POLITECHNIKA GDAŃSKA
WYDZIAŁ INŻYNIERII MECHANICZNEJ I OKRĘTOWNICTWA

Imię i nazwisko studenta: Agnieszka Osińska

Nr albumu: 192185

Poziom kształcenia: studia drugiego stopnia

Forma studiów: stacjonarne

Kierunek studiów: Mechatronika

Specjalność: projektowanie mechatroniczne

PRACA DYPLOMOWA MAGISTERSKA

Tytuł pracy w języku polskim: Sterowanie robotem współpracującym z wykorzystaniem gestów

Tytuł pracy w języku angielskim: Controlling a collaborative robot using gestures

Opiekun pracy: dr inż. Michał Mazur

POLITECHNIKA GDAŃSKA | WYDZIAŁ INŻYNIERII MECHANICZNEJ I OKRĘTOWNICTWA

OŚWIADCZENIE dotyczące pracy dyplomowej zatytułowanej: Sterowanie robotem współpracującym z wykorzystaniem gestów

Imię i nazwisko studenta: Agnieszka Osińska
Data i miejsce urodzenia: 04.02.1998, Toruń
Nr albumu: 192185

Wydział: Wydział Inżynierii Mechanicznej i Okrętownictwa
Kierunek: Mechatronika

Poziom kształcenia: studia drugiego stopnia
Forma studiów: stacjonarne

Typ pracy: praca dyplomowa magisterska

1. Oświadczenie dotyczące samodzielności pracy

Świadoma(y) odpowiedzialności karnej z tytułu naruszenia przepisów ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2022 r. poz. 2509, z późn. zm.) i konsekwencji dyscyplinarnych określonych w ustawie z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce (t.j. Dz.U. z 2024 r. poz. 1571, z późn. zm.),[1] a także odpowiedzialności cywilnoprawnej oświadczam, że przedkładana praca dyplomowa została opracowana przeze mnie samodzielnie.

Niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadaniem tytułu zawodowego.

Wszystkie informacje umieszczone w ww. pracy dyplomowej, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami zgodnie z art. 34 ustawy o prawie autorskim i prawach pokrewnych.

2. Oświadczenie o stosowaniu narzędzi GenAI

wykorzystywałam/wykorzystywałem narzędzia o potencjalnie niskim stopniu ingerencji

Oświadczam, że treści wygenerowane przy pomocy GenAI poddałam / poddałem krytycznej analizie i zweryfikowałam / zweryfikowałem.

11.06.2025, Agnieszka Osińska

Data i podpis lub uwierzytelnienie w portalu uczelnianym Moja PG

**) Dokument został sporządzony w systemie teleinformatycznym, na podstawie §15 ust. 3b Rozporządzenia MNiSW z dnia 12 maja 2020 r. zmieniającego rozporządzenie w sprawie studiów (Dz.U. z 2020 r. poz. 853). Nie wymaga podpisu ani stempla.*

---

[1] Ustawa z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce:

Art. 312. ust. 3. W przypadku podejrzenia popełnienia przez studenta czynu, o którym mowa w art. 287 ust. 2 pkt 1–5, rektor niezwłocznie poleca przeprowadzenie postępowania wyjaśniającego.

Art. 312. ust. 4. Jeżeli w wyniku postępowania wyjaśniającego zebrany materiał potwierdza popełnienie czynu, o którym mowa w ust. 5, rektor wstrzymuje postępowanie o nadanie tytułu zawodowego do czasu wydania orzeczenia przez komisję dyscyplinarną oraz składa zawiadomienie o podejrzeniu popełnienia przestępstwa.

STRESZCZENIE:

Celem niniejszej pracy magisterskiej było zaprojektowanie i implementacja systemu umożliwiającego sterowanie ramieniem robota współpracującego za pomocą gestów użytkownika. W ramach przeglądu literatury porównano istniejące rozwiązania, wybierając YOLOv8 jako narzędzie do detekcji punktów kluczowych ciała. System zintegrowano z ROS 2 Humble i MoveIt, tworząc węzeł analizujący pozycje dłoni, łokci i ramion. Na ich podstawie generowano trajektorie ruchu dwóch przegubów robota, zapewniając płynne sterowanie. System wyposażono w mechanizmy bezpieczeństwa i pomiar czasu reakcji. W ramach badań przetestowano dokładność rozpoznawania gestów oraz opóźnienia reakcji robota. Osiągnięto wysoką skuteczność i niskie opóźnienie, co potwierdza intuicyjność oraz niezawodność działania. System wykazał odporność na zakłócenia i stabilność w testowym środowisku. Opracowane rozwiązanie może być stosowane w aplikacjach przemysłowych i usługowych wymagających bezdotykowego sterowania robotem. Praca stanowi podstawę do dalszego rozwoju systemów interakcji człowiek–robot opartych na widzeniu komputerowym.

Słowa kluczowe:

HRI, interakcja człowiek–robot, YOLO, ROS 2, MoveIt, robot współpracujący, detekcja punktów kluczowych, głębokie uczenie, trajektoria przegubu.

Dziedzina nauki i techniki, zgodnie z wymogami OECD: Nauki inżynieryjne i techniczne, Elektrotechnika, elektronika, inżynieria informatyczna

ABSTRACT:

The purpose of this thesis was to design and implement a system that allows the control of a collaborative robotic arm using user gestures. A literature review compared existing solutions, selecting YOLOv8 as the tool for body keypoint detection. The system was integrated with ROS 2 Humble and MoveIt, creating a node that analyzes hand, elbow and shoulder positions. Based on these, motion trajectories were generated for the robot's two joints, ensuring smooth control. The system was equipped with safety mechanisms and reaction time measurement. As part of the research, the accuracy of gesture recognition and the robot's response latency were tested. High performance and low latency were achieved, confirming intuitive and reliable operation. The system demonstrated noise immunity and stability in the test environment. The developed solution can be used in industrial and service applications requiring non-contact robot control. The work provides a basis for further development of human-robot interaction systems based on computer vision.

Keywords:

HRI, human–robot interaction, YOLO, ROS 2, MoveIt, collaborative robot, keypoint detection, deep learning, joint trajectory.

Niniejszą pracę pragnę zadedykować moim Rodzicom - Veslavie oraz śp. Grzegorzowi Osińskim - dzięki którym miałam możliwość kształcić się i zdobywać cenną wiedzę.

Za cierpliwość, wyrozumiałość oraz pomoc przy realizacji tej pracy pragnę złożyć serdecznie podziękowania mojemu promotorowi dr inż. Michałowi Mazurowi.

Pragnę jeszcze osobno wyrazić wdzięczność moim przyjaciołom za wspieranie mnie w trudnych momentach.

SPIS TREŚCI

WYKAZ WAŻNIEJSZYCH OZNACZEŃ I SKRÓTÓW

ROS – Robot Operating System

RViz – Robot Operating System Visualization

HRI – Human Robot Interface

CNN – Convolutional Neural Network

YOLO – You Only Look Once

API – Application Programming Interface

VPL – Visual Programming Language

DDS – Data Distribution Service

APT – Advanced Package Tool

COCO – Common Objects in Context

1. CEL I ZAKRES PRACY

Celem niniejszej pracy magisterskiej było zaprojektowanie, implementacja oraz kompleksowa analiza systemu umożliwiającego sterowanie ramieniem robota współpracującego Hanwha HCR-3A przy pomocy gestów użytkownika, które są rejestrowane przez kamerę głębi Intel RealSense D435 i przetwarzane w środowisku ROS 2 oraz MoveIt. Główna idea projektu opierała się na stworzeniu naturalnego i intuicyjnego interfejsu człowiek–robot (HRI), który pozwalałby użytkownikowi na bezdotykowe kierowanie ruchami robota bez potrzeby użycia konwencjonalnych narzędzi sterujących. Inspiracją dla tej koncepcji były aktualne trendy w robotyce współpracującej, kładące nacisk na zwiększenie bezpieczeństwa, ergonomii oraz efektywności współdziałania człowieka z maszyną. System miał stanowić odpowiedź na potrzebę uproszczenia interakcji operator–robot, zwłaszcza w środowiskach, gdzie klasyczne formy sterowania nie są wygodne lub dostępne.

W ramach realizacji celu przeprowadzono szczegółowy przegląd literatury oraz dostępnych rozwiązań z zakresu rozpoznawania gestów, analizując zarówno klasyczne metody detekcji oparte na analizie obrazu, jak i nowoczesne techniki wykorzystujące głębokie sieci neuronowe. Przyjęto hipotezę, że możliwe jest stworzenie systemu, który umożliwi intuicyjne sterowanie robotem poprzez gesty, z wysoką skutecznością detekcji i niskim opóźnieniem reakcji. Cel ten został osiągnięty – opracowany system wykazał się skutecznością detekcji gestów na poziomie 100% oraz średnim opóźnieniem rzędu 0,36 s, co uznano za wynik w pełni akceptowalny w kontekście zastosowań przemysłowych i edukacyjnych. System był testowany w warunkach symulacyjnych, przy wykorzystaniu narzędzi takich jak RViz2 do wizualizacji oraz kontrolera JointTrajectoryController do sterowania przegubami robota.

Zrealizowanie celu pracy wymagało przezwyciężenia wielu wyzwań technicznych, w tym związanych z okablowaniem, stabilnością połączeń ROS 2, kompatybilnością bibliotek oraz koniecznością optymalizacji kodu pod kątem płynnego działania. Ostatecznie powstał kompletny, działający prototyp systemu, który może być rozwijany w kierunku pełnej integracji z innymi systemami HRI, a także rozszerzany o dodatkowe funkcje, takie jak rozpoznawanie twarzy operatora, wykrywanie stanu emocjonalnego, czy obsługa bardziej złożonych gestów dwuręcznych.

1.1. Zakres pracy

Niniejsza praca magisterska koncentruje się na zaprojektowaniu, wdrożeniu oraz analizie systemu sterowania robotem współpracującym Hanwha HCR-3A z wykorzystaniem gestów rozpoznawanych przez kamerę głębi Intel RealSense D435. Głównym celem jest stworzenie interfejsu pozwalającego na intuicyjną kontrolę ruchów robota bez użycia fizycznych kontrolerów czy interfejsów mechanicznych. Założono, że detekcja gestów odbywać się będzie na podstawie analizy obrazu z kamery w czasie rzeczywistym, a następnie dane te będą przekazywane do

środowiska robotycznego ROS 2 i MoveIt, co pozwoli na wizualizację oraz inicjację odpowiednich działań przez robota.

Zakres prac obejmuje pełen cykl projektowo-badawczy, rozpoczynający się od przeglądu literatury i istniejących rozwiązań w dziedzinie sterowania gestami w robotyce. W pracy analizowane są zarówno rozwiązania klasyczne, jak i te wykorzystujące metody głębokiego uczenia. W szczególności pod uwagę wzięto dwie biblioteki: trt_pose [1] od NVIDIA [2] [3] oraz YOLOv8 [4] od Ultralytics. Pierwsze z wymienionych rozwiązanie nie zostało ostatecznie zaimplementowane ze względu na liczne trudności z kompatybilnością ze środowiskiem ROS 2 i obsługą kamery RealSense D435. Zdecydowano się więc na implementację alternatywnego podejścia opartego na modelu YOLOv8, który okazał się bardziej elastyczny i możliwy do zintegrowania z wybranym środowiskiem.

Zakres działań obejmuje również konfigurację środowiska ROS 2 Humble, instalację i integrację MoveIt 2 oraz przygotowanie komunikacji między węzłami ROS a systemem detekcji gestów. Istotnym aspektem pracy była konfiguracja komunikacji między kamerą RealSense D435 a środowiskiem ROS, w tym publikacja danych wizualnych i głębi w formacie zgodnym z wymaganiami algorytmu YOLOv8m-pose. System detekcji gestów został oparty na modelu uczenia maszynowego, który rozpoznaje wytrenowane wcześniej punkty kluczowe, takie jak na przykład: dłoń, łokieć, bark, ucho czy nos.

Kolejnym etapem pracy była integracja sygnałów detekcji z systemem wizualizacji RViz2 oraz zdefiniowanie odpowiednich komunikatów ROS (topics), które umożliwiają transmisję informacji o rozpoznanym geście do komponentów odpowiedzialnych za planowanie ruchu robota. Opracowano mechanizm, w którym każdy gest odpowiada konkretnemu poleceniu ruchowemu, jak „podnieś”, „opuść”, „obróć w prawo”, „obróć w lewo”. Zaprojektowany system zakłada, że każdy z wykrytych gestów może uruchamiać konkretne działanie w środowisku ROS, w tym inicjację planowania trajektorii w MoveIt oraz aktywację odpowiednich komponentów wykonawczych ramienia robota.

W ramach pracy przeprowadzono szereg testów środowiskowych, których celem była weryfikacja skuteczności rozpoznawania gestów w warunkach rzeczywistych. Testowano działanie systemu przy różnym kącie ustawienia kamery oraz odległości operatora od obiektywu. Sprawdzono również opóźnienie między detekcją gestu a reakcją robota, co ma istotne znaczenie w kontekście użyteczności interfejsu sterowania gestami w środowiskach dynamicznych.

Istotnym ograniczeniem, jakie napotkano podczas realizacji projektu, była trudność w osiągnięciu pełnej integracji systemu detekcji z trajektorią ruchu robota. Ze względu na czasochłonność procesu uczenia i dostrajania algorytmów detekcji oraz konieczność dostosowania systemu komunikacji ROS do niestandardowych klas gestów, nie wszystkie założone cele udało się zrealizować w pełnym zakresie. Mimo to, uzyskane wyniki umożliwiły zbudowanie stabilnego systemu detekcji gestów i przesyłania informacji do ROS 2 oraz częściową integrację z systemem sterowania ruchem.

W pracy zrealizowano również zadania związane z dokumentacją architektury systemu, w tym przygotowanie graficznego schematu przepływu danych oraz opis interfejsów komunikacyjnych. Stworzono również dokumentację kodu i zaleceń dotyczących dalszego rozwoju systemu, co stanowi istotną wartość z punktu widzenia inżynierskiego.

Podsumowując, zakres niniejszej pracy magisterskiej obejmuje szerokie spektrum zagadnień od przetwarzania obrazu i detekcji gestów, przez komunikację systemów rozproszonych w ROS 2, aż po integrację i planowanie ruchu robota współpracującego Hanwha HCR-3A. Pomimo ograniczeń technicznych, wynikających głównie z trudności integracyjnych oraz czasochłonności procesów uczenia modeli, osiągnięto istotne rezultaty cząstkowe, które mogą posłużyć jako podstawa do dalszych badań i wdrożeń w zakresie naturalnego sterowania robotami. Praca pokazuje również potencjał, jaki niesie ze sobą integracja gestów z systemami robotycznymi oraz wyzwania, które należy rozwiązać w celu pełnego wykorzystania tego typu interfejsów w środowiskach przemysłowych i usługowych.

1.2. Uzasadnienie wyboru tematu

Temat pracy dyplomowej został wybrany ze względu na rosnące znaczenie robotów współpracujących w nowoczesnych środowiskach przemysłowych i produkcyjnych. Współczesna automatyzacja coraz częściej stawia na integrację maszyn z ludźmi, co wymaga opracowania intuicyjnych i efektywnych metod komunikacji między człowiekiem a robotem. Robot HCR3A, będący na wyposażeniu naszej uczelni, stanowi doskonałą platformę badawczą do eksploracji takich rozwiązań, ponieważ jest jednym z najbardziej zaawansowanych i elastycznych robotów współpracujących dostępnych na rynku. Dzięki temu możliwe jest bezpośrednie przeprowadzanie eksperymentów i testów w realnych warunkach laboratoryjnych, co podnosi wartość merytoryczną i praktyczną całej pracy. Roboty współpracujące charakteryzują się nie tylko wysokim poziomem bezpieczeństwa, ale również możliwością pracy w bezpośrednim sąsiedztwie człowieka bez konieczności stosowania tradycyjnych barier ochronnych. To właśnie czyni je kluczowymi urządzeniami przyszłości, zdolnymi do zwiększenia wydajności i jakości procesów produkcyjnych, przy jednoczesnym zmniejszeniu ryzyka wypadków i przeciążeń pracowników.

