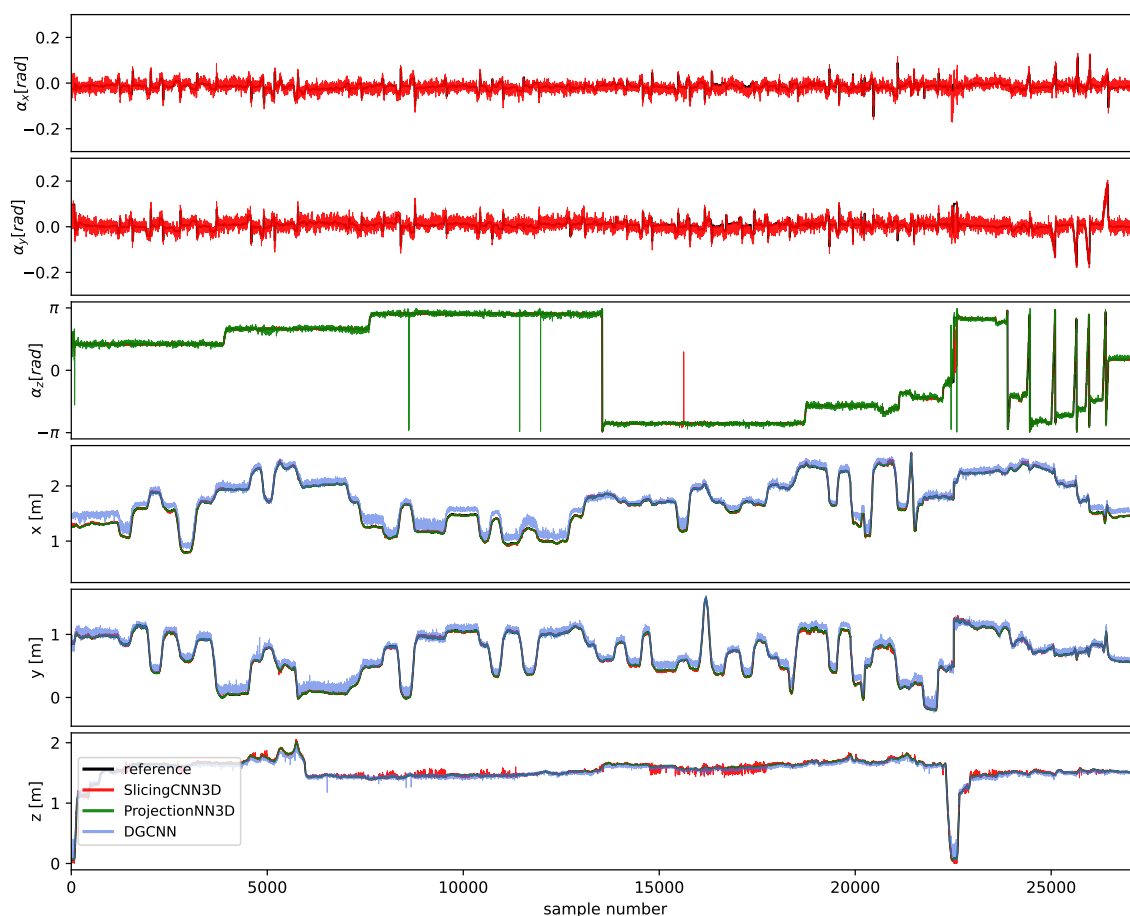
Rysunek 4.42: Porównanie widoku 2D i 3D dla nagrania z kamery – a) i c) oraz symulacji – b) i d). (Źródło: opracowanie własne)

Tabela 4.25: Porównanie badanych modeli na symulowanym zbiorze testowym.

Model	x [cm]	y [cm]	z [cm]	α_x [rad]	α_y [rad]	α_z [rad]
ProjectionCNN3D	0.64	0.46	0.32	—	—	0.061
SlicingCNN3D	0.94	1.05	1.20	0.011	0.013	0.034
DGCNN ($k = 40$)	7.25	5.43	2.48	—	—	—

cingCNN3D jest uzyskanie dwukrotnie wyższej dokładności przewidywania kąta obrotu drona w osi Z – na poziomie 0.03 radiana, co możliwe było dzięki zastosowaniu operacji unfold w architekturze sieci. Co więcej, model ten – w przeciwieństwie do wcześniej używanego do sterowania dronem – pozwala na przewidywanie kątów obrotu w osi X i Y . Dla tych zmiennych błąd predykcji wynosi nieco ponad 0.01 rad. Znacznie niższą jakość predykcji uzyskano z użyciem pracującego bezpośrednio na chmurze punktów modelu DGCNN, dla pozycji wynoszącą kilka centymetrów. Nie udało się uzyskać w rozsądnym czasie zbieżności modelu DGCNN dla kątów, mimo prób uczenia modelu zarówno dla trzech, jak i tylko jednego kąta obrotu.

W ostatniej części eksperymentu przetestowano modele DGCNN oraz SlicingCNN3D na danych rzeczywistych, mimo że trenowane były wyłącznie na danych symulowanych – wyniki przedstawiono w tabeli 4.26 i na rysunku 4.44. W porównaniu obu modeli zdecydowanie lepsze rezultaty uzyskano z użyciem autorskiej sieci, nieznacznie gorsze niż dla modelu DGCNN testowanego na danych symulowanych. Dla architektury SlicingCNN3D dokładność predykcji pozycji wyniosła 4.48 cm w osi X , 5.31 cm w osi Y oraz 14.60 cm



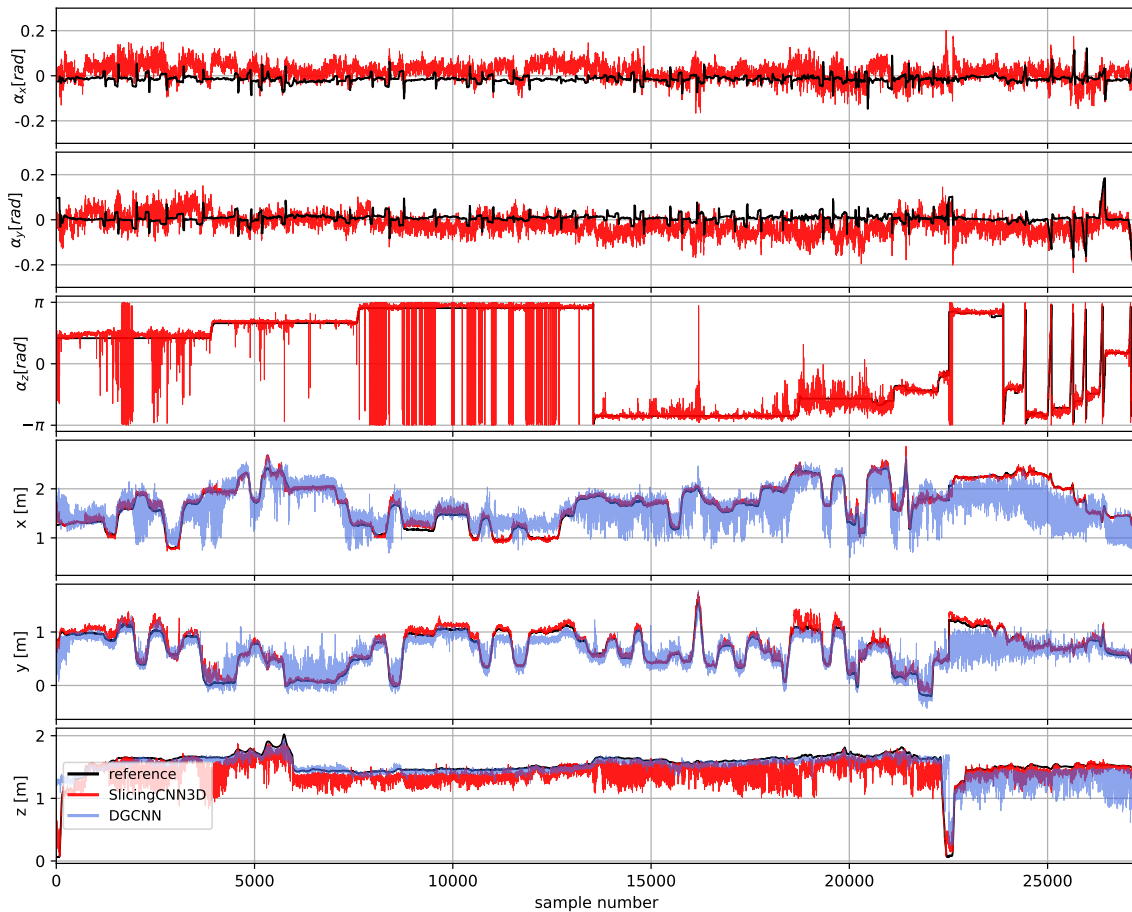
Rysunek 4.43: Porównanie modeli ProjectionCNN3D, SlicingCNN3D oraz DGCNN ($k = 40$) dla danych symulowanych. (Źródło: opracowanie własne)

w osi Z. Z kolei dla kątów obrotu uzyskano około 0.04 rad w osiach X i Y oraz 0.177 rad w osi Z. W przypadku błędów dla kątów należy uwzględnić fakt porównywania predykcji modelu z odczytami z drona, które są zasumowane. Widać to szczególnie dla kątów obrotu względem osi X i Y, których wskazania są tylko zbliżone do zera dla poziomego ułożenia drona.

Osiągnięty rezultat pokazuje duży potencjał zaprezentowanego podejścia. Obecne rozwiązanie pozostawia pole do rozwoju – symulacja może zostać ulepszona choćby poprzez wierniejsze rzeczywistemu generowanie szumu naśladującego proces usuwania tła. W efekcie realnym wydaje się osiąganie wysokiej jakości predykcji dla modelu trenowanego wyłącznie na danych symulowanych, ewentualnie rozszerzonych o niewielki zbiór danych rzeczywistych.

Tabela 4.26: Porównanie badanych modeli na rzeczywistym zbiorze testowym.

Model	x [cm]	y [cm]	z [cm]	α_x [rad]	α_y [rad]	α_z [rad]
SlicingCNN3D	4.48	5.31	14.60	0.040	0.041	0.177
DGCNN ($k = 40$)	22.20	10.64	8.18	—	—	—

Rysunek 4.44: Porównanie modeli SlicingCNN3D oraz DGCNN ($k = 40$) dla danych rzeczywistych. (Źródło: opracowanie własne)

5. Podsumowanie wyników i wnioski

W rozprawie przeprowadzono badania dotyczące sieci splotowych dostosowanych do pracy z danymi rzadkimi, w szczególności chmur punktów, oraz zastosowania takich architektur w wizualnym torze pomiarowym w śledzeniu i sterowaniu robotami mobilnymi. W ramach badań stworzono trzy rzeczywiste zbiory danych oraz wygenerowano trzy zestawy danych sztucznych. Opracowano też pięć autorskich architektur głębokich sieci splotowych.

W oparciu o rzadkie dane syntetyczne dowiedziono empirycznie, że sieci wykorzystujące operację Sparse Submanifold Convolution z powodzeniem mogą być stosowane w zadaniach polegających na lokalizacji i orientacji obiektów, zarówno w przypadku danych dwuwymiarowych, jak i w przypadku trójwymiarowym. W przypadku danych dwuwymiarowych w postaci obrazów w rozdzielczości 4K otrzymano dokładność predykcji pozycji na poziomie kilku pikseli, a dla kąta na poziomie kilku setnych radiana. W przypadku danych trójwymiarowych, złożonych z poddanych wokselizacji chmur punktów (o rozdzielczości $300 \times 300 \times 300$), osiągnięto przewidywanie pozycji z dokładnością na poziomie woksela, a kątów opisujących orientację obiektu na poziomie kilku setnych radiana. Wysoką jakość predykcji kątów osiągnięto stosując sieć z operacją unfold, dzielącą obraz na wiele mniejszych bloków sześciennych.

Wykazano też, że użycie rzadkiej wokselizacji oraz operatora splotu SSC pozwala na znaczące przyspieszenie treningu oraz predykcji modelu. Dla dwuwymiarowych danych syntetycznych osiągnięto kilkunastokrotnie krótszy czas uczenia oraz kilkakrotnie krótszy czas predykcji dla stopnia wypełnienia macierzy poniżej 10^{-2} . Wyznaczono też granicę gęstości danych, dla której stosowanie modelu rzadkiego nie jest korzystne czasowo – wyniosła ona około $6 \cdot 10^{-2}$. Podobną analizę czasu obliczeń przeprowadzono dla danych trójwymiarowych i zaobserwowano bardzo dużą różnicę w czasie uczenia (o czynnik 10^4) oraz dwudziestokrotnie szybszą predykcję dla gęstości danych poniżej 10^{-5} . Wykazano także, że dopiero dla stopnia wypełnienia macierzy danych na poziomie 0.03 zanika korzyść czasowa ze stosowania modelu rzadkiego. Należy przy tym zaznaczyć, że dla danych trójwymiarowych uzyskanych z lidarów czy kamer głębi tak gęste macierze są rzadko spotykane.

W badaniach realizowanych na danych rzeczywistych oraz z użyciem robotów pokazano, że sieci splotowe, a w szczególności sieci wykorzystujące operację SSC, mogą być z powodzeniem stosowane do śledzenia i sterowania robotami. Pozwalają nie tylko na uzyskanie wysokiej jakości przewidywania pozycji i orientacji, ale i osiąganie czasów predykcji pozwalających na zachowanie kontroli nad obiektem, a także wykorzystanie prędkości rejestracji obrazów przez kamerę.

W pierwszym eksperymencie z użyciem danych rzeczywistych osiągnięto dokładność

predykcji pozycji czterech robotów mobilnych na poziomie kilku pikseli, przy rozdzielczości obrazu wejściowego 1280×720 . Pozwoliło to na zastosowanie sieci splotowej do sterowania i nie wymagało dodatkowego wygładzania pomiaru filtrem Kalmana. Osiągnięto przy tym prędkość przetwarzania zbliżoną do prędkości rejestrowania obrazów przez kamerę.

W kolejnym eksperymencie – ograniczonym do śledzenia – wykazano, że uwzględniając naturalną rzadkość danych można stworzyć sieć splotową, która będzie przetwarzać dane rzadkie w rozdzielczości $750 \times 750 \times 1375$ z prędkością około 50 klatek na sekundę, używając wyłącznie jednego rdzenia procesora. Zachowana przy tym zostanie dokładność predykcji efektora ramienia robota na poziomie porównywalnym lub wyższym niż dokładność pomiaru kamery głębi deklarowana przez producenta. Uzyskując tak wysoką dokładność wykazano, że sieć taka, w połączeniu z kamerą głębi, może być wykorzystana w zadaniach robotyki.

W ostatnim eksperymencie wykazano, że korzystając z kamery głębi oraz sieci splotowej z operacją SSC można zrealizować nie tylko śledzenie obiektu, ale i sterowanie nim. Opracowana architektura pozwala na przewidywanie pozycji drona z bardzo wysoką dokładnością (na poziomie centymetrowego woksela) przy rozdzielczości rzadkiego obrazu $545 \times 545 \times 545$ i odległości obserwacji wynoszącej 1.5 – 4 m. Warto podkreślić, że pojedyncza klatka takiego obrazu w postaci gęstej miała by rozmiar rzędu 10^8 , czyli około 100 MB. Przetwarzanie tak dużych obrazów jest na granicy możliwości pojedynczej karty graficznej i byłoby znacznie wolniejsze w stosunku do danych rzadkich. Tak wysoką dokładność uzyskano posługując się kamerą głębi ZED 2i o dokładności pomiaru głębi na poziomie $< 1\%$ dla odległości obserwacji < 3 m. Uzyskana dokładność przewidywania kąta obrotu drona w osi Z wyniosła kilkanaście setnych radiana, mimo że dane uczące były mocno zaszumione ze względu na wykonywanie lotów w pomieszczeniu zamkniętym. Stworzono także drugi model (referencyjny), który dedykowany był przewidywaniu kąta i wykonywał predykcje z błędem mniejszym niż 0.1 rad. Opracowany model został zweryfikowany w zadaniu sterowania dronem wzdłuż zadanej trajektorii. W zadaniu sterowania dokładności odtwarzania trajektorii były nieco niższe niż dokładność samej obserwacji (rzędu kilku centymetrów oraz niespełna 0.2 rad), ponieważ obok błędów pomiarowych występują również niedokładności po stronie elementów wykonawczych oraz zakłócenia. Szczególnie duży negatywny wpływ miała ograniczona dokładność kontrolera drona.