Tematyka pracy łączy w sobie wiele aktualnych zagadnień z dziedziny automatyki i robotyki: przetwarzanie obrazu, głębokie uczenie, komunikację w systemach rozproszonych oraz planowanie ruchu manipulatorów. Dodatkowym uzasadnieniem wyboru tego tematu była chęć rozwiązania rzeczywistego problemu technicznego, jakim jest opracowanie systemu interakcji, który mógłby zostać wykorzystany w środowisku laboratoryjnym lub przemysłowym. Praca ma charakter aplikacyjny, ale jednocześnie umożliwia pogłębienie wiedzy teoretycznej z zakresu detekcji gestów i przetwarzania sygnałów w czasie.

Sterowanie robotem za pomocą gestów jest innowacyjną metodą interakcji, która znacząco upraszcza obsługę urządzenia, eliminując potrzebę stosowania skomplikowanych paneli sterujących czy komend głosowych, które mogą być zawodne w hałaśliwym środowisku przemysłowym. Gesty są naturalnym i uniwersalnym sposobem komunikacji, który pozwala na

szybkie i precyzyjne wydawanie poleceń robotowi, co wpływa na efektywność pracy i komfort operatora.

Ponadto, rozwój technologii rozpoznawania gestów wpisuje się w szerszy trend digitalizacji i Przemysł 5.0, gdzie kluczową rolę odgrywają inteligentne systemy autonomiczne, zdolne do adaptacji i współpracy z człowiekiem. Praca nad sterowaniem HCR3A z wykorzystaniem gestów daje możliwość pogłębienia wiedzy z zakresu robotyki, sztucznej inteligencji oraz interakcji człowiek-maszyna, co ma istotne znaczenie zarówno z punktu widzenia naukowego, jak i przemysłowego. Realizacja tego tematu pozwoli również na rozwój kompetencji praktycznych w obszarze programowania robotów współpracujących oraz integracji różnych sensorów i systemów sterowania.

Wreszcie, tematyka sterowania robotem HCR3A za pomocą gestów wpisuje się w aktualne kierunki badań i rozwoju technologii w zakresie robotyki współpracującej, co zwiększa szanse na publikację wyników pracy oraz nawiązanie współpracy z przemysłem. Rozwijanie tego typu rozwiązań ma potencjał do realnego wpływu na transformację przemysłu, czyniąc go bardziej elastycznym, bezpiecznym i przyjaznym dla pracowników. Wszystkie te aspekty sprawiają, że wybrany temat jest nie tylko aktualny i innowacyjny, ale także praktycznie użyteczny i zgodny z wizją przyszłości robotyki oraz automatyzacji.

2. PRZEGLĄD LITERATURY I TECHNOLOGII

Niniejszy rozdział dokładnie opisuje charakterystykę robotów współpracujących, ich genezę, kierunki rozwoju oraz szeroki wachlarz zastosowań w różnych gałęziach przemysłu. Szczególną uwagę poświęcono również technikom sterowania gestami, które stanowią nowoczesny i intuicyjny sposób komunikacji człowieka z robotem, znacząco zwiększający efektywność pracy i ergonomię obsługi. Omówienie tych zagadnień stanowi niezbędne tło teoretyczne dla zrozumienia rozwiązań zastosowanych w dalszej części pracy, dotyczącej implementacji sterowania gestami w robocie współpracującym HCR3A.

2.1. Charakterystyka robotów współpracujących

Roboty współpracujące, znane również jako coboty (ang. collaborative robots), to zaawansowane maszyny zaprojektowane do pracy w bezpośrednim sąsiedztwie ludzi. W przeciwieństwie do tradycyjnych robotów przemysłowych, które zwykle działają w odizolowanych przestrzeniach, roboty charakteryzują się zwiększonym bezpieczeństwem, elastycznością i możliwością adaptacji do zmieniających się warunków pracy. Są szeroko stosowane w różnych sektorach przemysłu, takich jak produkcja, logistyka, medycyna czy rolnictwo. Ich rola polega na uzupełnianiu pracy człowieka poprzez wykonywanie precyzyjnych, powtarzalnych lub wymagających fizycznie zadań, podczas gdy człowiek może skoncentrować się na bardziej kreatywnych lub decyzyjnych aspektach pracy [5].

Według raportu Międzynarodowej Federacji Robotyki z 2002 [6], roboty współpracujące stanowią jeden z najszybciej rozwijających się segmentów robotyki przemysłowej. Ich globalna adaptacja wynika z takich cech jak intuicyjność obsługi, zaawansowane systemy bezpieczeństwa oraz możliwość szybkiego dostosowania do zróżnicowanych zadań. W 2003 roku stanowiły 10,5% robotów przemysłowych zainstalowanych na całym świecie [7].

Bezpieczeństwo jest jedną z najważniejszych cech robotów współpracujących, dzięki czemu mogą one pracować w bliskim kontakcie z ludźmi. Wykorzystują czujniki siły i momentu obrotowego do monitorowania nacisku i momentu obrotowego, aby wykryć nieoczekiwane kolizje i zatrzymać pracę w przypadku zagrożenia. Ich systemy wizyjne skanują otoczenie i identyfikują potencjalne przeszkody, umożliwiając robotom unikanie kolizji [8]. Dodatkowo oprogramowanie reagujące szybko i dokładnie. Analizuje dane w celu szybkiego dostosowania pracy robota, np. redukcji prędkości lub zatrzymania w razie wykrycia człowieka w strefie działania. Dzięki temu są w stanie dostosowywać się do różnych zadań bez potrzeby kosztownych modyfikacji sprzętowych. Roboty są znacznie bardziej elastyczne niż tradycyjne roboty przemysłowe. Można je łatwo przystosować do wykonywania różnych zadań poprzez zmianę osprzętu, oprogramowania lub parametrów działania. Ich kompaktowa budowa i niska waga umożliwiają także szybkie przenoszenie między stanowiskami pracy.

Roboty współpracujące są projektowane również z myślą o intuicyjnej obsłudze. Ten efekt uzyskuje się poprzez stosowanie programowania graficznego lub stosując technikę uczenia przez przekaz. Oprogramowanie z interfejsem graficznym umożliwia łatwe ustawianie zadań za

pomocą przeciągania i upuszczania elementów. W tej drugiej technice operator może fizycznie manipulować ramieniem robota, a robot zapamiętuje wykonywane ruchy [9]. Ze względu na kompaktową budowę i lekką konstrukcję, roboty mogą być szybko przenoszone między stanowiskami pracy, co pozwala na ich adaptację do zmieniających się wymagań produkcyjnych. Cechy te sprawiają, że znajdują zastosowanie zarówno w dużych fabrykach, jak i małych przedsiębiorstwach [10].

2.1.1.Geneza

W latach 80 – tych ubiegłego wieku firma General Motors poszukiwała rozwiązania problemu zbyt obciążających zadań pracowników. Plan zakładał stworzenie robotów, które pomagałyby pracownikom z cięższymi elementami do montażu jednocześnie nie stwarzając dla niż żadnego zagrożenia. Do tego zadania została wyznaczona dwójka inżynierów mechanicznych: profesor Michael Peshkin i J. Edward Colgate [11] [12]. Wpadli na pomysł współpracy robota z człowiekiem, gdzie każda strona skupiałaby się na tym co robi najlepiej. Początkowa interakcja mogła wyglądać na przykład tak, że robot utrzymuje ładunek ale nie wprawia go w ruch, to zapewnia z kolei pracownik.

Zdjęcie 1 Absolwent Witaya Wannasuphoprasit z prototypem trójkołowego robota, zdolnego do obrony wirtualnych powierzchni w trzech wymiarach (x, y i y). Wirtualna powierzchnia (na ekranie) odpowiada krawędzi stołu [12] [11]

Początkowo naukowcy nazywali swój wynalazek programowalną maszyną ograniczającą (ang. Programmable constraint machine) (Zdjęcie 1) ze względu na powierzchni ograniczające które tworzył robot. Jednak wraz kolejnymi badaniami zauważono, że bliska interakcja maszyny i człowieka jest kluczowa dla dalszego rozwoju pomysłu. Wprowadzono wtedy termin cobot czyli robot współpracujący, a wynalazek opatentowano w 1999 roku [11]. Od tamtego czasu rodzina robotów współpracujących powiększa się, zwiększa swoje możliwości oraz zakres specjalizacji. Poza funkcją podnoszenia oraz przemieszczania przedmiotów roboty są użyteczne przy takich zadaniach jak: spawanie, wkręcanie, sortowanie, dopasowywanie czy łączenie [10]. Wiele modeli ramion posiada wymienne końcówki chwytaka by jak najlepiej dopasować robota do wyznaczonego zadania. Są również tworzone odpowiednie środowiska programistyczne które ułatwiają pracownikom ich obsługę.

2.1.2. Porównanie z robotami przemysłowymi

Z samego opisu zastosowania robotów współpracujących ciężko jest je odróżnić od powszechnie znanych robotów przemysłowych. Różnice są jednak spore. Roboty potrafią zatrzymać swoją pracę w przypadku wkroczenia człowieka w ich pole działania. Po odejściu pracownika robot kontynuuje przerwane zadanie. To kluczowa różnica która zapewnia bezpieczeństwo człowieka i umożliwia jego współpracę z maszyną. Roboty współpracujące potrafią uczyć się na przykładach. Najczęściej jest to widoczne w trakcie wykonywania zadań z dobieraniem i umieszczaniem elementów w zadanych miejscach. Na podstawie kilku sekwencji pokazanych przez pracownika robot może dalej powtarzać tę czynność [9]. W przypadku zadań wymagających częstszej interwencji człowieka roboty mogą mieć być zaprogramowane na zmniejszanie prędkości w miarę zbliżania się operatora. Aż do pełnego zatrzymania opisanego wcześniej. W porównaniu z tradycyjnymi robotami przemysłowymi, które wymagają kosztownych inwestycji w infrastrukturę i ochronę, roboty są relatywnie tańsze w instalacji. Dzięki możliwości współdzielenia przestrzeni roboczej z człowiekiem, wyeliminowano konieczność stosowania dodatkowych barier ochronnych [11]. Modele robotów, takie jak seria UR firmy Universal Robot [10] czy YuMi firmy ABB [13], często wymagają niewielkiej lub żadnej technologii bezpieczeństwa ze względu na wbudowane funkcje ograniczające moc i siłę. Roboty współpracujące o ograniczonej sile mogą odczytywać siły w swoich stawach, takie jak ciśnienie, opór lub zderzenia, za pomocą wbudowanych czujników. Po wyczuciu zakłócenia robot zatrzyma się lub zmieni kurs. Ich miękkie powłoki rozpraszają możliwą siłę uderzenia. Ponadto stosując bezpośrednie napędy w przegubach łatwiej jest monitorować występujących w nich momenty i siły, a to jest kluczowe z wcześniej wspomnianych względów bezpieczeństwa i wykrywania kolizji. Prowadzenie wewnętrzne kabli jest bezpieczne i unika się dodatkowych problemów, szczególnie w przypadku wykonywania przez robota skomplikowanych trajektorii.

2.1.3.Budowa

Rodzina robotów współpracujących dąży do wykorzystania zaawansowanych technologii, takich jak systemy wizyjne, czujniki siły, algorytmy sztucznej inteligencji oraz robotyka miękka [14] [15]. Miękka Robotyka (ang. Soft Robotics) to specyficzne podłoże robotyki

zajmujące się konstruowaniem robotów z materiałów podatnych na rozciąganie, na podobieństwo tych, które można znaleźć u organizmów żywych [16].

Systemy wizyjne i sensoryczne odgrywają kluczową rolę w zapewnieniu zdolności percepcyjnych robotów. Dzięki nim roboty mogą wykrywać obiekty w środowisku, analizować otoczenie oraz dostosowywać swoje działania do zmieniających się warunków. Dokładnie mapują otoczenie za pomocą kamer RGB-D, które łączą obraz kolorowy z danymi głębi, umożliwiając tworzenie szczegółowego modelu 3D otoczenia [17] [15]. Są one szeroko stosowane w procesach montażu [18] [19], takich jak dokładne pozycjonowanie komponentów [20]. Dodatkowo analiza obrazu z uczeniem maszynowym, pozwala robotom identyfikować różne typy obiektów [21], ich właściwości [22] oraz podejmować decyzje, np. wybór odpowiedniego miejsca chwytu.

Czujniki siły zainstalowane w przegubach najnowszych robotów są odpowiedzialne za monitorowanie obciążenia mechanicznego i wczesne wykrywanie kolizji. Regulują siły chwytania przy manipulacji delikatnymi obiektami. Na przykład w chirurgii robot współpracujący KUKA LBR iiwa [23] wykorzystuje wbudowane czujniki do wykonywania precyzyjnych manipulacji w czasie rzeczywistym, co czyni go idealnym do procedur neurochirurgicznych.

Dzięki robotyce miękkiej, wspomnianej wcześniej, roboty mogą wykonywać zadania wrażliwe na siłę, jak również bezpiecznie współpracować z ludźmi. Miękkie chwytaki, wykonane z materiałów takich jak silikon lub elastomer, znajdują zastosowanie w delikatnych operacjach, takich jak zbieranie owoców i warzyw, np. robot Agrobot SW6010 [24], który dzięki miękkim chwytakom może zbierać truskawki bez ich uszkodzenia. Są również wykorzystywane w rehabilitacji gdzie umożliwiają pacjentom powtarzalne wykonywanie ruchów w sposób bezpieczny i wspierany mechanicznie [23].

2.1.4.Zastosowanie

Roboty współpracujące mają niskie zapotrzebowanie na energię i często są mniejszych rozmiarów niż roboty przemysłowe, a dzięki wykrywaniu kolizji zapewniają bezpieczeństwo swoim współpracownikom i innym robotom. Dlatego znajdują zastosowanie w różnych gałęziach przemysłu, począwszy od produkcji, poprzez logistykę i rolnictwo, aż po medycynę. Poniżej opisuję przykłady konkretnych robotów wybranych z wyróżnionych sekcji przemysłu.

Przemysł jest najważniejszym odbiorcą technologii, a zastosowania robotów współpracujących w tej branży obejmują montaż, obsługę maszyn i pakowanie produktów.

Zdjęcie 2 Kobot UR10 firmy Universal Robots [25]

Robot UR10 (Zdjęcie 2) firmy Universal Robots [25] to uniwersalne rozwiązanie przemysłowe, szczególnie popularne w środowiskach produkcyjnych. Jego kluczowe cechy obejmują:

- Udźwig do 10 kg oraz zasięg 1300 mm, co czyni go idealnym do obsługi cięższych komponentów.
- Łatwość instalacji i programowania, co pozwala na szybkie wdrożenie w niemal dowolnym zakładzie produkcyjnym.
- Elastyczność w dostosowywaniu osprzętu, co umożliwia wykonywanie różnorodnych zadań, takich jak wkręcanie śrub, zgrzewanie czy montaż podzespołów.