Niemniej wyniki te są wciąż wystarczające do zachowania kontroli nad dronem i odtworzenia zadanej trajektorii. Osiągnięta prędkość przetwarzania danych wyniosła niemal 20 klatek na sekundę i była na poziomie prawie dwukrotnie wyższym niż możliwa do uzyskania częstotliwość komunikacji z dronem za pomocą API.

W badaniu tym wykazano też, że stworzona architektura jest kilkakrotnie lepsza w śledzeniu pozycji drona niż modele referencyjne pracujące bezpośrednio na chmurach punktów, jak PointNet, DGCNN, Point Cloud Transformer, oraz osiąga dokładność predykcji kąta na poziomie zbliżonym do porównywanych metod. Z rozważanych modeli referencyjnych sieć PointNet dawała najgorsze wyniki, co można częściowo tłumaczyć tym, że oryginalnie była projektowana do segmentacji. Natomiast model PCT był nieco lepszy od sieci grafowej DGCNN w przewidywaniu pozycji.

Znacznie wyższą dokładność predykcji pozycji uzyskano pomimo tego, że metody

referencyjne pracują na oryginalnej chmurze punktów, a opracowane modele korzystają z wokselowej reprezentacji, która jest pewnym przybliżeniem oryginalnej sceny.

Praca na rzeczywistych danych, mających ograniczoną precyzję, nie pozwalała na pełne sprawdzenie potencjału stworzonego modelu sieci spłotowej. W związku z tym odtworzono scenę w środowisku symulacyjnym i w ten sposób wykazano, że architektura stosowana do śledzenia i sterowania rzeczywistym dronem pozwala – dla danych symulowanych – na osiągnięcie dokładności przewidywania pozycji około 0.5 cm i kąta na poziomie kilku setnych radiana. Co więcej, zastosowanie modelu używanego w jednym z wcześniejszych eksperymentów pozwoliło na osiągnięcie dwukrotnie dokładniejszej predykcji kąta, przy zachowaniu wysokiej dokładności predykcji pozycji (na poziomie woksele). Model ten – uczony wyłącznie na danych sztucznych – przetestowany na danych rzeczywistych był w stanie przewidywać pozycję drona z dokładnością kilku-kilkunastu centymetrów i kąt z dokładnością lepszą niż 0.2 radiana – świadczy to o stosunkowo dużej odporności takiej struktury. Zastosowany w analogicznym eksperymencie model DGCNN nie był w stanie odtworzyć kąta i miał kilkukrotnie większy błąd przy pomiarze pozycji.

W oparciu o eksperyment dotyczący drona, z użyciem danych rzeczywistych i symulowanych wykazano, że zaproponowane podejścia do przewidywania pozycji lokalizacji i orientacji są konkurencyjne lub lepsze w stosunku do rozwiązań referencyjnych. Zaznaczyć przy tym należy, że jednoczesne rozwiązywanie obu zadań pomiaru lokalizacji i orientacji obiektu jest problem trudnym. Predykcja kąta powinna być niezmiennicza ze względu na przesunięcia obiektu na scenie, natomiast predykcja pozycji powinna być niezmiennicza ze względu na obrót obiektu. Samodzielna operacja spłotu nie ma żadnej z tych właściwości, natomiast przy odpowiednim połączeniu tej operacji z operacjami poolingu, rzutowania, oraz rozbijania na bloki można uzyskać wynik wysokiej jakości.

5.1 Kierunki dalszych badań

Przytoczony eksperyment dotyczący symulacji drona otwiera jeden z kierunków dalszych badań, którym jest uczenie modeli na danych symulowanych (lub z niewielkim dodatkiem danych rzeczywistych) w celu stosowania do danych rzeczywistych. Jest to zasadne ze względu na złożoność procesu zbierania danych w oparciu o rzeczywiste sensory wizyjne i roboty. Generowanie danych syntetycznych jest bezpieczniejsze oraz często mniej kosztowne. Jednak pewnym wyzwaniem jest generowanie obrazów jak najbardziej zbliżonych do naturalnych. Sam proces zbierania danych jest czasochłonny, ponieważ sieci głębokie wymagają odpowiednio dużej próby uczącej do uzyskania zbieżności. Stworzenie środowiska symulacyjnego pozwala natomiast na łatwe generowanie danych uczących. W przypadku badanego wcześniej drona, można łatwo uzyskać dane, które trudno zebrać w rzeczywistości. Przykładowo: problematyczne jest długotrwałe utrzymanie drona w położeniach, dla których wysokie są wartości kątów roll oraz pitch.

Innym kierunkiem przyszłych badań jest skupienie się na dalszej eksploracji zagadnienia konstruowania sieci spłotowych z operacjami spłotu zachowującymi rzadkość macierzy. Jak przedstawiono w [121], wykorzystywana biblioteka SpConv v2 nie w pełni wykorzystuje potencjał obliczeń z użyciem karty graficznej. Obserwowane zyski z przejścia na model rzadki nie są tak duże, jak w przypadku modelu jednowątkowego. Na przykład jednym z problemów jest potencjalnie zmienny rozmiar pojedynczej próbki w

modelu rzadkim. Możliwe jest więc stworzenie wydajniejszej implementacji wybranych operacji, co skutkowałoby jeszcze większym zyskiem czasowym w stosunku do modeli gęstych. Ponadto, nie wszystkie operacje implementowane w modelach gęstych mają swoje odpowiedniki w bibliotekach obsługujących dane rzadkie. Przykładami są tu operacje fold/unfold, których implementacja w postaci rzadkiej nie wydaje się szczególnie trudna.

W oparciu o przeprowadzone badania dotyczące lokalizacji i orientacji możliwe jest tworzenie ogólniejszych modeli, zdolnych do wykrywania obiektów umieszczonych na trójwymiarowej scenie, lub dokonywanie bardziej złożonych pomiarów niż pozycja i kąty.

Ciekawym obszarem przyszłych badań wydają się także modele niewymagające wokselizacji, a opracowanie wydajnych operatorów spłotu czy poolingu dla danych bezpośrednio w postaci chmur punktów jest wciąż przedmiotem intensywnych badań.

Spis literatury

- [1] A brief history and overview of LiDAR technology and how it has evolved. <https://sickconnect.com/what-is-lidar-technology/>. Dostęp: 2025-03-14.
- [2] Martín Abadi i in. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org. 2015. URL: <https://www.tensorflow.org/>.
- [3] Amir Abboud i in. “The Time Complexity of Fully Sparse Matrix Multiplication”. W: Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), s. 4670–4703. DOI: 10.1137/1.9781611977912.167. URL: <https://epubs.siam.org/doi/abs/10.1137/1.9781611977912.167>.
- [4] Mohamed Afham i in. “CrossPoint: Self-Supervised Cross-Modal Contrastive Learning for 3D Point Cloud Understanding”. W: 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2022, s. 9892–9902. DOI: 10.1109/CVPR52688.2022.00967.
- [5] Mohammed Aledhari i in. “Sensor Fusion for Drone Detection”. W: 2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring). 2021, s. 1–7. DOI: 10.1109/VTC2021-Spring51267.2021.9448699.
- [6] Md. Hazrat Ali i in. “Vision-based Robot Manipulator for Industrial Applications”. W: Procedia Computer Science 133 (2018). International Conference on Robotics and Smart Manufacturing (RoSMa2018), s. 205–212. ISSN: 1877-0509. DOI: <https://doi.org/10.1016/j.procs.2018.07.025>. URL: <https://www.sciencedirect.com/science/article/pii/S1877050918309694>.
- [7] Les Atlas, Toshiteru Honna i Robert Marks. “An Artificial Neural Network for Spatio-Temporal Bipolar Patterns: Application to Phoneme Classification”. W: Neural Information Processing Systems. Red. D. Anderson. T. O. American Institute of Physics, 1987. URL: https://proceedings.neurips.cc/paper_files/paper/1987/file/98f13708210194c475687be6106a3b84-Paper.pdf.
- [8] Matan Atzmon, Haggai Maron i Yaron Lipman. Point Convolutional Neural Networks by Extension Operators. 2018. arXiv: 1803.10091 [cs.CV].
- [9] Beginner’s guide to depth (Updated). <https://www.intelrealsense.com/beginners-guide-to-depth/>. Dostęp: 2025-03-10.
- [10] Saifullahi Aminu Bello i in. “Review: Deep Learning on 3D Point Clouds”. W: Remote Sensing 12.11 (2020). ISSN: 2072-4292. DOI: 10.3390/rs12111729. URL: <https://www.mdpi.com/2072-4292/12/11/1729>.

- [11] Axel Berg, Magnus Oskarsson i Mark O'Connor. "Points to Patches: Enabling the Use of Self-Attention for 3D Shape Recognition". W: 2022 26th International Conference on Pattern Recognition (ICPR). Los Alamitos, CA, USA: IEEE Computer Society, sierp. 2022, s. 528–534. DOI: 10.1109/ICPR56361.2022.9956172. URL: <https://doi.ieeecomputersociety.org/10.1109/ICPR56361.2022.9956172>.
- [12] Joan Bruna i in. Spectral Networks and Locally Connected Networks on Graphs. 2014. arXiv: 1312.6203 [cs.LG]. URL: <https://arxiv.org/abs/1312.6203>.
- [13] R. Qi Charles i in. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". W: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2017, s. 77–85. DOI: 10.1109/CVPR.2017.16.
- [14] F. Chaumette i E. Marchand. "Recent results in visual servoing for robotics applications". W: 8th ESA Workshop on Advanced Space Technologies for Robotics and Automation, ASTRA 2004. Noordwijk, The Netherlands, 2004, s. 471–478.
- [15] Can Chen, Luca Zanotti Fragonara i Antonios Tsourdos. "GAPointNet: Graph attention based point neural network for exploiting local feature of point cloud". W: Neurocomputing 438 (2021), s. 122–132. ISSN: 0925-2312. DOI: <https://doi.org/10.1016/j.neucom.2021.01.095>. URL: <https://www.sciencedirect.com/science/article/pii/S0925231221001776>.
- [16] Xingye Chen i in. "PointSCNet: Point Cloud Structure and Correlation Learning Based on Space-Filling Curve-Guided Sampling". W: Symmetry 14.1 (2022). ISSN: 2073-8994. DOI: 10.3390/sym14010008. URL: <https://www.mdpi.com/2073-8994/14/1/8>.
- [17] Yukang Chen i in. "Focal Sparse Convolutional Networks for 3D Object Detection". W: 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2022, s. 5418–5427. DOI: 10.1109/CVPR52688.2022.00535.
- [18] Graziano Chesi i Koichi Hashimoto. Visual Servoing via Advanced Numerical Methods. 1st. Springer Publishing Company, Incorporated, 2010. ISBN: 1849960887.
- [19] Christopher Choy, JunYoung Gwak i Silvio Savarese. "4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks". W: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019, s. 3075–3084.
- [20] Blender Online Community. Blender - a 3D modelling and rendering package. Blender Foundation. Stichting Blender Foundation, Amsterdam, 2018. URL: <http://www.blender.org>.
- [21] John J Craig. Wprowadzenie do robotyki: mechanika i sterowanie. Warszawa: Wydawnictwa Naukowo-Techniczne, 1995. ISBN: 83-204-1835-6.
- [22] Hanz Cuevas-Velasquez i in. Hybrid Multi-camera Visual Servoing to Moving Target. 2018. arXiv: 1803.02285 [cs.CV].
- [23] Heng Deng i in. "Indoor Multi-Camera-Based Testbed for 3-D Tracking and Control of UAVs". W: IEEE Transactions on Instrumentation and Measurement 69.6 (2020), s. 3139–3156. DOI: 10.1109/TIM.2019.2928615.
- [24] Jacob Devlin i in. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. 2019. arXiv: 1810.04805 [cs.CL]. URL: <https://arxiv.org/abs/1810.04805>.

- [25] James Diebel. “Representing Attitude : Euler Angles , Unit Quaternions , and Rotation Vectors”. W: 2006. URL: <https://api.semanticscholar.org/CorpusID:16450526>.
- [26] Sedat Dogru i Lino Marques. “Drone Detection Using Sparse Lidar Measurements”. W: IEEE Robotics and Automation Letters 7.2 (2022), s. 3062–3069. DOI: 10.1109/LRA.2022.3145498.
- [27] Alexey Dosovitskiy i in. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. W: International Conference on Learning Representations. 2021.
- [28] Lunhao Duan i in. “ConDaFormer: Disassembled Transformer with Local Structure Enhancement for 3D Point Cloud Understanding”. W: Thirty-seventh Conference on Neural Information Processing Systems. 2023.
- [29] Nico Engel, Vasileios Belagiannis i Klaus Dietmayer. “Point Transformer”. W: IEEE Access 9 (2021), s. 134826–134840. DOI: 10.1109/ACCESS.2021.3116304.
- [30] Francis Engelmann i in. “Exploring Spatial Context for 3D Semantic Segmentation of Point Clouds”. W: 2017 IEEE International Conference on Computer Vision Workshops (ICCVW). 2017, s. 716–724. DOI: 10.1109/ICCVW.2017.90.
- [31] Euler’s Rotation Theorem. <https://mathworld.wolfram.com/EulersRotationTheorem.html>. Dostęp: 2025-04-04.
- [32] Event-based vision camera. <https://centuryarks.com/en/about-evcam/>. Dostęp: 2025-03-10.
- [33] Yifan Feng i in. “GVCNN: Group-View Convolutional Neural Networks for 3D Shape Recognition”. W: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2018, s. 264–272. DOI: 10.1109/CVPR.2018.00035.
- [34] Matthias Fey i Jan E. Lenssen. “Fast Graph Representation Learning with PyTorch Geometric”. W: ICLR Workshop on Representation Learning on Graphs and Manifolds. 2019.
- [35] Adriano Garcia i in. “A Convolutional Neural Network Vision System Approach to Indoor Autonomous Quadrotor Navigation”. W: 2019 International Conference on Unmanned Aircraft Systems (ICUAS). 2019, s. 1344–1352. DOI: 10.1109/ICUAS.2019.8798183.
- [36] Rodrigo Gonzalez i Paolo Dabov. “Performance Assessment of an Ultra Low-Cost Inertial Measurement Unit for Ground Vehicle Navigation”. W: Sensors 19.18 (2019). ISSN: 1424-8220. DOI: 10.3390/s19183865.
- [37] Benjamin Graham, Martin Engelcke i Laurens van der Maaten. “3D Semantic Segmentation with Submanifold Sparse Convolutional Networks”. W: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2018, s. 9224–9232. DOI: 10.1109/CVPR.2018.00961.
- [38] Benjamin Graham i Laurens van der Maaten. Submanifold Sparse Convolutional Networks. 2017. arXiv: 1706.01307 [cs.NE]. URL: <https://arxiv.org/abs/1706.01307>.
- [39] Graham, Benjamin and Engelcke, Martin and van der Maaten, Laurens. SparseConvNet: Submanifold Sparse Convolutional Networks. Maj 2018. URL: <https://github.com/facebookresearch/SparseConvNet>.
- [40] Jia Guo i in. “Convolutional Neural Network-Based Robot Control for an Eye-in-Hand Camera”. W: IEEE Transactions on Systems, Man, and Cybernetics: Systems 53.8 (2023), s. 4764–4775. DOI: 10.1109/TSMC.2023.3257416.