Zdjęcie 3 Robot dwuramienny YuMi firmy ABB [26]

Model YuMi (Zdjęcie 3) jest dwuramiennym robotem [26] zaprojektowanym do delikatnych zadań, takich jak montaż drobnych komponentów w branży elektronicznej. Robot ten, dzięki wbudowanym czujnikom w ramionach, umożliwia precyzyjne dopasowanie ruchów do

montowanych elementów. Przykładem zastosowania jest montaż zegarków, telefonów komórkowych czy innych urządzeń wymagających wyjątkowej dokładności.

Rolnictwo, jako dynamicznie rozwijająca się branża technologiczna, coraz częściej korzysta z robotów, które wspomagają zbieranie plonów, kontrolę jakości upraw i zarządzanie dużymi gospodarstwami.

Zdjęcie 4 LettuceBot firmy Blue River [27]

Robot LettuceBot 2 (Zdjęcie 4) firmy Blue River [27] jest używany do pielęgnacji i monitorowania plantacji sałaty. Wyposażony w zaawansowane systemy wizyjne oraz sztuczną inteligencję, LettuceBot może:

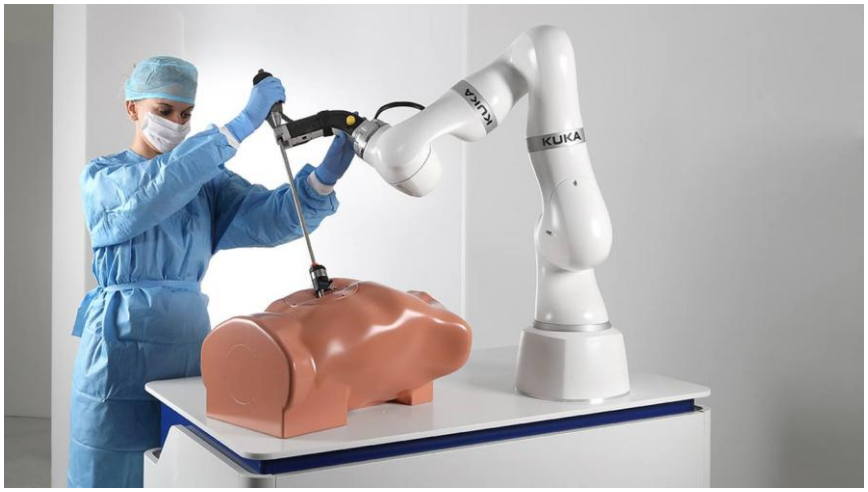
- Identyfikować chwasty i precyzyjnie aplikować środki ochrony roślin.
- Analizować kondycję upraw, umożliwiając optymalizację procesów rolniczych.
- Automatyzować złożone i czasochłonne czynności, zwiększając plony o nawet 15%.

Zdjęcie 5 Robot SW 6010 firmy Agrobot [24]

Agrobot SW6010 (Zdjęcie 5) [24] to robot specjalnie zaprojektowany do zbioru owoców miękkich, takich jak truskawki. Kluczowe funkcje tego urządzenia obejmują:

- Zaawansowane kamery i algorytmy analizy obrazu do oceny dojrzałości owoców.
- System chwytaków miękkich, które umożliwiają delikatny zbiór owoców bez ich uszkodzenia.
- Integrację z systemami IoT, co pozwala na analizę danych o uprawach w czasie rzeczywistym.

W medycynie roboty współpracujące znajdują zastosowanie w różnych obszarach, takich jak chirurgia, rehabilitacja czy logistyka szpitalna. Ich precyzja, bezpieczeństwo i możliwość pracy w sterylnym środowisku czynią je niezastąpionymi narzędziami wspomagającymi personel medyczny.



Zdjęcie 6 Robot LBR iiwa firmy KUKA [23]

Jednym z najbardziej zaawansowanych robotów stosowanych w medycynie jest model KUKA LBR iiwa (Zdjęcie 6) (niem. Leichtbauroboter, ang. intelligent industrial work assistant) [23]. Zastosowany w chirurgii i terapii rehabilitacyjnej, robot ten wyróżnia się:

- Czujnikami momentu w każdej osi, które umożliwiają precyzyjne manipulacje przy minimalnym nacisku na pacjenta.
- Możliwością wykonywania delikatnych czynności, takich jak asystowanie w operacjach neurochirurgicznych czy implantacji protez.
- Sterowaniem w czasie rzeczywistym, co pozwala na współpracę z chirurgiem podczas procedur wymagających dużej dokładności.

W nadchodzących latach roboty współpracujące staną się bardziej precyzyjne i wszechstronne. W miarę jak sztuczna inteligencja będzie coraz lepsza, wzrastać będzie także wykonywanie precyzyjnych i poznawczych zadań podejmowanych przez roboty. Co więcej, łączność robotów z IoT, czyli połączenie z innymi maszynami, urządzeniami, sieciowymi bazami

danych itp., pozwoli im usprawnić przepływ pracy w wielu fabrykach, a nawet zapewnić cenne analizy danych. Kiedy ludzie i roboty współpracujące zaczną wspólnie pracować nad bardziej złożonymi zadaniami, komunikacja stanie się konieczna. Można zbudować roboty współpracujące, które będą rozpoznawać określone polecenia głosowe lub gesty dłoni, co pozwoli uniknąć konieczności przeprogramowywania i powodowania przestojów w trakcie realizacji projektu. W miarę rozwoju robotów współpracujących i szukania nowych zastosowań w coraz większej liczbie branż, prawdopodobnie będą one napotykać coraz bardziej nieustrukturyzowane środowiska. Przyszłe maszyny będą mogły wykonywać nieograniczone ruchy w celu wykonania zadania. W ten sposób robot nie byłby ograniczony do tych samych, powtarzalnych ruchów, gdy zmieniało się otoczenie wokół niego. Zamiast tego mógłby odpowiednio zareagować i mimo to dokończyć zamierzone zadanie. W przyszłości roboty współpracujące będą mogły wykrywać ludzkie zachowania w kontekście większego projektu i odpowiednio dostosowywać swoje zachowanie, aby zmaksymalizować produktywność. Chociaż część tej technologii może wydawać się odległa od rzeczywistości, roboty współpracujące przyszłości będą znacznie bardziej zaawansowane i zdolne do osiągnięcia wyższego poziomu interaktywności i współpracy część jest już realizowana w projekcie Helix [28].

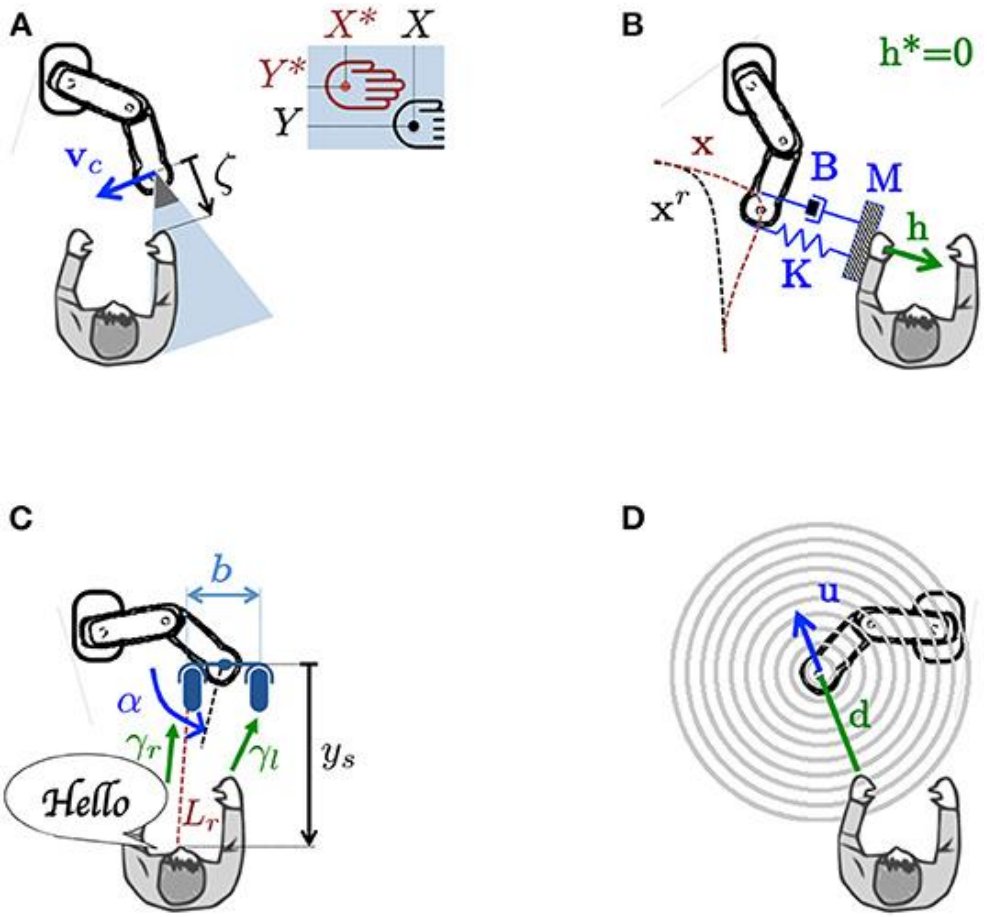
2.1.5.Sterowanie

Ramiona mogą być wyposażone w kamery, które rejestrują gesty operatora i za pomocą analizy obrazu wykonują wyuczone wcześniej polecenie. Mogą posiadać również mikrofony pozwalające na rozpoznawanie głosu i podanych komend. Operatorzy mogą również nosić na sobie sensory inercyjne lub inne czujniki które pomagają urządzeniu w obserwacji ciała i gestów.

W artykule „Sensor-Based Control for Collaborative Robots: Fundamentals, Challenges, and Opportunities” [29] przedstawiony jest przegląd metod sterowania robotami współpracującymi opartych na różnych sensorach. Celem tych metod jest umożliwienie bezpiecznej i efektywnej interakcji człowiek - robot, zarówno w kontekście fizycznej współpracy, jak i koegzystencji w tym samym środowisku.

Autorzy wyróżniają cztery główne podejścia do sterowania opartego na czujnikach (zobrazowane na Rysunek 1). Sterowanie wizyjne (Rysunek 1 (A)) wykorzystuje dane z kamer do śledzenia ruchów ludzi, rozpoznawania gestów i planowania trajektorii robota w czasie rzeczywistym. Systemy te mogą być skonfigurowane jako „eye-in-hand” (kamera zamontowana na robocie) lub „eye-to-hand” (kamera stacjonarna). Sterowanie wizyjne jest fundamentalne dla zrozumienia środowiska i intencji człowieka, co pozwala robotowi odpowiednio reagować. Sterowanie dotykowe (Rysunek 1 (B)) opiera się na czujnikach siły i momentu obrotowego, umożliwiając robotowi wyczuwanie kontaktu fizycznego i dostosowywanie siły interakcji. Strategie te dzielą się na bezpośrednie (regulacja siły kontaktu do zadanej wartości) i pośrednie (np. sterowanie impedancyjne), które modelują interakcję jako system masa-sprężyna-tłumik. Sterowanie akustyczne (Rysunek 1 (C)) wykorzystuje mikrofony do lokalizacji źródeł dźwięku, takich jak głos ludzki, i kierowania robota w ich stronę. Techniki takie jak różnica czasu przybycia (ITD) i różnica poziomu dźwięku (ILD) są stosowane do określenia kierunku źródła dźwięku.

Sterowanie akustyczne jest szczególnie przydatne w aplikacjach bezkontaktowych, wzbogacając interakcję człowiek–robot. Ostatnim typem jest sterowanie oparte na odległości (Rysunek 1 (D)), które używa czujników zbliżeniowych do wykrywania obecności człowieka i unikania kolizji. Dzięki rozwojowi kamer 3D, takich jak Microsoft Kinect czy Intel RealSense, możliwe jest jednoczesne wykorzystanie danych wizyjnych i odległościowych.



Rysunek 1 Przykłady czterech typów sterowania opartych na czujnikach. (A) Sterowanie wizyjne. (B) Sterowanie dotykowe. (C) Sterowanie akustyczne. (D) Sterowanie na odległość.

Integracja danych z różnych czujników jest kluczowa dla uzyskania pełniejszego obrazu sytuacji i bardziej naturalnej interakcji z kobotem. Metody takie jak sterowanie współdzielone (shared control) czy sterowanie naprzemienne (traded control) pozwalają na dynamiczne przełączanie między różnymi modalnościami sensorycznymi w zależności od kontekstu zadania. Na przykład, w aplikacjach montażu przemysłowego, wizja może być używana do nawigacji, podczas gdy czujniki siły monitorują kontakt z człowiekiem [30]. Autorzy podkreślają, że chociaż wizja i dotyk są obecnie najczęściej stosowanymi metodami komunikacji z robotami współpracujących, rozwój tanich i precyzyjnych czujników dotykowych, odległościowych i akustycznych stwarza nowe możliwości. W szczególności, rozwój „skórek” robotycznych i czujników zbliżeniowych może znacznie poprawić zdolność robotów do percepcji otoczenia i interakcji z ludźmi. Jednakże, istniejące wyzwania, takie jak opóźnienia w przetwarzaniu danych, złożoność integracji różnych czujników oraz zapewnienie bezpieczeństwa interakcji, wymagają dalszych badań i rozwoju.

2.2. Sterowanie gestami

Sterowanie gestami umożliwia naturalną i intuicyjną kontrolę nad robotem bez potrzeby stosowania fizycznych interfejsów. Rozwój technologii głębokiego uczenia i widzenia komputerowego przyczynił się do znacznego postępu w tej dziedzinie [31]. Podejścia do rozpoznawania gestów:

a) Modele oparte na splotowych sieciach neuronowych (CNN) – skuteczne w analizie obrazów i wykrywaniu kluczowych punktów [32].
b) Detekcja pozycji (pose estimation) – algorytmy takie jak OpenPose wykrywają kluczowe punkty ciała [33].
c) YOLO (You Only Look Once) – szybka metoda detekcji obiektów z wysoką skutecznością oparta na CNN ale wyróżniająca się dokładnymi wynikami już po jednym sygnale (2.2.2Algorytmy detekcji i rozpoznawania gestów) [34].

Według artykułu z czasopisma „Complex & Intelligent Systems” [35] sterowanie gestami w robotyce znajduje zastosowanie w medycynie, przemyśle i rozrywce. Szczególną uwagę zwraca się na precyzję rozpoznawania gestów oraz czas reakcji systemu. Do zalet sterowania gestami można zaliczyć naturalną interakcję, brak potrzeby dodatkowych urządzeń sterujących czy możliwość zastosowania w dynamicznych środowiskach.

Pomimo potencjału, jaki niesie ze sobą taka forma sterowania, istnieje szereg problemów technicznych i praktycznych, które utrudniają jej niezawodne wdrożenie. Przede wszystkim skuteczność systemów wizyjnych może być silnie uzależniona od warunków oświetleniowych – zbyt jasne lub zbyt ciemne otoczenie może zakłócać jakość przechwytywanego obrazu. Dodatkowo, różnorodność w sposobie wykonywania gestów przez różnych operatorów (np. różne rozmiary dłoni, szybkość ruchu, indywidualna ekspresja) stanowi wyzwanie dla algorytmów klasyfikacyjnych, które muszą być odpowiednio przeszkolone na szerokim zestawie danych. Problematyczna może być również jakość obrazu – niska rozdzielczość, zakłócenia czy obecność tła o wysokiej złożoności mogą znacząco obniżyć skuteczność wykrywania i rozpoznawania gestów. W związku z tym, niezbędne jest nie tylko zapewnienie odpowiednich warunków środowiskowych, lecz także ciągłe doskonalenie algorytmów oraz odpowiedni dobór sprzętu i strategii uczenia maszynowego.