- [41] Meng-Hao Guo i in. PCT: Point cloud transformer. en. 2021. DOI: 10.1007/s41095-021-0229-5. URL: <http://dx.doi.org/10.1007/s41095-021-0229-5>.
- [42] Abdullah Hamdi, Silvio Giancola i Bernard Ghanem. “MVTN: Multi-View Transformation Network for 3D Shape Recognition”. W: 2021 IEEE/CVF International Conference on Computer Vision (ICCV). 2021, s. 1–11. DOI: 10.1109/ICCV48922.2021.00007.
- [43] Xian-Feng Han i in. “3CROSSNet: Cross-Level Cross-Scale Cross-Attention Network for Point Cloud Representation”. W: IEEE Robotics and Automation Letters 7.2 (2022), s. 3718–3725. DOI: 10.1109/LRA.2022.3147907.
- [44] Harald Weber. SICK AG whitepaper. LiDAR sensor functionality and variants. whitepaper. 2018. URL: https://cdn.sick.com/media/docs/3/63/963/whitepaper_lidar_en_im0079963.pdf.
- [45] Sepp Hochreiter i Jürgen Schmidhuber. “LSTM can solve hard long time lag problems”. W: Proceedings of the 10th International Conference on Neural Information Processing Systems. NIPS’96. Denver, Colorado: MIT Press, 1996, 473–479.
- [46] Qingyong Hu i in. “RandLA-Net: Efficient Semantic Segmentation of Large-Scale Point Clouds”. W: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Los Alamitos, CA, USA: IEEE Computer Society, czer. 2020, s. 11105–11114. DOI: 10.1109/CVPR42600.2020.01112. URL: <https://doi.ieeecomputersociety.org/10.1109/CVPR42600.2020.01112>.
- [47] Yuhao Huang i in. “Voxel or Pillar: Exploring Efficient Point Cloud Representation for 3D Object Detection”. W: Proceedings of the AAAI Conference on Artificial Intelligence 38.3 (2024), s. 2426–2435. DOI: 10.1609/aaai.v38i3.28018. URL: <https://ojs.aaai.org/index.php/AAAI/article/view/28018>.
- [48] Peter J. Huber. “Robust estimation of a location parameter”. W: Annals of Mathematical Statistics 35.1 (mar. 1964), s. 73–101. ISSN: 0003-4851. DOI: 10.1214/aoms/1177703732. URL: <http://dx.doi.org/10.1214/aoms/1177703732>.
- [49] Intel. Intel RealSense TMD400 Series Product Family Datasheet. Spraw. tech. 2019.
- [50] iRobot Corp. iRobot® Create® 2 Open Interface (OI) Specification based on the iRobot® Roomba® 600. Spraw. tech. iRobot Corp, 2018. URL: <https://edu.irobot.com/learning-library/create-2-oi-spec>.
- [51] Jan Rodziewicz-Bielewicz. “Comparison of Point Cloud-Based Deep Learning Methods for Drone State Prediction”. W: ICSYS 2024 (21 International Conference of Students and Young Scientists, September 23-25, 2024, Dziwnów).
- [52] Jan Rodziewicz-Bielewicz i Marcin Korzeń. “Application of Sparse Deep Learning Model to Robot Arm Tracking”. W: PP-RAI’2022 (3rd Polish Conference on Artificial Intelligence, April 25-27, 2022, Gdynia).
- [53] Jan Rodziewicz-Bielewicz i Marcin Korzeń. “Indoor drone trajectory planning with depth-vision-based feedback”. W: PP-RAI’2024 (5th Polish Conference on Artificial Intelligence, April 18-20, 2024, Warszawa).
- [54] Jan Rodziewicz-Bielewicz, Marek Wernikowski, Marcin Korzeń i Radosław Mantiuk. “Training 3D CNN on synthetic data for real drone tracking application”. W: PP-RAI’2025 (6th Polish Conference on Artificial Intelligence, April 7-9, 2024, Katowice).

- [55] Tianyuan Jiang i in. “VIC-Net: Voxelization Information Compensation Network for Point Cloud 3D Object Detection”. W: 2021 IEEE International Conference on Robotics and Automation (ICRA). 2021, s. 13408–13414. DOI: 10.1109/ICRA48506.2021.9561597.
- [56] Glenn Jocher. ultralytics/yolov5: 2022-8-9. Wer. 2022-8-9. Sierp. 2022. URL: <https://github.com/ultralytics/yolov5>.
- [57] Roger R. Labbe Jr. filterpy: Python Kalman filtering and optimal estimation library. 2015. URL: <https://github.com/rlabbe/filterpy>.
- [58] Asako Kanezaki, Yasuyuki Matsushita i Yoshifumi Nishida. “RotationNet: Joint Object Categorization and Pose Estimation Using Multiviews from Unsupervised Viewpoints”. W: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2018, s. 5010–5019. DOI: 10.1109/CVPR.2018.00526.
- [59] Shinya Kawabata, Jae Hoon Lee i Shingo Okamoto. “Obstacle Avoidance Navigation Using Horizontal Movement for a Drone Flying in Indoor Environment”. W: 2019 International Conference on Control, Artificial Intelligence, Robotics and Optimization (ICCAIRO). 2019, s. 1–6. DOI: 10.1109/ICCAIRO47923.2019.00009.
- [60] Thomas N. Kipf i Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. 2017. arXiv: 1609.02907 [cs.LG]. URL: <https://arxiv.org/abs/1609.02907>.
- [61] Roman Klokov i Victor Lempitsky. “Escape from Cells: Deep Kd-Networks for the Recognition of 3D Point Cloud Models”. W: 2017 IEEE International Conference on Computer Vision (ICCV). 2017, s. 863–872. DOI: 10.1109/ICCV.2017.99.
- [62] Teuvo Kohonen. “Automatic formation of topological maps of patterns in a self-organizing system”. W: Proceedings of 2SCIA, Scand. Conference on Image Analysis. Red. E. Oja i O. Simula. Helsinki, Finland, 1981, s. 214–220.
- [63] A. Komarichev, Z. Zhong i J. Hua. “A-CNN: Annularly Convolutional Neural Networks on Point Clouds”. W: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Los Alamitos, CA, USA: IEEE Computer Society, 2019, s. 7413–7422. DOI: 10.1109/CVPR.2019.00760.
- [64] Andrey V. Kudryavtsev i in. “Eye-in-Hand Visual Servoing of Concentric Tube Robots”. W: IEEE Robotics and Automation Letters 3.3 (2018), s. 2315–2321. DOI: 10.1109/LRA.2018.2807592.
- [65] Roger R. Labbe Jr. Kalman And Bayesian Filters In Python. 2015. URL: <https://archive.org/details/KalmanAndBayesianFiltersInPython>.
- [66] Truc Le i Ye Duan. “PointGrid: A Deep Network for 3D Shape Understanding”. W: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2018, s. 9204–9214. DOI: 10.1109/CVPR.2018.00959.
- [67] Dilong Li i in. “Graph Neural Networks in Point Clouds: A Survey”. W: Remote Sensing 16.14 (2024). ISSN: 2072-4292. DOI: 10.3390/rs16142518. URL: <https://www.mdpi.com/2072-4292/16/14/2518>.
- [68] Jiaxin Li, Ben M. Chen i Gim Hee Lee. “SO-Net: Self-Organizing Network for Point Cloud Analysis”. W: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2018, s. 9397–9406. DOI: 10.1109/CVPR.2018.00979.

- [69] Ling Li i in. “Autonomous Removing Foreign Objects for Power Transmission Line by Using a Vision-Guided Unmanned Aerial Manipulator”. W: Journal of Intelligent and Robotic Systems 103.2 (2021), s. 23. ISSN: 1573-0409. DOI: 10.1007/s10846-021-01482-3.
- [70] Xiaohan Li i in. “UPG: 3D vision-based prediction framework for robotic grasping in multi-object scenes”. W: Knowledge-Based Systems 270 (2023), s. 110491. ISSN: 0950-7051. DOI: <https://doi.org/10.1016/j.knosys.2023.110491>.
- [71] Yangyan Li i in. “PointCNN: Convolution On X-Transformed Points”. W: Advances in Neural Information Processing Systems. Red. S. Bengio i in. T. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/f5f8590cd58a54e94377e6ae2eded4d9-Paper.pdf.
- [72] Xinwu Liang i in. “Image-Based Position Control of Mobile Robots With a Completely Unknown Fixed Camera”. W: IEEE Transactions on Automatic Control 63.9 (2018), s. 3016–3023. DOI: 10.1109/TAC.2018.2793458.
- [73] P. Lichtsteiner i T. Delbruck. “A 64x64 aer logarithmic temporal derivative silicon retina”. W: Research in Microelectronics and Electronics, 2005 PhD. T. 2. 2005, s. 202–205. DOI: 10.1109/RME.2005.1542972.
- [74] Tsung-Yi Lin i in. “Microsoft COCO: Common Objects in Context”. W: CoRR abs/1405.0312 (2014). arXiv: 1405.0312. URL: <http://arxiv.org/abs/1405.0312>.
- [75] Zhi-Hao Lin, Sheng-Yu Huang i Yu-Chiang Frank Wang. “Convolution in the Cloud: Learning Deformable Kernels in 3D Graph Convolution Networks for Point Cloud Analysis”. W: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2020, s. 1797–1806. DOI: 10.1109/CVPR42600.2020.00187.
- [76] Fangyu Liu i in. “3DCNN-DQN-RNN: A Deep Reinforcement Learning Framework for Semantic Parsing of Large-Scale 3D Point Clouds”. W: 2017 IEEE International Conference on Computer Vision (ICCV). 2017, s. 5679–5688. DOI: 10.1109/ICCV.2017.605.
- [77] Shan Liu i in. 3D point cloud analysis: Traditional, deep learning, and explainable machine learning methods. Springer, 2021.
- [78] Xinhai Liu i in. “Point2Sequence: learning the shape representation of 3D point clouds with an attention-based sequence to sequence network”. W: Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence. AAAI’19/IAAI’19/EAAI’19. Honolulu, Hawaii, USA: AAAI Press, 2019. ISBN: 978-1-57735-809-1. DOI: 10.1609/aaai.v33i01.33018778. URL: <https://doi.org/10.1609/aaai.v33i01.33018778>.
- [79] Yongcheng Liu i in. “Relation-Shape Convolutional Neural Network for Point Cloud Analysis”. W: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Los Alamitos, CA, USA: IEEE Computer Society, 2019, s. 8887–8896. DOI: 10.1109/CVPR.2019.00910. URL: <https://doi.ieeecomputersociety.org/10.1109/CVPR.2019.00910>.
- [80] Zhijian Liu i in. “Point-Voxel CNN for Efficient 3D Deep Learning”. W: Advances in Neural Information Processing Systems. Red. H. Wallach i in. T. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/5737034557ef5b8c02c0e46513b98f90-Paper.pdf.

- [81] Dening Lu i in. “3DCTN: 3D Convolution-Transformer Network for Point Cloud Classification”. W: IEEE Transactions on Intelligent Transportation Systems 23.12 (2022), s. 24854–24865. DOI: 10.1109/TITS.2022.3198836.
- [82] Dening Lu i in. “3DGTN: 3-D Dual-Attention GLocal Transformer Network for Point Cloud Classification and Segmentation”. W: IEEE Transactions on Geoscience and Remote Sensing 62 (2024), s. 1–13. DOI: 10.1109/TGRS.2024.3393845.
- [83] Qiang Lu i in. “PointNGCNN: Deep convolutional networks on 3D point clouds with neighborhood graph filters”. W: Computers & Graphics 86 (2020), s. 42–51. ISSN: 0097-8493. DOI: <https://doi.org/10.1016/j.cag.2019.11.005>. URL: <https://www.sciencedirect.com/science/article/pii/S0097849319301748>.
- [84] Kevin M. Lynch i Frank C. Park. Modern Robotics: Mechanics, Planning, and Control. 1st. USA: Cambridge University Press, 2017. ISBN: 1107156300.
- [85] Jiageng Mao i in. “Voxel Transformer for 3D Object Detection”. W: 2021 IEEE/CVF International Conference on Computer Vision (ICCV). 2021.
- [86] Daniel Maturana i Sebastian Scherer. “VoxNet: A 3D Convolutional Neural Network for real-time object recognition”. W: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2015, s. 922–928. DOI: 10.1109/IROS.2015.7353481.
- [87] Benjamaporn Nenchoo i Suradet Tantrairatn. “Real-Time 3D UAV Pose Estimation by Visualization”. W: Proceedings 39.1 (2019). ISSN: 2504-3900. DOI: 10.3390/proceedings2019039018. URL: <https://www.mdpi.com/2504-3900/39/1/18>.
- [88] van-Truong Nguyen i in. “Vision-Based Pick and Place Control System for Industrial Robots Using an Eye-in-Hand Camera”. W: IEEE Access 13 (2025), s. 25127–25140. DOI: 10.1109/ACCESS.2025.3536496.
- [89] Jongyoun Noh, Sanghoon Lee i Bumsub Ham. “HVPR: Hybrid Voxel-Point Representation for Single-Stage 3D Object Detection”. W: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2021, s. 14605–14614.
- [90] Ridha Nurlaili, Indra Adji Sulistijono i Anhar Risnumawan. “Mobile Robot Position Control Using Computer Vision”. W: 2019 International Electronics Symposium (IES). 2019, s. 382–386. DOI: 10.1109/ELECSYM.2019.8901619.
- [91] Yatian Pang i in. “Masked Autoencoders for Point Cloud Self-supervised Learning”. W: Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part II. Tel Aviv, Israel: Springer-Verlag, 2022, 604–621. ISBN: 978-3-031-20085-4. DOI: 10.1007/978-3-031-20086-1_35. URL: https://doi.org/10.1007/978-3-031-20086-1_35.
- [92] Parrot Drone SAS. ANAFI White Paper v1.4. Spraw. tech. 2020.
- [93] Adam Paszke i in. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. W: Advances in Neural Information Processing Systems 32. Red. H. Wallach i in. Curran Associates, Inc., 2019, s. 8024–8035. URL: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.
- [94] F. Pedregosa i in. “Scikit-learn: Machine Learning in Python”. W: Journal of Machine Learning Research 12 (2011), s. 2825–2830.

- [95] Kim-Phuong Phung i in. “Multi-model Deep Learning Drone Detection and Tracking in Complex Background Conditions”. W: 2021 International Conference on Advanced Technologies for Communications (ATC). 2021, s. 189–194. DOI: 10.1109/ATC52653.2021.9598317.
- [96] Marcin Pluciński i Marcin Korzeń. “Application of the Peano Curve for the Robot Trajectory Generating”. W: Advances in Information Processing and Protection. Red. Jerzy Pejaś i Khalid Saeed. Boston, MA: Springer US, 2007, s. 25–34. ISBN: 978-0-387-73137-7.
- [97] Adrien Poulenard i in. “Effective Rotation-Invariant Point CNN with Spherical Harmonics Kernels”. W: 2019 International Conference on 3D Vision (3DV). Los Alamitos, CA, USA: IEEE Computer Society, wrz. 2019, s. 47–56. DOI: 10.1109/3DV.2019.00015. URL: <https://doi.ieeecomputersociety.org/10.1109/3DV.2019.00015>.
- [98] Charles R. Qi i in. “PointNet++: deep hierarchical feature learning on point sets in a metric space”. W: Proceedings of the 31st International Conference on Neural Information Processing Systems. NIPS’17. Long Beach, California, USA: Curran Associates Inc., 2017, 5105–5114. ISBN: 9781510860964.
- [99] Shengwei Qin, Zhong Li i Ligang Liu. “Robust 3D Shape Classification via Non-local Graph Attention Network”. W: 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2023, s. 5374–5383. DOI: 10.1109/CVPR52729.2023.00520.
- [100] qinglew. Implementation of PCT(Point Cloud Transformer) in PyTorch. <https://github.com/qinglew/PointCloudTransformer>. 2021.
- [101] Haoxi Ran i in. “Learning Inner-Group Relations on Point Clouds”. W: 2021 IEEE/CVF International Conference on Computer Vision (ICCV). Los Alamitos, CA, USA: IEEE Computer Society, paź. 2021, s. 15457–15467. DOI: 10.1109/ICCV48922.2021.01519. URL: <https://doi.ieeecomputersociety.org/10.1109/ICCV48922.2021.01519>.
- [102] Joseph Redmon i in. “You Only Look Once: Unified, Real-Time Object Detection”. W: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016, s. 779–788. DOI: 10.1109/CVPR.2016.91.
- [103] Gernot Riegler, Ali Osman Ulusoy i Andreas Geiger. “OctNet: Learning Deep 3D Representations at High Resolutions”. W: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2017, s. 6620–6629. DOI: 10.1109/CVPR.2017.701.
- [104] J. Rodziewicz-Bielewicz i M. Korzeń. Code and data for "Comparison of Graph Fitting and Sparse Deep Learning Model for Robot Pose Estimation". 2022. URL: https://github.com/jrod12/py_realsense_sensors.
- [105] J. Rodziewicz-Bielewicz i M. Korzeń. Code and data for "Sparse Convolutional Neural Network for Localization and Orientation Prediction and Application to Drone Control". 27th European Conference on Artificial Intelligence (ECAI-2024), Santiago de Compostela, Spain. 2024. DOI: 10.5281/zenodo.13354053. URL: <https://doi.org/10.5281/zenodo.13354053>.
- [106] Jan Rodziewicz-Bielewicz i Marcin Korzeń. “Vision-Based Mobile Robots Control Along a Given Trajectory”. W: Artificial Intelligence and Soft Computing. Red. Leszek Rutkowski i in. Cham: Springer Nature Switzerland, 2023, s. 69–77. ISBN: 978-3-031-42508-0.