2.2.1.Mechanizm detekcji

Detekcja pozycji (ang. pose estimation) jest kluczowym procesem w systemach rozpoznawania gestów, polegającym na precyzyjnym określeniu lokalizacji charakterystycznych punktów ciała, takich jak dłonie, łokcie czy barki, na podstawie obrazu lub sekwencji wideo. Proces ten składa się z kilku istotnych etapów, z których każdy wpływa na końcową skuteczność rozpoznania gestu. Pierwszym krokiem jest przetwarzanie obrazu, w którym kamera rejestruje dane w postaci pojedynczych klatek w formacie RGB lub informacji o głębi. Następnie algorytm

detekcji obiektów analizuje uzyskany obraz w celu identyfikacji elementów istotnych z punktu widzenia zadania, na przykład dłoni użytkownika. Kolejnym etapem jest ekstrakcja kluczowych punktów, podczas którego lokalizowane są szczegółowe części anatomiczne, takie jak końcówki palców, nadgarstek czy zgięcia stawów. Te punkty tworzą szkielet reprezentujący pozycję ciała lub jego fragmentów. Ich rozmieszczenie w przestrzeni stanowi podstawę do dalszej analizy, prowadzącej do interpretacji wykonanego gestu. Na tym etapie wykorzystywane są modele matematyczne lub sieci neuronowe, które przetwarzają układ punktów w celu rozpoznania określonego gestu. Efektywność całego systemu zależy nie tylko od dokładności wykrycia punktów, ale również od zdolności do interpretacji ich wzajemnych relacji w przestrzeni. Wysoka precyzja detekcji pozycji jest zatem kluczowa dla niezawodnego działania systemów interakcji człowiek–maszyna, zwłaszcza w kontekście sterowania gestami. Dzięki niej możliwe jest tworzenie naturalnych i intuicyjnych interfejsów użytkownika.

2.2.2. Algorytmy detekcji i rozpoznawania gestów

W literaturze naukowej i praktyce inżynierskiej istnieje szereg algorytmów wykorzystywanych do detekcji i rozpoznawania gestów. Należą do nich zarówno klasyczne metody widzenia komputerowego, jak i nowoczesne modele głębokiego uczenia. Do najbardziej znanych i szeroko stosowanych rozwiązań należą: OpenPose (2018), MediaPipe (2019), YOLO (od 2016 w kolejnych wersjach), DensePose (2018), PoseNet (2017) oraz trt_pose (2019).

OpenPose to jeden z pierwszych otwartych systemów umożliwiających detekcję pozycji w czasie rzeczywistym dla całego ciała. Działa na podstawie wykrywania szkieletu ciała i punktów charakterystycznych. Jest ceniony za dokładność, lecz jego największym ograniczeniem jest wydajność – wymaga dużej mocy obliczeniowej [33].

MediaPipe to rozwiązanie od Google pozwalające na szybkie i efektywne wykrywanie dłoni i twarzy. Wyróżnia się bardzo wysoką szybkością działania i możliwością uruchomienia na urządzeniach mobilnych, jednak oferuje ograniczone możliwości adaptacji do niestandardowych gestów [36].

YOLO to metoda bazująca na jednoprzebiegowej analizie obrazu. Wersje od YOLOv3 w górę zapewniają znaczący wzrost dokładności i efektywności. YOLOv8m-pose, bazujące na tej architekturze, zostało wykorzystane w niniejszej pracy z uwagi na dobrą równowagę pomiędzy dokładnością a szybkością działania [34] [37].

DensePose przenosi analizę poza dwuwymiarową projekcję punktów, pozwalając na odwzorowanie pozycji ciała w przestrzeni 3D. Jest to rozwiązanie bardziej zasobożerne, ale potencjalnie dokładniejsze w analizie ruchu całego ciała [38].

PoseNet to lekka sieć neuronowa, wykorzystywana głównie w przeglądarkach i aplikacjach webowych. Dobrze sprawdza się w prostych przypadkach, jednak jest mniej dokładna niż OpenPose czy MediaPipe [39].

trt_pose to zoptymalizowana wersja detekcji pozycji dla platform Jetson. W teorii zapewnia wysoką wydajność, ale jej użycie okazało się trudne ze względu na ograniczenia kompatybilności.

Tabela 2.1 Porównanie wybranych algorytmów detekcji i rozpoznawania gestów (opracowanie własne)

Algorytm	Rok	Dokładność	Szybkość działania	Wymagania sprzętowe	Zastosowania	Uwagi
OpenPose	2018	Wysoka	Niska	Wysokie	Analiza całego ciała	Wysoka dokładność, ale duże zużycie zasobów
MediaPipe	2019	Średnia	Bardzo wysoka	Niskie	Dłonie, twarz	Mobilność i wydajność, ograniczona adaptacja
YOLOv8m-pose	2023	Wysoka	Wysoka	Średnie	Rozpoznawanie pozycji ciała	Dobry kompromis między szybkością a dokładnością
DensePose	2018	Bardzo wysoka	Niska	Bardzo wysokie	Analiza 3D ciała	Precyzyjna, ale wymagająca zasobów
PoseNet	2017	Niska-średnia	Średnia	Niskie	Aplikacje webowe	Lekka, ale mniej precyzyjna
trt_pose	2019	Średnia	Bardzo wysoka	Niskie (Jetson)	Jetson, wbudowane systemy	Trudna integracja, brak wsparcia dla ROS 2

Tabela 2.1 zawiera analizę porównawczą wymienionych algorytmów detekcji pozycji gdzie kluczowymi kryteriami oceny były: dokładność detekcji, szybkość działania, wymagania sprzętowe oraz typowe obszary zastosowania. Algorytmy takie jak OpenPose czy DensePose oferują bardzo wysoką precyzję wykrywania punktów charakterystycznych ciała, jednak ich znaczne wymagania obliczeniowe ograniczają ich praktyczne wykorzystanie w systemach czasu rzeczywistego. MediaPipe oraz PoseNet wyróżniają się wysoką szybkością działania i niskim zapotrzebowaniem na zasoby, jednak ich skuteczność spada w przypadku bardziej złożonych gestów lub niestandardowych ułożeń ciała. Z kolei trt_pose, choć zoptymalizowany pod kątem urządzeń edge, napotyka problemy z kompatybilnością, co ogranicza jego uniwersalność. Natomiast algorytm YOLOv8m-pose prezentuje dobrą równowagę między szybkością a dokładnością nie wymagając wysokich parametrów sprzętowych.

W badaniach przeprowadzonych na rzecz niniejszej pracy, szczególnie istotne są szybkość działania oraz jednoznaczność klasyfikacji gestów. Z tego względu, jako najbardziej optymalny wybór w ramach przeprowadzonych badań, wskazano YOLOv8m-pose, który łączy wysoką dokładność z wydajnością i łatwością integracji, oferując jednocześnie wsparcie dla nowoczesnych architektur sprzętowych i możliwość dalszego rozwoju.

2.2.3.ROS 2 i MoveIt 2 jako platforma sterowania robotami

ROS 2 (3.3.2) to nowoczesne, modułowe i elastyczne środowisko do tworzenia oprogramowania robotycznego, zaprojektowane jako następca klasycznego ROS 1. MoveIt 2 (3.3.4) to natomiast zaawansowany framework służący do planowania trajektorii, sterowania ramionami robotów oraz zarządzania percepcją i kolizjami w środowisku robotycznym. ROS 2

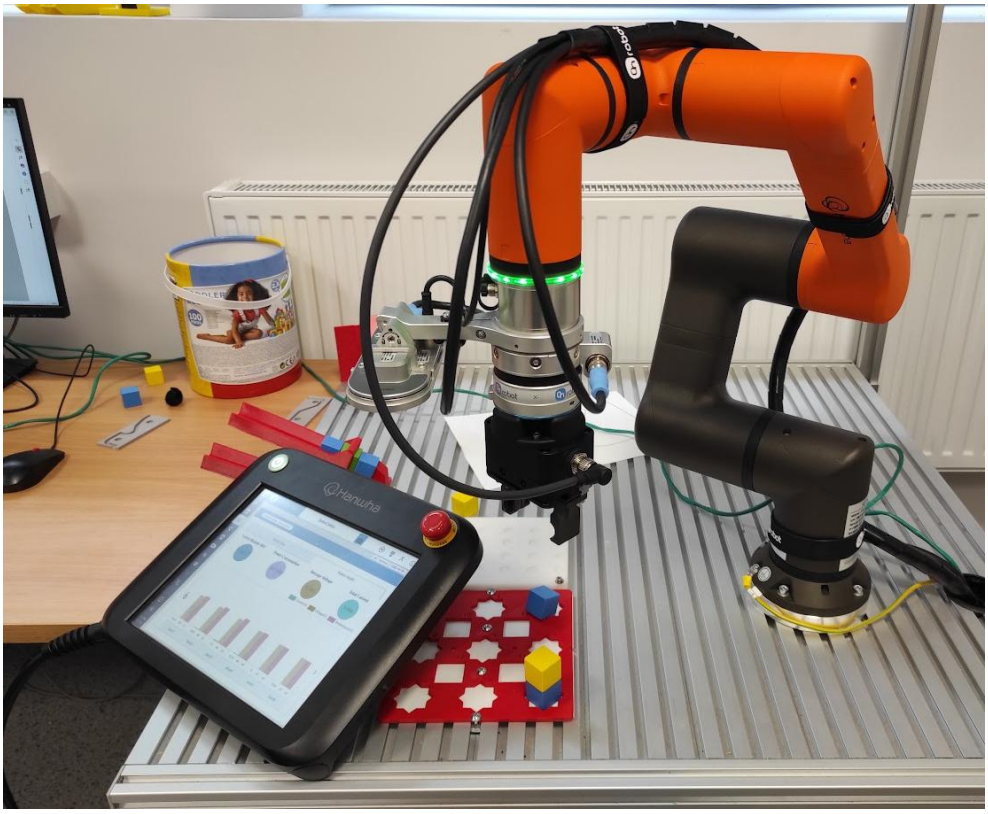
oferuje szereg zalet względem poprzednika: lepsze wsparcie dla systemów rozproszonych, większą niezawodność oraz kompatybilność z systemami czasu rzeczywistego (Real-Time Operating Systems). Dzięki tym cechom ROS 2 znajduje zastosowanie w nowoczesnych projektach przemysłowych, medycznych i naukowych, gdzie wymagana jest niezawodność, bezpieczeństwo i skalowalność. MoveIt 2 współpracuje bezpośrednio z ROS 2, oferując wsparcie dla języków programowania takich jak C++ i Python, integrację z Rviz2 (3.3.3) oraz obsługę popularnych platform robotycznych, w tym Hanwha HCR-3A. MoveIt 2 umożliwia m.in. planowanie trajektorii w oparciu o dane z kamer 3D, zarządzanie przestrzeniami kolizyjnymi oraz integrację z czujnikami zewnętrznymi [40]. Wybór środowiska ROS 2 i MoveIt 2 w tej pracy został podyktowany dostępnością pracy z systemem open-source, obsługą nowoczesnych standardów komunikacyjnych (DDS) oraz wsparciem dla zaawansowanego planowania ruchu i integracji z kamerami głębi. Zastosowanie Linuxa (Ubuntu) jako systemu operacyjnego również jest w pełni uzasadnione. Po pierwsze, większość bibliotek i bibliotek robotycznych, w tym ROS 2, jest natywnie wspierana tylko przez Linux. Po drugie, Linux umożliwia dostęp do narzędzi niskopoziomowych, wsparcie dla sterowników urządzeń oraz większą kontrolę nad procesami systemowymi [41]. Windows, mimo rosnącej kompatybilności z ROS 2, wciąż pozostaje mniej stabilnym wyborem dla projektów robotycznych wymagających niskiego opóźnienia, wysokiej niezawodności i szybkiego dostępu do sterowników sprzętowych.

Podsumowując, ROS 2 i MoveIt 2 oferują bogate środowisko pracy dla inżynierów robotyki, umożliwiające realizację złożonych projektów takich jak bezdotykowe sterowanie robotem w oparciu o rozpoznawanie gestów, co w połączeniu z systemem Linux zapewnia wysoką niezawodność, elastyczność oraz otwartość rozwiązania.

3. STANOWISKO BADAWCZE

3.1. Ramię Hanwha HCR3A

Robot współpracujący Hanwha HCR-3A (Zdjęcie 7) to nowoczesne rozwiązanie przemysłowe opracowane przez firmę Hanwha Robotics, które znajduje zastosowanie w zautomatyzowanych środowiskach produkcyjnych wymagających wysokiej precyzji i elastyczności. Jego konstrukcja została zoptymalizowana pod kątem bezpiecznej współpracy z ludźmi — dzięki zaawansowanym czujnikom kolizji robot potrafi natychmiast reagować na obecność człowieka w swoim otoczeniu, minimalizując ryzyko wypadków przy pracy.



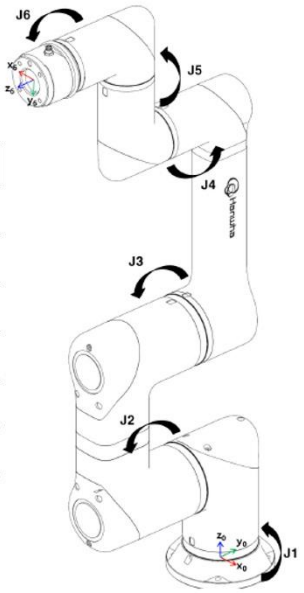
Zdjęcie 7 Stanowisko badawcze (wykonanie własne)

Robot charakteryzuje się wysoką powtarzalnością ruchów, co przekłada się na niezawodność w wykonywaniu powtarzalnych zadań takich jak montaż, pakowanie czy obsługa maszyn. Dodatkowo, modularna budowa HCR-3A umożliwia jego szybkie dostosowanie do różnorodnych procesów produkcyjnych, m.in. poprzez łatwą wymianę narzędzi końcowych [42].

3.1.1. Budowa

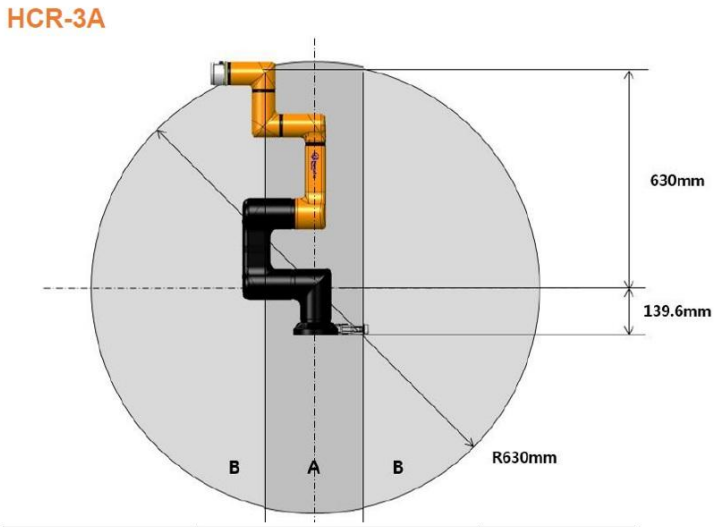
Ramię cechuje się kompaktową i lekką konstrukcją, co czyni go idealnym do pracy w ograniczonej przestrzeni roboczej. Robot ten posiada sześć osi przegubowych (Rysunek 2), które zapewniają dużą swobodę ruchu oraz precyzyjne operowanie narzędziami w różnych kierunkach i pozycjach [43]. Jego masa własna wynosi jedynie 13 kg, przy czym udźwig roboczy sięga do 3 kg, co wystarcza do większości lekkich zastosowań przemysłowych, takich jak przenoszenie komponentów czy obsługa urządzeń.