- [107] Jan Rodziewicz-Bielewicz i Marcin Korzeń. “Comparison of Graph Fitting and Sparse Deep Learning Model for Robot Pose Estimation”. W: *Sensors* 22.17 (2022). ISSN: 1424-8220. DOI: 10.3390/s22176518. URL: <https://www.mdpi.com/1424-8220/22/17/6518>.
- [108] Jan Rodziewicz-Bielewicz i Marcin Korzeń. “Sparse Convolutional Neural Network for Localization and Orientation Prediction and Application to Drone Control”. W: IOS Press, paź. 2024. DOI: 10.3233/faia240530. URL: <http://dx.doi.org/10.3233/FAIA240530>.
- [109] Rotations and Orientation. <https://www.cs.utexas.edu/~theshark/courses/cs354/lectures/cs354-14.pdf>. Dostęp: 2025-04-04.
- [110] Septimiu E. Salcudean i in. “Robot-Assisted Medical Imaging: A Review”. W: *Proceedings of the IEEE* 110.7 (2022), s. 951–967. DOI: 10.1109/JPROC.2022.3162840.
- [111] Milton C.P. Santos i in. “Indoor low-cost localization system for controlling aerial robots”. W: *Control Engineering Practice* 61 (2017), s. 93–111. ISSN: 0967-0661. DOI: <https://doi.org/10.1016/j.conengprac.2017.01.011>.
- [112] Franco Scarselli i in. “The Graph Neural Network Model”. W: *IEEE Transactions on Neural Networks* 20.1 (2009), s. 61–80. DOI: 10.1109/TNN.2008.2005605.
- [113] Bernard Schmidt i Lihui Wang. “Depth camera based collision avoidance via active robot control”. W: *Journal of Manufacturing Systems* 33.4 (2014), s. 711–718. ISSN: 0278-6125. DOI: <https://doi.org/10.1016/j.jmsy.2014.04.004>.
- [114] Shaoshuai Shi i in. “PV-RCNN: Point-Voxel Feature Set Abstraction for 3D Object Detection”. W: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.
- [115] Martin Simonovsky i Nikos Komodakis. “Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs”. W: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, s. 29–38. DOI: 10.1109/CVPR.2017.11.
- [116] Spconv Contributors. Spconv: Spatially Sparse Convolution Library. Paź. 2022. URL: <https://github.com/traveller59/spconv>.
- [117] Stereolabs Inc. ZED 2i Camera and SDK Overview. Spraw. tech. 2022.
- [118] Hang Su i in. “Multi-view Convolutional Neural Networks for 3D Shape Recognition”. W: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, s. 945–953. DOI: 10.1109/ICCV.2015.114.
- [119] Fredrik Svanström, Cristofer Englund i Fernando Alonso-Fernandez. “Real-Time Drone Detection and Tracking With Visible, Thermal and Acoustic Sensors”. W: *2020 25th International Conference on Pattern Recognition (ICPR)*. 2021, s. 7265–7272. DOI: 10.1109/ICPR48806.2021.9413241.
- [120] Mingxing Tan i Quoc Le. “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”. W: *Proceedings of the 36th International Conference on Machine Learning*. Red. Kamalika Chaudhuri i Ruslan Salakhutdinov. T. 97. *Proceedings of Machine Learning Research*. PMLR, 2019, s. 6105–6114. URL: <https://proceedings.mlr.press/v97/tan19a.html>.
- [121] Haotian Tang i in. “TorchSparse++: Efficient Training and Inference Framework for Sparse Convolution on GPUs”. W: *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2023.

- [122] An Tao. DGCNN.pytorch. <https://github.com/antao97/dgcnn.pytorch>. 2020.
- [123] Intel RealSense team. Intel Realsense Python SDK. 2025. URL: <https://dev.intelrealsense.com/docs/python2>.
- [124] UFactory. uArm Swift Pro Developers Guide V1.0.4. Spraw. tech. UFactory.
- [125] UFactory. uArm-Python-SDK. 2020. URL: <https://github.com/uArm-Developer/uArm-Python-SDK>.
- [126] Eren Unlu i in. “An autonomous drone surveillance and tracking architecture”. W: Electronic Imaging 31.15 (2019), s. 35–1–35–1. DOI: 10.2352/ISSN.2470-1173.2019.15.AVM-035.
- [127] Eren Unlu i in. “Deep learning-based strategies for the detection and tracking of drones using several cameras”. W: IPSJ Transactions on Computer Vision and Applications 11.1 (lip. 2019), s. 7.
- [128] Ashish Vaswani i in. “Attention is All you Need”. W: Advances in Neural Information Processing Systems. Red. I. Guyon i in. T. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.
- [129] Pauli Virtanen i in. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. W: Nature Methods 17 (2020), s. 261–272. DOI: 10.1038/s41592-019-0686-2.
- [130] Changshuo Wang i in. “GPSFormer: A Global Perception and Local Structure Fitting-Based Transformer for Point Cloud Understanding”. W: Computer Vision – ECCV 2024. Red. Aleš Leonardis i in. Cham: Springer Nature Switzerland, 2025, s. 75–92. ISBN: 978-3-031-73242-3.
- [131] Chu Wang, Marcello Pelillo i Kaleem Siddiqi. “Dominant Set Clustering and Pooling for Multi-View 3D Object Recognition”. W: ArXiv abs/1906.01592 (2019). URL: <https://api.semanticscholar.org/CorpusID:6175076>.
- [132] Lei Wang i in. “Graph Attention Convolution for Point Cloud Semantic Segmentation”. W: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2019, s. 10288–10297. DOI: 10.1109/CVPR.2019.01054.
- [133] Peng-Shuai Wang i in. “O-CNN: octree-based convolutional neural networks for 3D shape analysis”. W: ACM Trans. Graph. 36.4 (lip. 2017). ISSN: 0730-0301. DOI: 10.1145/3072959.3073608. URL: <https://doi.org/10.1145/3072959.3073608>.
- [134] Xiaolong Wang i in. “Non-local Neural Networks”. W: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2018, s. 7794–7803. DOI: 10.1109/CVPR.2018.00813.
- [135] Yue Wang i Justin M Solomon. “Object DGCNN: 3D Object Detection using Dynamic Graphs”. W: Advances in Neural Information Processing Systems. Red. M. Ranzato i in. T. 34. Curran Associates, Inc., 2021, s. 20745–20758. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/ade1d98c5ab2997e867b1151a5c5028d-Paper.pdf.
- [136] Yue Wang i in. “Dynamic Graph CNN for Learning on Point Clouds”. W: ACM Trans. Graph. 38.5 (paź. 2019). ISSN: 0730-0301. DOI: 10.1145/3326362. URL: <https://doi.org/10.1145/3326362>.