Operational axis	Operational range
J1 axis	$-360° < rz_1 \leq +360°$
J2 axis	$-360° < rz_2 \leq +360°$
J3 axis	$-160° < rz_3 \leq +160°$
J4 axis	$-360° < rz_4 \leq +360°$
J5 axis	$-360° < rz_5 \leq +360°$
J6 axis	$-\text{infinite} < rz_6 \leq +\text{infinite}$



Rysunek 2 Współrzędne osiowe i zakres operacyjny ramienia HCR-3A [43]

Zasięg robota to 630 mm (Rysunek 3), a jego powtarzalność pozycjonowania wynosi ±0,05 mm, co gwarantuje bardzo wysoką dokładność pracy. Obudowa robota została wykonana z wytrzymałych materiałów, odpornych na działanie czynników zewnętrznych, a zintegrowane okablowanie zmniejsza ryzyko uszkodzeń mechanicznych oraz ułatwia konserwację.



Typ	A	B
HCR-3A	74.3mm	630mm

Rysunek 3 Zakres pracy ramienia HCR-3A [43]

W robocie HCR-3A diody LED pełnią istotną funkcję informacyjną, umożliwiając szybką identyfikację aktualnego stanu systemu przez użytkownika. Zielony kolor diody wskazuje na normalny tryb pracy, w którym robot wykonuje zaprogramowane zadania w sposób ciągły i bezpieczny. Niebieskie światło LED oznacza aktywację trybu bezpośredniego nauczania, co umożliwia operatorowi ręczne prowadzenie robota w celu zapamiętania trajektorii ruchu.

Czerwony sygnał świetlny jednoznacznie informuje o aktywacji trybu zatrzymania awaryjnego, co wiąże się z natychmiastowym przerwaniem działania w przypadku wykrycia zagrożenia lub błędu systemowego. Z kolei kolor różowy sygnalizuje, że podczas sterowania w trybie nauczania została osiągnięta granica obrotu jednej z osi, co wymaga interwencji operatora w celu korekty położenia lub zmiany ustawień ruchu. System sygnalizacji LED w robocie HCR-3A stanowi zatem kluczowy element komunikacji człowiek–maszyna, zwiększając bezpieczeństwo pracy i ułatwiając diagnostykę operacyjną [43].

3.1.2.RODI

Do programowania ruchów ramienia Hanwha HCR-3A wykorzystywane jest dedykowane oprogramowanie RODI, które domyślnie można obsługiwać na pilocie uczenia robota. RODI oferuje przyjazny interfejs użytkownika (Zdjęcie 8) oraz intuicyjny sposób tworzenia programów oparty na logice blokowej, co znacząco upraszcza proces konfiguracji nawet dla mniej zaawansowanych operatorów.

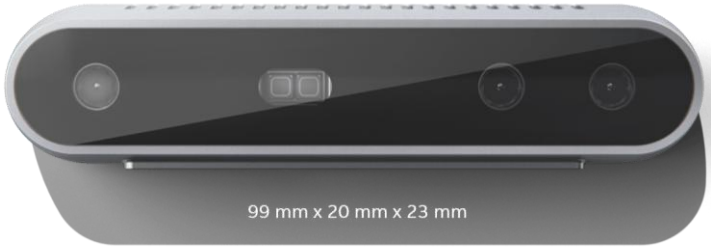
Zdjęcie 8 Przykładowy widok programowania pracy ramienia za pomocą oprogramowania RODI [44]

Programowanie blokowe polega na dzieleniu złożonych trajektorii ruchu na pojedyncze kroki, takie jak ruch liniowy, łukowy, rotacja poszczególnych osi, czy operacje warunkowe (if/else), pętle i przełączniki (switch/case). Istnieje również możliwość korzystania z wersji offline oprogramowania – RODI Off-line Simulator [44] – która umożliwia tworzenie, testowanie i optymalizację programów bez konieczności fizycznej obecności robota. Symulacja środowiska pracy pozwala na wczesne wykrywanie potencjalnych kolizji i błędów logicznych, co znacząco zwiększa bezpieczeństwo oraz efektywność wdrażania aplikacji.

3.2. System wizyjny

Kamera Intel® RealSense™ D435 (Zdjęcie 9) to zaawansowane urządzenie stereowizyjne, które łączy w sobie dwie kamery głębi, projektor podczerwieni oraz kamerę RGB, umożliwiając precyzyjne pozyskiwanie danych przestrzennych w czasie rzeczywistym. Dzięki

zastosowaniu aktywnej technologii stereoskopowej oraz projektora IR, kamera ta jest w stanie generować dokładne mapy głębi, co czyni ją idealnym narzędziem w aplikacjach takich jak robotyka, automatyzacja przemysłowa, skanowanie 3D czy systemy wizyjne [45].



Zdjęcie 9 Kamera Intel® RealSense™ model D435 z wymiarami [45]

3.2.1. Budowa i działanie

Konstrukcja D435 opiera się na wąskim polu widzenia (około 65° w poziomie i 40° w pionie) oraz zastosowaniu migawki typu rolling shutter, co przekłada się na wysoką rozdzielczość głębi i precyzyjne pomiary, szczególnie w przypadku mniejszych obiektów lub gdy wymagana jest większa dokładność. Urządzenie oferuje rozdzielczość głębi do 1280 × 720 pikseli przy 90 klatkach na sekundę oraz rozdzielczość obrazu RGB do 1920 × 1080 pikseli przy 30 klatkach na sekundę. Minimalna odległość pomiaru wynosi około 0,3 metra, a maksymalny zasięg efektywnego pomiaru sięga 3 metrów, co sprawia, że kamera doskonale sprawdza się w zastosowaniach wymagających precyzyjnego pomiaru w bliskim zasięgu [46].

3.2.2. Sterowniki i oprogramowanie

Integracja kamery z systemem ROS 2 odbyła się przy użyciu dedykowanego sterownika realsense2_camera, który umożliwia bezpośrednią komunikację z urządzeniem, konfigurację parametrów pracy (takich jak częstotliwość odświeżania, rozdzielczość czy tryb głębi), a także publikację danych w postaci wiadomości ROS. Alternatywnie, w przypadku korzystania wyłącznie z obrazu RGB, zastosowanie znajduje ogólny węzeł /v4l2_camera, zgodny ze standardem Video4Linux2, który umożliwia dostęp do obrazu z większości urządzeń wideo USB. Sterownik realsense2_camera zapewnia jednak szerszą funkcjonalność, w tym generowanie map głębi, chmur punktów 3D oraz automatyczną synchronizację z danymi kolorystycznymi. Dzięki temu kamera RealSense™ D435 może być skutecznie wykorzystywana w zadaniach takich jak detekcja gestów, śledzenie obiektów czy nawigacja w przestrzeni trójwymiarowej. Oprogramowanie dostarczane przez firmę Intel pozwala również na precyzyjne dostrajanie parametrów działania kamery, co ma istotne znaczenie dla zapewnienia stabilności i jakości danych wykorzystywanych w aplikacjach robotycznych.

3.3. Środowisko programistyczne

Domyślnym systemem sterowania ramienia Hanwha HCR3a jest opisany wcześniej RODI. Jednak do osiągnięcia wyznaczonego celu niezbędne było stworzenie innej formy komunikacji. Aby zapewnić kompleksowe środowisko do sterowania, wizualizacji i symulacji pracy robota Hanwha HCR-3A z wykorzystaniem zewnętrznych komponentów, takich jak kamera Intel® RealSense™ D435, wykorzystywane zostały otwarte platformy programistyczne, w tym ROS 2, RViz2, MoveIt na systemie operacyjnym Linux. Linux, jako otwarty system operacyjny, stanowi bazę dla uruchamiania całego środowiska. Dzięki rozproszonej architekturze ROS 2, opartej na DDS (Data Distribution Service), możliwa była niezawodna komunikacja pomiędzy różnymi modułami systemu, co jest istotne przy pracy z fizycznym robotem oraz obrazem w czasie rzeczywistym. Do pobierania i zarządzania pakietami oraz komponentami oprogramowania używano platformy GitHub, która stanowiła główne źródło bibliotek, narzędzi oraz gotowych implementacji. Dodatkowo wykorzystano narzędzia do kompilacji pakietów ROS 2 (`colcon build`), konfiguracji (`rviz2`, `rqt`, `ros2 launch`), zarządzania środowiskiem (`rosdep`, `vcstool`) oraz systemy kontroli wersji (`git`). W projekcie zastosowano języki programowania takie jak Python 3.10+ oraz C++. Python, jako język wysokiego poziomu, został użyty do tworzenia skryptów przetwarzających dane z kamery, integrujących algorytm detekcji gestów oraz do publikowania wiadomości ROS. C++ natomiast został wykorzystany w miejscach wymagających większej wydajności i niższego poziomu kontroli, m.in. w komponentach sterowania ruchem robota i przetwarzania danych w czasie rzeczywistym. Użyto również skryptów powłoki Bash do automatyzacji zadań instalacyjnych oraz konfiguracyjnych (np. inicjalizacja środowiska, instalacja zależności, budowa pakietów ROS 2).

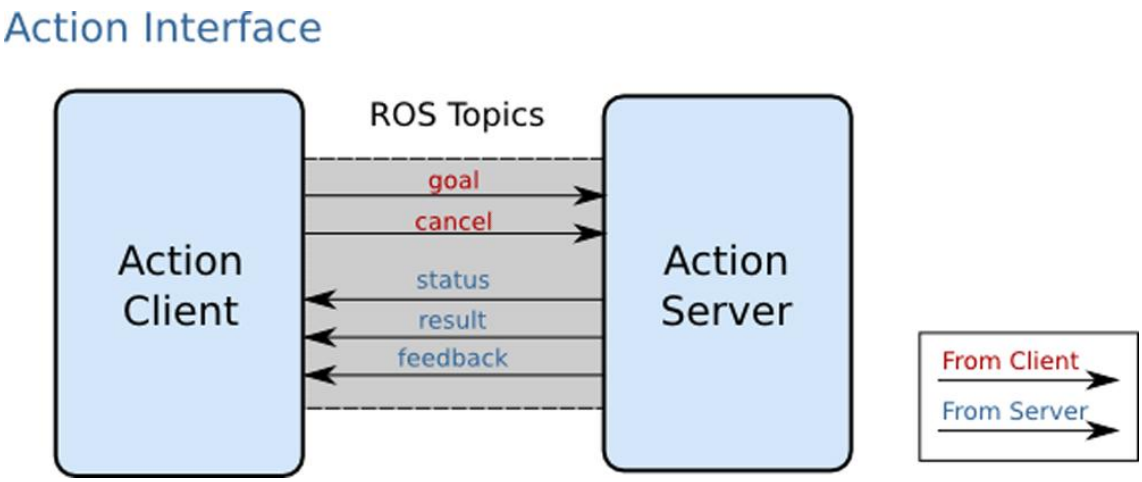
3.3.1. Ubuntu i jednostka operacyjna

Badania i testy zostały przeprowadzone na komputerze osobistym z procesorem Intel Core i5 12. generacji, 32 GB pamięci RAM oraz dedykowaną kartą graficzną NVIDIA T600. Do akceleracji obliczeń sieci neuronowych wykorzystywano środowisko CUDA, co pozwoliło na znaczne przyspieszenie działania modelu YOLOv8 w czasie rzeczywistym. Systemem operacyjnym była dystrybucja Ubuntu 22.04 LTS Jammy Jellyfish - stabilna wersja systemu Linux zalecana dla środowiska ROS 2 w wersji Humble Hawksbill.

Ubuntu zapewnia szerokie wsparcie dla środowisk developerskich, bibliotek open-source i narzędzi typowych dla robotyki. Jego przewaga nad systemem Windows wynika z pełnej kompatybilności z ROS 2, łatwego dostępu do pakietów i bibliotek (APT), wyższej stabilności systemowej, większej kontroli nad procesami oraz niskiego narzutu systemowego, co jest kluczowe w aplikacjach czasu rzeczywistego.

3.3.2.ROS 2

ROS 2 pełni rolę szkieletu komunikacyjnego – to system umożliwiający wymianę danych między różnymi komponentami systemu robotycznego, takimi jak kontroler robota, czujniki, algorytmy planowania ruchu czy systemy wizyjne. Dzięki architekturze opartej na publikatorach i subskrybentach, ROS 2 zapewnia wysoką skalowalność i niezawodność, co jest istotne w systemach, w których działanie musi być szybkie, dokładne i przewidywalne. W niniejszej pracy zastosowano ROS 2 w wersji Humble Hawksbill, która oferuje bogaty ekosystem gotowych pakietów i narzędzi [40].



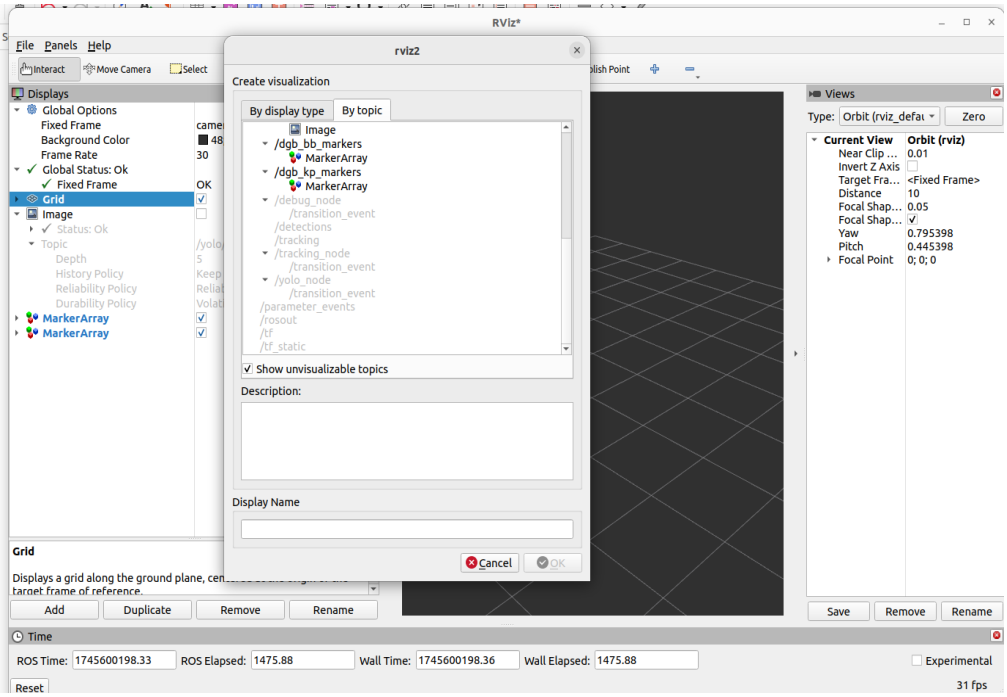
Rysunek 4 Schemat działania akcji w ROS 2 [47]

Jednym z kluczowych elementów funkcjonalnych ROS 2 są dodatki i pluginy, takie jak dla RViz2 do wizualizacji danych sensorycznych, ros2_control do zarządzania sprzętem wykonawczym, czy tf2 umożliwiający śledzenie układów współrzędnych w przestrzeni. Przykładowy schemat działania ROS 2 w niniejszym projekcie obejmuje współpracę węzłów odpowiadających za odbiór obrazu z kamery (np. realsense2_camera), analizę gestów (np. yolo_pose_node) oraz sterowanie robotem (np. cobot_controller), które komunikują się poprzez określone tematy i serwisy. Elastyczność ROS 2 oraz możliwość rozszerzania jego funkcjonalności za pomocą własnych wtyczek sprawia, że stanowi on solidną podstawę do budowy nowoczesnych, modułowych systemów robotycznych.

3.3.3.RViz2

RViz2 to narzędzie do wizualizacji danych w czasie rzeczywistym. Umożliwia przedstawienie na ekranie informacji pochodzących z czujników (np. dane głębi z kamery RealSense), modelu robota, planowanych trajektorii czy map środowiska. Jest niezbędne do diagnostyki i weryfikacji poprawności działania systemu podczas uruchamiania i testów. Interfejs użytkownika pozwala na elastyczne konfigurowanie widoku poprzez dodawanie różnych typów wizualizacji, takich jak punkty, linie, wektory sił, chmury punktów, a także trajektorie i ścieżki planowania ruchu. Możliwe jest również bezpośrednie monitorowanie pozycji i orientacji robota w przestrzeni dzięki integracji z transformacjami tf2. tf2 to biblioteka ROS 2 służąca do zarządzania układami współrzędnych, które posiada każdy element robota. Transformacje tf2 pozwalają

określać zależności przestrzenne między tymi układami i dzięki temu dynamicznie przeliczać pozycję i orientację jednego elementu względem innego.



Rysunek 5 Przykładowy widok interface'u programu Rviz2 (wykonanie własne)

RViz2 obsługuje różne formaty danych publikowanych w systemie ROS 2 i pozwala na ich filtrowanie oraz wizualną analizę w czasie rzeczywistym. Narzędzie to umożliwia także współpracę z symulatorami, takimi jak Gazebo, oraz z systemami SLAM i nawigacji, co czyni je uniwersalnym środowiskiem testowym. Przykład użycia RViz2 w niniejszej pracy obejmuje wizualizację punktów wykrytych przez algorytm detekcji gestów oraz potwierdzenie poprawnego przesyłu danych z kamery RealSense D435. RViz2 jest powszechnie stosowany w projektach akademickich, badawczo-rozwojowych oraz przemysłowych, gdzie pełni rolę kluczowego narzędzia diagnostycznego i prezentacyjnego. Zastosowany symulację hardware'u robota HCR3A stanowi symulowane środowisko sprzętowe, umożliwiające testowanie i rozwój oprogramowania sterującego bez konieczności fizycznego dostępu do rzeczywistego robota. Dzięki temu rozwiązaniu możliwe jest przeprowadzanie wstępnych analiz, symulacji oraz wdrażanie algorytmów sterowania w sposób bezpieczny i efektywny, co znacząco przyspiesza proces tworzenia oraz minimalizuje ryzyko uszkodzenia sprzętu w trakcie testów.

3.3.4.MoveIt

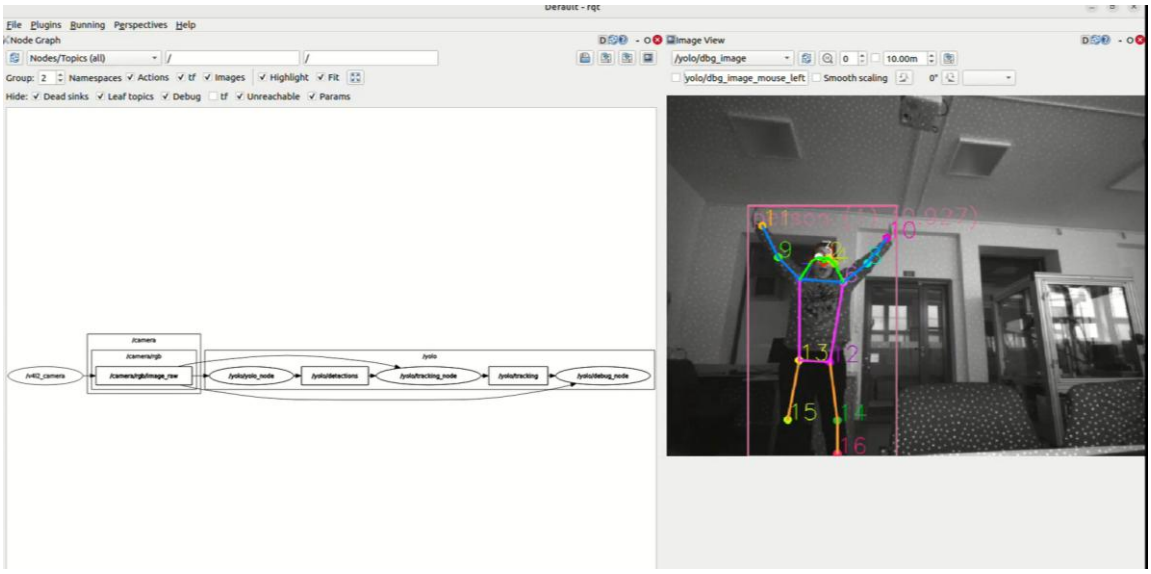
MoveIt z kolei odpowiada za planowanie i sterowanie ruchem manipulatora. Współpracując z ROS 2, pozwala na obliczanie trajektorii, omijanie przeszkód i wykonywanie precyzyjnych zadań z uwzględnieniem kinematyki robota. MoveIt 2 może być zintegrowany z RViz2 w celu wizualizacji zaplanowanego ruchu jeszcze przed jego wykonaniem w rzeczywistym środowisku, co znacząco zwiększa bezpieczeństwo i kontrolę nad procesem. System ten

obsługuje zarówno planowanie w przestrzeni konfiguracyjnej, jak i sterowanie za pomocą układu kartezjańskiego, umożliwiając wygodne definiowanie pozycji docelowej efektora robota.

W ramach projektu planowano zastosować moduł MoveIt Servo, który pozwala na płynne i szybkie sterowanie manipulatorem w czasie rzeczywistym, reagując dynamicznie na zmieniające się dane wejściowe, np. pozycję ciała wykrytą przez system wizyjny. MoveIt Servo opiera się na ciągłym przeliczeniu trajektorii na podstawie strumienia celów pozycyjnych, co czyni go idealnym rozwiązaniem do sterowania gestami. Przykładowym zastosowaniem było odwzorowanie pozycji końcówki ramienia robota na podstawie aktualnego położenia dłoni operatora, co umożliwiło naturalne i intuicyjne prowadzenie manipulatora bez konieczności ręcznego programowania pozycji. MoveIt 2, dzięki swojej modularności, wsparciu dla wielu typów manipulatorów oraz zgodności z ROS 2, stanowi obecnie jedno z najbardziej zaawansowanych narzędzi do sterowania ruchem w robotyce przemysłowej i badawczej.

3.3.5. RQT

RQT to graficzne narzędzie w ekosystemie ROS 2, które służy do monitorowania i zarządzania komponentami systemu robotycznego w czasie rzeczywistym. Umożliwia użytkownikowi przeglądanie tematów (topics), węzłów (nodes), usług (services), drzew tf (transformacji współrzędnych), a także wizualizację danych w postaci wykresów i interfejsów GUI. Rqt pełni ważną rolę diagnostyczną – pozwala szybko identyfikować błędy w komunikacji między węzłami, analizować wartości parametrów oraz monitorować stan systemu bez konieczności korzystania z terminala. Dzięki modularnej budowie, można dostosować środowisko do swoich potrzeb poprzez dodawanie lub usuwanie wtyczek.



Rysunek 6 Przykładowy widok interface'u programu RQt, po prawej autorka niniejszej pracy (wykonanie własne)

W praktyce użytkownik może zbudować własny układ okien, obejmujący np. podgląd aktualnie publikowanych wiadomości, graficzne przedstawienie parametrów dynamicznych lub wykresy zmian w czasie. Tak skonfigurowana perspektywa może zostać zapisana i ponownie

załadowana przy kolejnych uruchomieniach. W ramach niniejszej pracy RQT posłużyło m.in. do monitorowania działania kamery RealSense D435 oraz algorytmu YOLO, co pozwoliło na bieżąco oceniać jakość transmisji danych i poprawność przetwarzania informacji przez kolejne węzły systemu. Narzędzie to, ze względu na swoją elastyczność i szerokie możliwości konfiguracji, znajduje zastosowanie zarówno w środowiskach badawczych, jak i przemysłowych.

3.3.6.Algorytmy

W niniejszej pracy do detekcji pozycji i rozpoznawania gestów dłoni zastosowano algorytm YOLOv8 w wariancie modelu pose, który łączy wysoką dokładność wykrywania punktów charakterystycznych z możliwością działania w czasie rzeczywistym. Pochodzi z repozytorium Ultralytics i jest udostępniany na licencji GNU AGPL-3.0. [4]. YOLOv8 pose wykorzystuje jednoprzebiegową architekturę detekcyjną, co pozwala na szybkie przetwarzanie obrazu i równoczesne określenie położenia szkieletu dłoni lub całego ciała. Kluczowym czynnikiem wyboru tego rozwiązania była jego kompatybilność z systemem ROS 2 (YOLOv8 pose może być uruchomiony jako węzeł ROS 2) oraz stabilność działania na platformie sprzętowej używanej w projekcie. Alternatywnie rozważano użycie algorytmu trt_pose opracowanego przez firmę NVIDIA, który teoretycznie oferuje bardzo wysoką wydajność na urządzeniach typu Jetson. Jednak w praktyce jego konfiguracja okazała się problematyczna, głównie ze względu na ograniczoną dokumentację, brak oficjalnego wsparcia dla ROS 2 oraz niekompatybilność z kamerą Intel® RealSense™ D435. Z tego powodu podjęto decyzję o zastosowaniu rozwiązania bardziej uniwersalnego i łatwiejszego we wdrożeniu, jakim jest YOLOv8 pose, co pozwoliło na szybsze osiągnięcie założonych celów projektu bez kompromisu w zakresie jakości detekcji.

Algorytm nie jest trenowany wyłącznie na jednym zbiorze danych – jego architektura jest ogólnego przeznaczenia, a model może być trenowany lub dostosowany (fine-tuned) na wielu różnych zestawach danych, w zależności od konkretnego zastosowania (np. detekcja obiektów, segmentacja, estymacja pozycji – pose estimation). Do rozpoznawania pozycji został wykorzystany zestaw danych COCO (Common Objects in Context) to jeden z najważniejszych i najczęściej wykorzystywanych benchmarków w dziedzinie widzenia komputerowego, opracowany przez organizację Microsoft. COCO zawiera ponad 330 000 obrazów, z czego ponad 200 000 jest dokładnie oznaczonych, obejmując około 1,5 miliona instancji obiektów należących do 80 kategorii [48]. Cechą charakterystyczną tego zbioru jest uwzględnienie kontekstu – obiekty występują w realistycznych scenach, często w złożonych układach przestrzennych i z częściowym zasłonięciem. COCO dostarcza nie tylko adnotacji w postaci ramek ograniczających (bounding boxes), ale także masek segmentacji, punktów kluczowych (keypoints) dla estymacji pozycji ludzi, opisów obrazów oraz danych dla zadania panoptic segmentation (rodzaj segmentacji obrazu który, nie tylko rozpoznaje poszczególne obiekty ale też analizuje całą scenę). Dzięki temu umożliwia trenowanie i testowanie algorytmów w wielu podzadaniach, takich jak detekcja obiektów, segmentacja semantyczna, estymacja pozycji czy opis obrazu.

Algorytm YOLOv8 w wariancie pose estimation umożliwia precyzyjne rozpoznawanie punktów kluczowych (ang. keypoints) na ludzkim ciele, co jest kluczowe dla zadań takich jak

analiza postawy, śledzenie ruchu czy rozpoznawanie gestów. W przypadku estymacji pozycji człowieka, YOLOv8 wykrywa 17 predefiniowanych punktów kluczowych (Rysunek 6): (1) Nose (nos), (2) Left Eye (lewe oko), (3) Right Eye (prawe oko), (4) Left Ear (lewe ucho), (5) Right Ear (prawe ucho), (6) Left Shoulder (lewy bark), (7) Right Shoulder (prawy bark), (8) Left Elbow (lewy łokieć), (9) Right Elbow (prawy łokieć), (10) Left Wrist (lewe nadgarstek), (11) Right Wrist (prawy nadgarstek), (12) Left Hip (lewe biodro), (13) Right Hip (prawe biodro), (14) Left Knee (lewe kolano), (15) Right Knee (prawe kolano), (16) Left Ankle (lewa kostka) oraz (17) Right Ankle (prawa kostka). Każdy z punktów lokalizowany jest w przestrzeni dwuwymiarowej obrazu wraz z prawdopodobieństwem detekcji, co pozwala na ocenę wiarygodności pomiaru.

4. PRZEBIEG PROJEKTU

4.1. Implementacja środowiska

W pierwszym etapie skonfigurowano struktury katalogowe środowiska ROS 2 zgodnie z zaleceniami dokumentacji: utworzono przestrzeń roboczą (ros2_ws), w której znajdowały się katalogi src z kodem źródłowym pakietów. Pakiety potrzebne do obsługi kamery, detekcji gestów oraz sterowania robotem zostały pobrane głównie z platformy GitHub, w tym: realsense-ros do obsługi kamery Intel RealSense D435, YOLOv8m-pose do rozpoznawania gestów oraz pakiety MoveIt 2 i yolo_ros do integracji detekcji obiektów z ROS 2.

Podczas budowania przestrzeni roboczej za pomocą narzędzia colcon build, napotkano szereg błędów związanych z zależnościami oraz wersjami bibliotek systemowych. Przykładowo, brak kompatybilności pomiędzy wersją ONNX Runtime a implementacją YOLO wymagał ręcznej modyfikacji plików CMakeLists.txt oraz package.xml i dopasowania wersji pakietów wirtualnym środowisku Pythona. Dodatkowo, konieczne było zainstalowanie zależności przy pomocy rosdep install oraz ręczne dogranie brakujących bibliotek systemowych (np. librealsense2-dev, python3-colcon-common-extensions). W celu śledzenia błędów podczas kompilacji używano poleceń colcon build --event-handlers console_cohesion+ oraz --continue-on-error, co pozwalało diagnozować problemy w sposób iteracyjny.