- [137] Ziming Wang i in. “Efficient point cloud representation learning with a recurrent hierarchical framework”. W: Applied Soft Computing 171 (2025), s. 112814. ISSN: 1568-4946. DOI: <https://doi.org/10.1016/j.asoc.2025.112814>. URL: <https://www.sciencedirect.com/science/article/pii/S1568494625001255>.
- [138] Mingqiang Wei i in. “AGConv: Adaptive Graph Convolution on 3D Point Clouds”. W: IEEE Transactions on Pattern Analysis & Machine Intelligence 45.08 (sierp. 2023), s. 9374–9392. ISSN: 1939-3539. DOI: [10.1109/TPAMI.2023.3238516](https://doi.org/10.1109/TPAMI.2023.3238516). URL: <https://doi.ieeecomputersociety.org/10.1109/TPAMI.2023.3238516>.
- [139] Cheng Wen i in. “PointWavelet: Learning in Spectral Domain for 3-D Point Cloud Analysis”. W: IEEE Transactions on Neural Networks and Learning Systems (2024), s. 1–13. DOI: [10.1109/TNNLS.2024.3363244](https://doi.org/10.1109/TNNLS.2024.3363244).
- [140] Christopher Williams i Matthias Seeger. “Using the Nyström Method to Speed Up Kernel Machines”. W: Advances in Neural Information Processing Systems. Red. T. Leen, T. Dietterich i V. Tresp. T. 13. MIT Press, 2000. URL: https://proceedings.neurips.cc/paper_files/paper/2000/file/19de10adbaa1b2ee13f77f679fa1483a-Paper.pdf.
- [141] Wenxuan Wu, Zhongang Qi i Li Fuxin. “PointConv: Deep Convolutional Networks on 3D Point Clouds”. W: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Los Alamitos, CA, USA: IEEE Computer Society, czer. 2019, s. 9613–9622. DOI: [10.1109/CVPR.2019.00985](https://doi.org/10.1109/CVPR.2019.00985). URL: <https://doi.ieeecomputersociety.org/10.1109/CVPR.2019.00985>.
- [142] Fei Xia. PointNet.pytorch. <https://github.com/fxia22/pointnet.pytorch>. 2018.
- [143] Tiange Xiang i in. “Walk in the Cloud: Learning Curves for Point Clouds Shape Analysis”. W: 2021 IEEE/CVF International Conference on Computer Vision (ICCV). 2021, s. 895–904. DOI: [10.1109/ICCV48922.2021.00095](https://doi.org/10.1109/ICCV48922.2021.00095).
- [144] Mutian Xu i in. “Learning Geometry-Disentangled Representation for Complementary Understanding of 3D Object Point Cloud”. W: Proceedings of the AAAI Conference on Artificial Intelligence 35.4 (maj 2021), s. 3056–3064. DOI: [10.1609/aaai.v35i4.16414](https://doi.org/10.1609/aaai.v35i4.16414). URL: <https://ojs.aaai.org/index.php/AAAI/article/view/16414>.
- [145] Yifan Xu i in. “SpiderCNN: Deep Learning on Point Sets with Parameterized Convolutional Filters”. W: Computer Vision – ECCV 2018: 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VIII. Munich, Germany: Springer-Verlag, 2018, 90–105. ISBN: 978-3-030-01236-6. DOI: [10.1007/978-3-030-01237-3_6](https://doi.org/10.1007/978-3-030-01237-3_6). URL: https://doi.org/10.1007/978-3-030-01237-3_6.
- [146] Xu Yan i in. “PointASNL: Robust Point Clouds Processing Using Nonlocal Neural Networks With Adaptive Sampling”. W: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2020, s. 5588–5597. DOI: [10.1109/CVPR42600.2020.00563](https://doi.org/10.1109/CVPR42600.2020.00563).
- [147] Jiancheng Yang i in. “Modeling Point Clouds With Self-Attention and Gumbel Subset Sampling”. W: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2019, s. 3318–3327. DOI: [10.1109/CVPR.2019.00344](https://doi.org/10.1109/CVPR.2019.00344).
- [148] Lidong Yang i Li Zhang. “Large-Workspace and High-Resolution Magnetic Micro-robot Navigation Using Global-Local Path Planning and Eye-in-Hand Visual Servoing”. W: 2020 IEEE 16th International Conference on Automation Science and Engineering (CASE). 2020, s. 876–881. DOI: [10.1109/CASE48305.2020.9216900](https://doi.org/10.1109/CASE48305.2020.9216900).

- [149] Tianwei Yin, Xingyi Zhou i Philipp Krähenbühl. “Center-based 3D Object Detection and Tracking”. W: 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2021, s. 11779–11788. DOI: 10.1109/CVPR46437.2021.01161.
- [150] Haoxuan You i in. “PVRNet: point-view relation neural network for 3D shape recognition”. W: Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence. AAAI’19/IAAI’19/EAAI’19. Honolulu, Hawaii, USA: AAAI Press, 2019. ISBN: 978-1-57735-809-1. DOI: 10.1609/aaai.v33i01.33019119. URL: <https://doi.org/10.1609/aaai.v33i01.33019119>.
- [151] Tan Yu, Jingjing Meng i Junsong Yuan. “Multi-view Harmonized Bilinear Network for 3D Object Recognition”. W: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2018, s. 186–194. DOI: 10.1109/CVPR.2018.00027.
- [152] Xumin Yu i in. “Point-BERT: Pre-training 3D Point Cloud Transformers with Masked Point Modeling”. W: 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2022, s. 19291–19300. DOI: 10.1109/CVPR52688.2022.01871.
- [153] Wei Zeng i Theo Gevers. “3DContextNet: K-d Tree Guided Hierarchical Learning of Point Clouds Using Local and Global Contextual Cues”. W: Computer Vision – ECCV 2018 Workshops: Munich, Germany, September 8-14, 2018, Proceedings, Part III. Munich, Germany: Springer-Verlag, 2019, 314–330. ISBN: 978-3-030-11014-7. DOI: 10.1007/978-3-030-11015-4_24. URL: https://doi.org/10.1007/978-3-030-11015-4_24.
- [154] Cheng Zhang i in. “PVT: Point-voxel transformer for point cloud learning”. W: Int. J. Intell. Syst. 37.12 (grud. 2022), 11985–12008. ISSN: 0884-8173. DOI: 10.1002/int.23073. URL: <https://doi.org/10.1002/int.23073>.
- [155] Gang Zhang i in. “HEDNet: A Hierarchical Encoder-Decoder Network for 3D Object Detection in Point Clouds”. W: Advances in Neural Information Processing Systems. Red. A. Oh i in. T. 36. Curran Associates, Inc., 2023, s. 53076–53089. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/a64e641fa00a7eb9500cb7e1835d0495-Paper-Conference.pdf.
- [156] Huang Zhang i in. “Deep learning-based 3D point cloud classification: A systematic survey and outlook”. W: Displays 79 (2023), s. 102456. ISSN: 0141-9382. DOI: <https://doi.org/10.1016/j.displa.2023.102456>.
- [157] Kuangen Zhang i in. “Linked Dynamic Graph CNN: Learning through Point Cloud by Linking Hierarchical Features”. W: 2021 27th International Conference on Mechatronics and Machine Vision in Practice (M2VIP). 2021, s. 7–12. DOI: 10.1109/M2VIP49856.2021.9665104.
- [158] Hengshuang Zhao i in. “PointWeb: Enhancing Local Neighborhood Features for Point Cloud Processing”. W: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2019, s. 5560–5568. DOI: 10.1109/CVPR.2019.00571.
- [159] Hengshuang Zhao i in. “Point Transformer”. W: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV). 2021, s. 16259–16268.

- [160] Xiao Zhao i in. “D-CONFORMER: Deformable Sparse Transformer Augmented Convolution for Voxel-Based 3D Object Detection”. W: ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). 2023, s. 1–5. DOI: 10.1109/ICASSP49357.2023.10097060.
- [161] Wu Zheng i in. “ SE-SSD: Self-Ensembling Single-Stage Object Detector From Point Cloud ”. W: 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Los Alamitos, CA, USA: IEEE Computer Society, 2021, s. 14489–14498. DOI: 10.1109/CVPR46437.2021.01426. URL: <https://doi.ieeecomputersociety.org/10.1109/CVPR46437.2021.01426>.
- [162] Junwen Zhong i in. “Deep Learning Based Strategy for Eye-to-Hand Robotic Tracking and Grabbing”. W: Neural Information Processing. Red. Haiqin Yang i in. Cham: Springer International Publishing, 2020, s. 787–798. ISBN: 978-3-030-63833-7.