Dzięki modułowej architekturze ROS 2 oraz wykorzystaniu wielu terminali, możliwe było jednoczesne uruchomienie kamery, detekcji gestów, węzła sterującego oraz narzędzi do wizualizacji. Każdy komponent był najpierw testowany osobno, a następnie integrowany i testowany w środowisku sprzętowym, co pozwalało na wczesne wykrycie potencjalnych konfliktów i optymalizację parametrów systemu (np. rozdzielczości obrazu, częstotliwości publikacji wiadomości, itp.). Każdy komponent systemu był testowany osobno w celu walidacji poprawności działania przed jego integracją. Kamera Intel RealSense D435 była uruchamiana za pomocą polecenia ros2 run v4l2_camera v4l2_camera_node, co pozwalało uzyskać strumienie obrazu RGB i głębi. Następnie uruchamiano detekcję gestów opartą na YOLOv8 przy użyciu dedykowanego węzła ROS2 (ros2 launch yolo_bringup yolo.launch.py), który publikował wiadomości na odpowiednich tematach. Do wizualizacji danych używano narzędzia rqt, które umożliwiało monitorowanie wiadomości na tematach takich jak /yolo/dbg_image, /yolo/dgb_bb_markerd, /yolo/detection czy /yolo/tracking. Uruchamiano również rviz2, gdzie w oparciu o model URDF robota można było obserwować pozycję ramienia w czasie rzeczywistym oraz planowane trajektorie ruchu.

```
#!/bin/bash
source /opt/ros/humble/setup.bash
source /home/student/STUDENT/ros2_ws/install/setup.bash

# Terminal 1: Kamera (v412) z remapem do YOLO
gnome-terminal --tab --title="Kamera" -- bash -c "ros2 run v412_camera v412_camera_node --ros-
args \
  -p video_device:=/dev/video2 \
  -r /image_raw:=/camera/rgb/image_raw; exec bash"

# Terminal 2: YOLO w trybie rozpoznawania pozy
gnome-terminal --tab --title="YOLO" -- bash -c "ros2 launch yolo_bringup yolo.launch.py model:
=yolov8m-pose.pt; exec bash"

# Terminal 3: RQT z załadowaną perspektywą (Node Graph + Image View)
gnome-terminal --tab --title="RQT" -- bash -c "rqt --perspective-file
/home/student/STUDENT/rqt_camera_yolo.perspective; exec bash"

# Terminal 4: Diagnostyka
#gnome-terminal -- bash -c "ros2 topic echo /diagnostics; exec bash"
gnome-terminal --tab --title="diagnostyka" -- bash -c "watch ros2 topic list; exec bash"

# Terminal 5: fake hardware
#gnome-terminal -- bash -c "ros2 topic echo /diagnostics; exec bash"
gnome-terminal --tab --title="fake hardware" -- bash -c "ros2 launch hcr3a
hcr_control.launch.py use_fake_hardware:=true ; exec bash"

# Terminal 6: RViz2 z modelem robota
#gnome-terminal --tab --title="RVIZ" -- bash -c "rviz2; exec bash"
#gnome-terminal --tab -- bash -c "cd ~/ROS2_PG/ws_ros2_hanwha; ros2 launch hcr3a
hcr_moveit.launch.py; exec bash"
gnome-terminal --tab --title="RVIZ" -- bash -c "ros2 launch hcr3a hcr_moveit.launch.py; exec
bash"
```

Rysunek 7 Skrypt uruchamiający wszystkie komponenty środowiska w osobnych kartach terminalu (wykonanie własne)

W celu uporządkowania wyświetlanych terminali w osobnych oknach napisano skrypt (Rysunek 7) uruchamiający wszystkie komponenty środowiska w osobnych kartach jednego terminalu. Napisano również skrypt, który zamyka opisane wcześniej karty w terminalu.

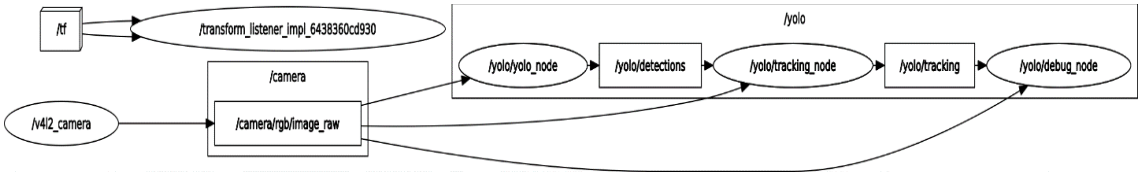
```
  GNU nano 6.2                              /home/student/.bashrc
alias src='source ~/STUDENT/ros2_ws/install/setup.bash'

# Rebuild (czyści i buduje od nowa)
alias rebuild='rm -rf build install log && cb'

# Rviz
alias rv='rviz2'

# Uruchomienie YOLO (jeśli pakiet yolo_bringup istnieje)
alias yolo='ros2 launch yolo_bringup yolo.launch.py'

# Kamera RealSense
alias rs_cam='ros2 launch realsense2_camera rs_launch.py'

# Lista tematów (topics)
alias rtl='ros2 topic list'

# Obserwuj obraz z kamery (jeśli dostępny)
alias cam_view='ros2 run rqt_image_view rqt_image_view'

# Echa tematów (np. obraz)
alias echo_img='ros2 topic echo /camera/color/image_raw'
alias echo_det='ros2 topic echo /yolo/detections'

alias startenv='bash /home/student/STUDENT/start_env.sh'
alias stopenv='bash /home/student/STUDENT/stop_env.sh'
source ~/STUDENT/ros2_ws/install/setup.bash

^G Help       ^O Write Out  ^W Where Is   ^K Cut       ^T Execute    ^C Location   M-U Undo
^X Exit       ^R Read File  ^\ Replace    ^U Paste     ^J Justify    ^/ Go To Line M-E Redo
```

Zdjęcie 10 Fragment skryptu .bashsrc

Dodano również aliasy (Zdjęcie 10) w skrypcie .bashsrc, który ułatwiały implementowanie środowiska korzystając z komendy startenv do włączanie oraz do zamykania środowiska – stopenv.

4.2. Komunikacja kamery z algorytmem YOLO

W ramach realizacji systemu rozpoznawania gestów z wykorzystaniem głębokiego uczenia, kluczowym etapem było nawiązanie połączenia pomiędzy strumieniem obrazu pochodzącym z kamery a algorytmem odpowiadającym za estymację pozycji. Integracja ta została zrealizowana w środowisku ROS 2, przy użyciu węzła /v4l2_camera generującego tematy typu /image_raw, które następnie były przetwarzane przez moduł inferencyjny YOLO.

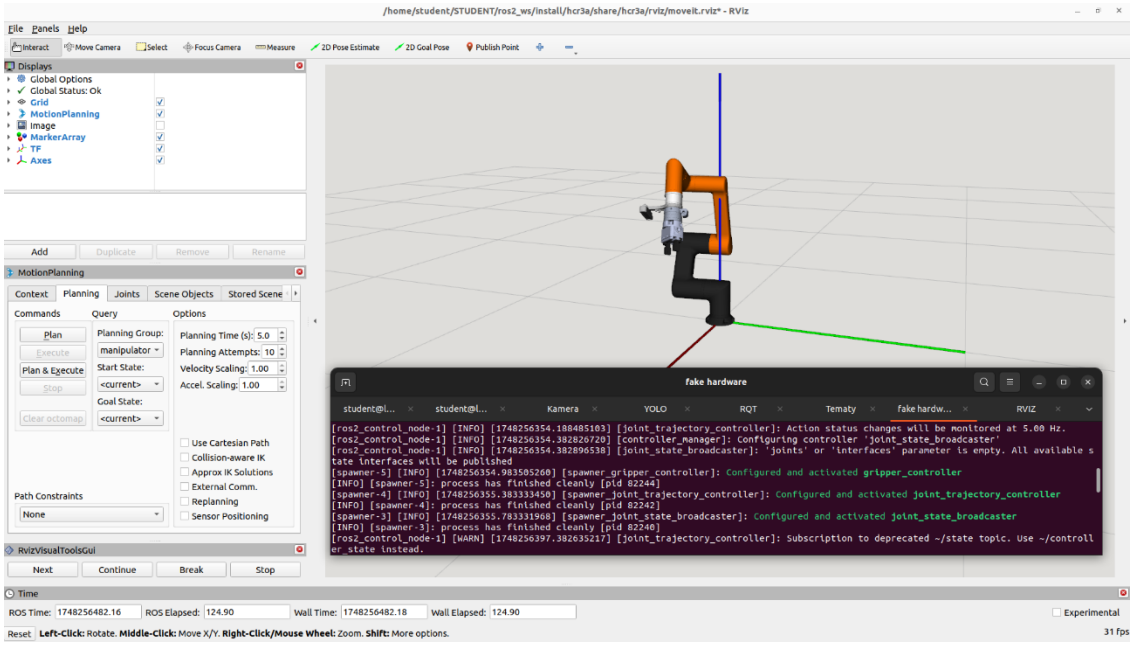


Rysunek 8 Schemat połączeń węzłów i tematów do obsługi algorytmu YOLO (wykonanie własne)

Obraz RGB z kamery stanowił wejście do algorytmu, który w czasie rzeczywistym lokalizował 17 punktów kluczowych na ciele człowieka. Poprawność działania systemu została zweryfikowana wizualnie w narzędziu RQT (Rysunek 8), gdzie na żywo prezentowano wykryte punkty wraz z identyfikacją postury osoby znajdującej się w kadrze (Rysunek 6). Osiągnięta płynność działania oraz precyzja detekcji potwierdziły skuteczność połączenia strumienia wideo z algorytmem sieci neuronowej, umożliwiając dalsze wykorzystanie wyników w kontekście rozpoznawania gestów i sterowania robotem.

4.3. Uruchamianie modelu robota HCR3a w programie RVz

W celu wizualizacji oraz testowania planowania ruchu robota HCR-3A w środowisku MoveIt 2, zastosowano sekwencję uruchamiania systemu w trybie symulowanym. W pierwszym kroku uruchomiono węzeł sterujący robotem przy użyciu polecenia ros2 launch hcr3a hcr_control.launch.py use_fake_hardware:=true (Zdjęcie 11), które aktywuje kontroler robota w trybie „fake hardware”, umożliwiając jego działanie bez potrzeby fizycznego połączenia z rzeczywistym kontrolerem ruchu robota HCR3a.

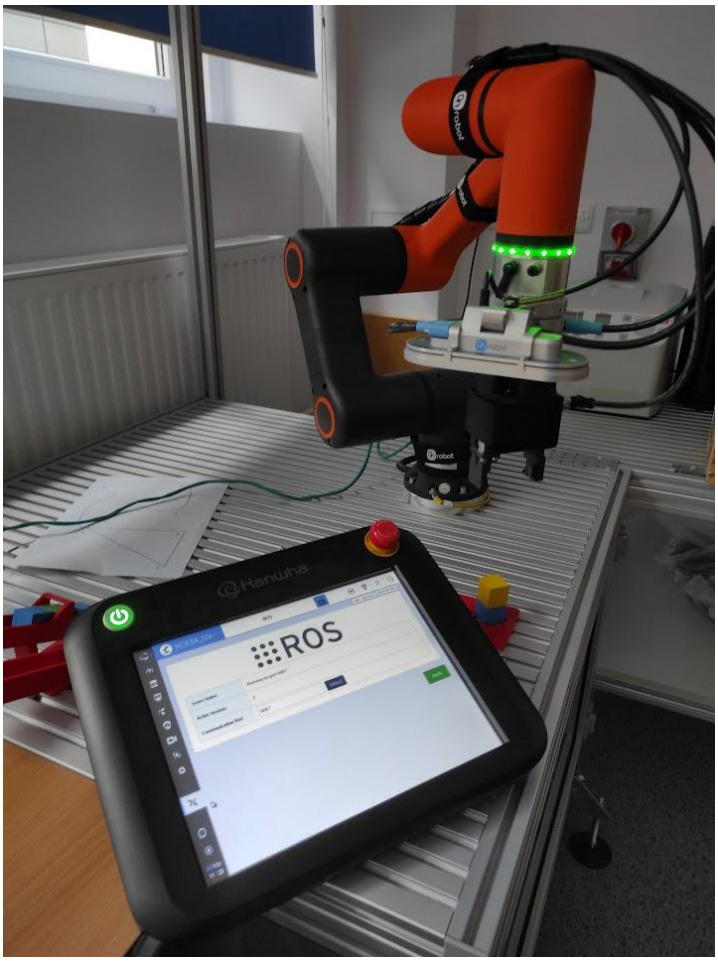


Zdjęcie 11 Fake hardware modelu HCR3a w Rviz

Tryb ten służy do weryfikacji poprawności konfiguracji oraz umożliwia przeprowadzanie testów z użyciem narzędzi symulacyjnych. Następnie, poprzez komendę ros2 launch hcr3a hcr_moveit.launch.py, uruchomiono komponent MoveIt 2 odpowiedzialny za planowanie i wykonanie trajektorii ruchu manipulatora. Połączenie z RViz2 pozwoliło na wizualizację modelu robota, jego pozycji oraz planowanych trajektorii w trójwymiarowym środowisku. Dzięki tej konfiguracji możliwe było pełne przetestowanie działania planowania ruchu oraz sposobu poruszania się ramienia bez uruchamiania fizycznego sprzętu.

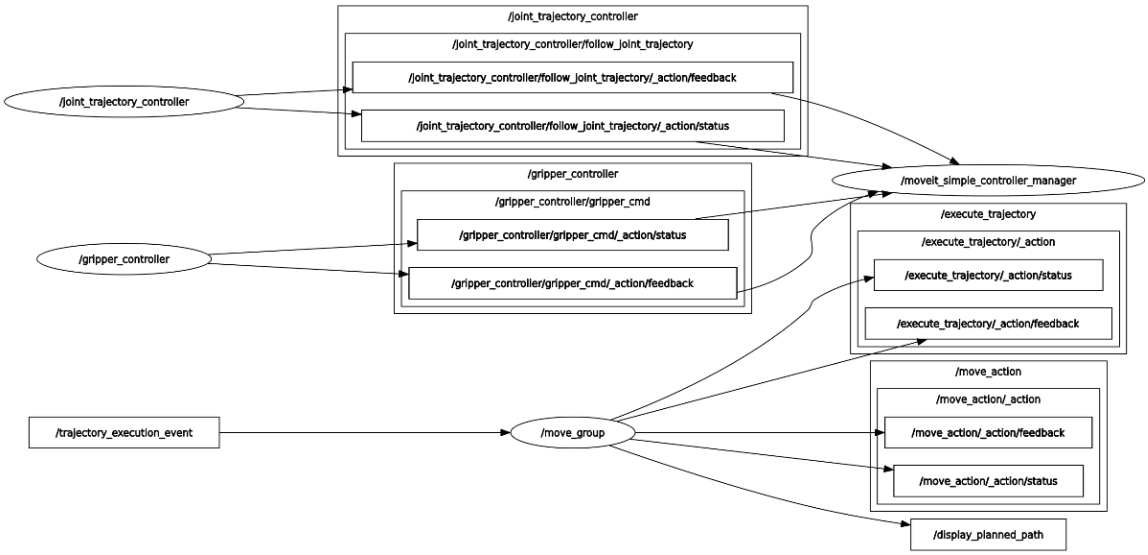
4.4. Komunikacja YOLO oraz kamery z modelem w RViz

W ramach projektu zintegrowano trzy kluczowe komponenty systemu sterowania gestami: kamerę, algorytm, środowisko ROS2 z modelem robota oraz zainstalowanym pakietem MoveIt 2. Przetestowano ponadto komunikację rzeczywistego robota HCR3a z programem Rviz (Zdjęcie 12).



Zdjęcie 12 Ramię HCR3A połączone ze środowiskiem ROS

W ramach przygotowania środowiska testowego uruchomiono kluczowe węzły odpowiedzialne za sterowanie robotem: /joint_trajectory_controller, odpowiadający za realizację zaplanowanych trajektorii ruchu manipulatora oraz /gripper_controller, zarządzający pracą chwytaka (Zdjęcie 13).



Zdjęcie 13 Fragment schematu węzłów z RQt

Węzły te zostały poprawnie zainicjalizowane i umożliwiły wizualne śledzenie zaplanowanych trajektorii w RViz2, co stanowi istotny krok przygotowawczy do pełnej integracji systemu sterowania gestami w następnym etapie pracy.

4.5. Problemy implementacyjne i ich rozwiązanie

W trakcie implementacji sterowania manipulatorem robota z wykorzystaniem MoveIt oraz interfejsu MoveGroupInterface napotkano szereg problemów uniemożliwiających poprawne działanie planowania kartezjańskiego. Pomimo uruchomionego środowiska symulacyjnego i aktywnych węzłów move_group oraz dostępnych parametrów robot_description i robot_description_semantic, pojawiały się błędy związane z niepoprawnym ładowaniem modelu SRDF, wskazujące na brak możliwości skonstruowania pełnego modelu robota w czasie wykonywania programu. Błąd ten uniemożliwiał poprawne zainicjalizowanie obiektu MoveGroupInterface, co skutkowało krytycznym wyjątkiem i zakończeniem pracy programu. Próby rozwiązania problemu poprzez modyfikację kolejności uruchamiania, zmianę sposobu przekazywania parametrów do węzła oraz wykorzystanie różnych metod planowania trajektorii nie przyniosły oczekiwanych rezultatów. W związku z tym, w celu zachowania możliwości sterowania robotem, zdecydowano się na alternatywne podejście polegające na bezpośrednim wysyłaniu trajektorii do poszczególnych stawów manipulatora, co pozwoliło na obejście problemów z inicjalizacją pełnego modelu robota i zapewniło działające rozwiązanie sterowania, choć kosztem rezygnacji z wygody i elastyczności planowania kartezjańskiego.

Decyzja o odejściu od sterowania kartezjańskiego za pomocą interfejsu MoveGroupInterface i wykorzystania bezpośredniego sterowania stawami robota była podyktowana nie tylko problemami technicznymi związanymi z inicjalizacją modelu robota, ale również ograniczeniami wynikającymi z dynamiki działania ROS 2 oraz mechanizmów ładowania parametrów w czasie rzeczywistym. W środowisku ROS 2 parametry takie jak robot_description i robot_description_semantic, które zawierają odpowiednio pełny opis modelu robota w formacie

URDF oraz dodatkowe informacje semantyczne zapisane w SRDF, muszą być poprawnie załadowane do serwera parametrów jeszcze przed inicjalizacją interfejsu MoveIt. W przypadku niedeterministycznego opóźnienia lub błędnego czasu ładowania parametrów, nawet chwilowy ich brak skutkuje przerwaniem działania i błędem krytycznym. Dodatkowo, w środowisku laboratoryjnym, w którym realizowano implementację, dostępne zasoby systemowe oraz potencjalne konflikty wersji bibliotek mogły pogłębiać problem inicjalizacji.

W związku z powyższym, sterowanie poszczególnymi stawami za pomocą bezpośredniego wysyłania trajektorii do kontrolera /joint_trajectory_controller/joint_trajectory stanowiło bardziej niezawodne rozwiązanie. Podejście to pomija warstwę planowania ruchu, opierając się na ręcznie definiowanych pozycjach dla każdego stawu robota. Choć takie rozwiązanie nie umożliwia automatycznego omijania przeszkód czy planowania optymalnej trajektorii w przestrzeni operacyjnej (co jest możliwe dzięki MoveIt), pozwala jednak na pełną kontrolę nad każdym stopniem swobody i jest wystarczające w przypadku prostych zadań manipulacyjnych, takich jak precyzyjne przemieszczanie chwytaka wzdłuż jednej osi lub wykonywanie powtarzalnych operacji w znanym środowisku.

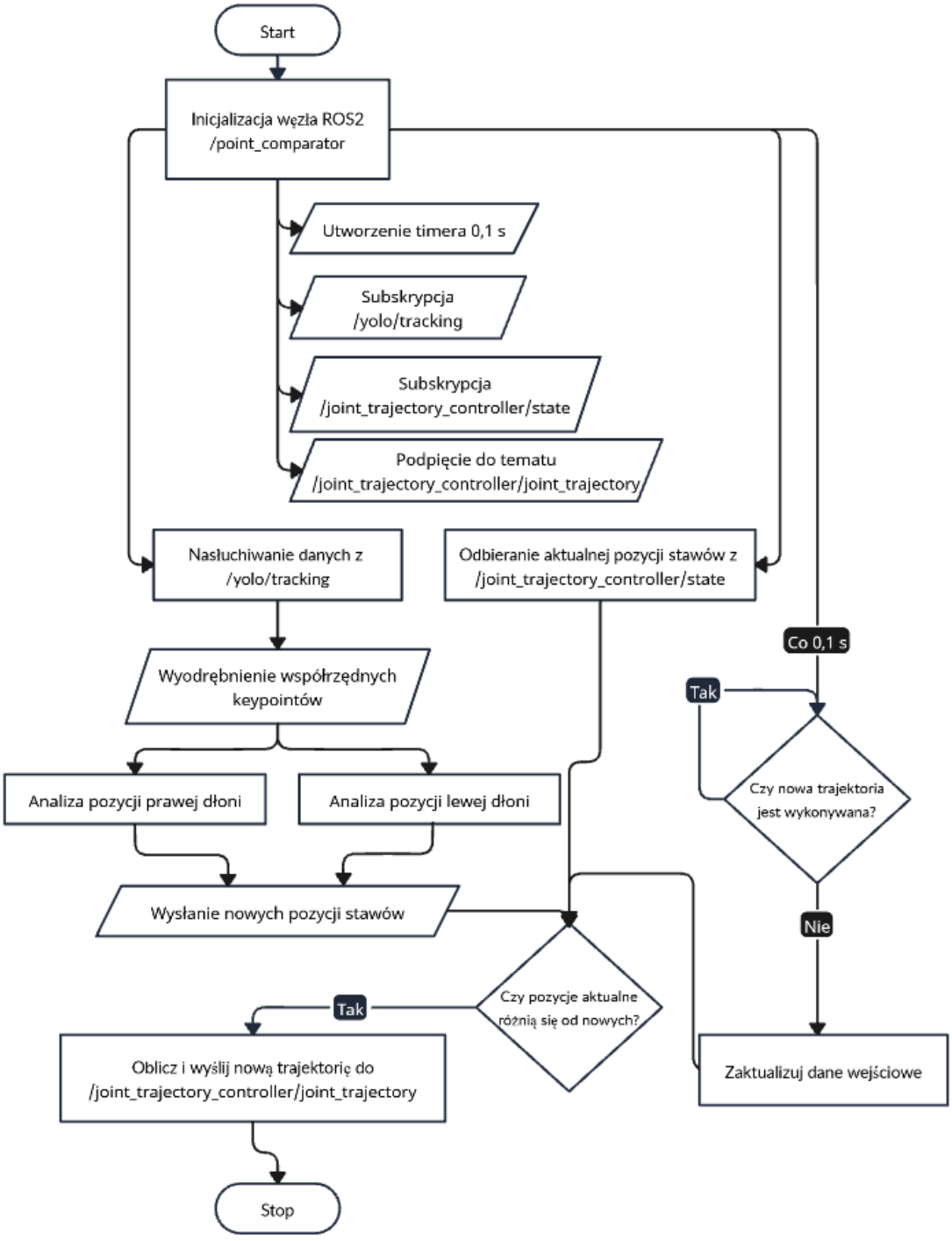
W kontekście realizacji projektu, zmiana podejścia do sterowania robotem wymusiła również modyfikację architektury oprogramowania oraz sposobu przetwarzania danych z systemu wizyjnego (YOLO). Zamiast przekształcać współrzędne obiektów wykrytych w obrazie do pozycji końcówki robota w przestrzeni kartezjańskiej i planować do nich ruch, dane te zostały użyte do obliczenia żądanych wartości pozycji jednego lub kilku stawów, co uprościło cały proces. Mimo ograniczenia funkcjonalności, rozwiązanie to pozwoliło na osiągnięcie podstawowego celu projektu, czyli reaktywnego sterowania robotem na podstawie wizji komputerowej, przy jednoczesnym zapewnieniu większej stabilności i niezawodności działania w rzeczywistym

4.6. Sterowanie ramieniem – kod źródłowy

Moduł PointComparator (**Błąd! Nie można odnaleźć źródła odwołania.**), z aimplementowany jako węzeł ROS 2 w języku Python, stanowi kluczowy komponent odpowiedzialny za sterowanie ruchem ramienia robota na podstawie danych pochodzących z detekcji pozycji dłoni użytkownika. Program ten wykorzystuje bibliotekę rclpy, będącą oficjalnym klientem ROS 2 dla Pythona, oraz komunikaty typu DetectionArray z pakietu yolo_msgs, służące do przekazywania współrzędnych punktów charakterystycznych wykrytych przez sieć YOLOv8 przystosowaną do detekcji pozycji ciała. Głównym założeniem działania węzła jest porównywanie współrzędnych punktów o identyfikatorach przypisanych do określonych punktów ciała (np. ID=1 – nos, ID=10 – lewa dłoń, ID=6 – lewy bark) i podejmowanie decyzji ruchowych w zależności od ich położenia na osi Y, czyli wysokości na obrazie.

W konstruktorze klasy PointComparator tworzony jest timer o częstotliwości 10 Hz, który regularnie wywołuje funkcję timer_callback odpowiedzialną za publikację trajektorii, jeżeli została uprzednio zakwalifikowana jako potrzebna. Subskrypcja tematu /yolo/tracking umożliwia odbieranie komunikatów zawierających wykryte punkty kluczowe dłoni. W oddzielnym kanale

komunikacyjnym /joint_trajectory_controller/state węzeł odbiera informacje o aktualnym i oczekiwanym stanie przegubów manipulatora, co pozwala na weryfikację, czy poprzednia trajektoria została już zakończona. Do wysyłania nowych trajektorii wykorzystywana jest publikacja na temat /joint_trajectory_controller/joint_trajectory.



Rysunek 9 Schemat blokowy kodu odpowiedzialnego za sterowanie ramieniem (opracowanie własne)

Najważniejszym mechanizmem programu jest analiza położenia punktów ID=1, ID=10 i ID=6 dla lewej ręki oraz ID=1, ID=11 i ID=7 dla prawej. W przypadku wykrycia, że punkt ID=10 (lewa dłoń) znajduje się powyżej punktu ID=1 (nos), węzeł interpretuje to jako polecenie

podniesienia ramienia (ruch w kierunku ujemnych wartości joint_3). Analogicznie, jeśli ID=10 znajduje się pomiędzy ID=1 a ID=6, ramie jest opuszczane poprzez zwiększenie wartości joint_3. W sytuacji, gdy ID=10 znajduje się poniżej ID=6, wydawany jest sygnał do zatrzymania ruchu. Ten sam schemat stosowany jest dla prawej ręki, z tą różnicą, że kontrolowany jest joint_1, odpowiedzialny za ruch boczny manipulatora.

Dzięki zastosowaniu zmiennych logicznych takich jak executing i new_trajectory_needed, uniknięto problemu nadpisywania trajektorii w trakcie wykonywania ruchu. Każda zmiana kąta obrotu przegubu musi przekraczać określony próg (tutaj: 1e-4 radiana), by system uznał, że zasadne jest wygenerowanie nowej trajektorii. Zmienna step, ustalona na 0.04, kontroluje wielkość zmiany kąta, zapewniając płynność ruchu i eliminując ryzyko oscylacji wokół wartości granicznych. Zakres możliwego obrotu jest dodatkowo ograniczony zmiennymi min_rad i max_rad, co zapobiega mechanicznemu przeciążeniu robota.

Funkcja send_trajectory konstruuje obiekt JointTrajectory, przypisując do listy positions aktualne wartości kątów obrotu joint_1 i joint_3, przy czym pozostałe przeguby pozostają w spoczynku. Komunikat ten publikowany jest na odpowiedni temat, co skutkuje wykonaniem ruchu przez robota. Dodatkowo, system śledzi ostatnio wysłane wartości przegubów, aby zapobiec wielokrotnemu publikowaniu identycznych trajektorii.

Funkcja state_callback analizuje, czy aktualna pozycja przegubów odpowiada pozycji oczekiwanej z dokładnością do 0.01 radiana. Jeśli wszystkie przeguby znajdują się w zadanej pozycji, system uznaje, że trajektoria została zakończona, co resetuje flagę executing.

Ostatecznie, program działa jako zamknięta pętla sprzężenia zwrotnego pomiędzy systemem percepcji (kamera + YOLO), przetwarzaniem logiki gestów (analiza punktów) oraz układem wykonawczym (ramię robota sterowane poprzez trajektorie). W czasie rzeczywistym możliwa jest płynna interakcja użytkownika z robotem poprzez naturalne gesty dłoni, co zostało potwierdzone podczas testów z użyciem RViz oraz terminalowego debugowania komunikatów w RQT. Zastosowana architektura zapewnia modularność, skalowalność i wysoką responsywność systemu, a sam węzeł PointComparator stanowi istotny komponent pomostowy pomiędzy warstwą percepcyjną a wykonawczą całego układu sterowania.

5. TESTOWANIE KODU I ANALIZA WYNIKÓW

Celem niniejszego etapu pracy było przeprowadzenie badań eksperymentalnych nad działaniem zaprojektowanego i zaimplementowanego programu sterującego ramieniem robota na podstawie analizy pozycji ciała użytkownika. Główne pytania badawcze dotyczyły:

- skuteczności systemu w interpretacji gestów jako komend sterujących,
- czasu reakcji systemu, czyli opóźnienia między wykryciem pozycji ciała a rozpoczęciem ruchu robota,
- powtarzalności i niezawodności działania algorytmu w warunkach rzeczywistych.

Postawiono hipotezę, że zastosowanie przetwarzania strumienia danych z kamery w połączeniu z wykrywaniem punktów kluczowych za pomocą modelu YOLOv8 oraz użyciem infrastruktury ROS 2 umożliwi precyzyjne i płynne sterowanie ramieniem robota w czasie rzeczywistym, z minimalnym opóźnieniem nieprzekraczającym 0,5 sekundy od momentu detekcji gestu do momentu fizycznego rozpoczęcia ruchu przez manipulator.

W celu weryfikacji powyższej hipotezy przeprowadzono szereg testów jakościowych i ilościowych. Testy jakościowe polegały na obserwacji działania systemu w rzeczywistych warunkach, z udziałem operatora wykonującego zestaw zdefiniowanych gestów – zarówno dla lewej, jak i prawej ręki. Analizowano, czy gesty są prawidłowo interpretowane przez system oraz czy generowany na ich podstawie ruch robota odpowiada zamierzonej komendzie. W testach ilościowych skupiono się na pomiarze czasu reakcji systemu, definiowanego jako różnica czasowa pomiędzy momentem odbioru wiadomości ROS zawierającej pozycje punktów kluczowych a momentem opublikowania trajektorii sterującej ruchem ramienia. W tym celu konieczne było wprowadzenie modyfikacji do kodu źródłowego węzła PointComparator. Zaimportowano bibliotekę time a w funkcji __init__(self) dodano zmienną przechowującą czas wysłania wiadomości oraz ścieżkę zapisu danych w pliku tekstowym trajectory_log.txt.

```
self.last_sent_time = None
self.log_file_path = '/home/student/STUDENT/ros2_ws/trajec-
tory_log.txt'
```

Do funkcji listener_callback dodano rejestrację czasu odbioru wiadomości za pomocą funkcji self.last_sent_time = time.time(). Następnie, w momencie publikacji trajektorii w funkcji state_callback, również zarejestrowano czas systemowy i obliczono różnicę pomiędzy tymi dwoma znacznikami czasu.

```
if self.executing:
   elapsed = time.time() - self.last_sent_time if
self.last_sent_time else -1
   self.get_logger().info(f"Trajektoria zakończona po {elap-
sed:.3f} sek.")
```

```
    with open(self.log_file_path, 'a') as f:
        f.write(f"Czas: {elapsed:.3f} s | joint_1={self.cur-
rent_rad_joint_1:.4f}, joint_3={self.cur-
rent_rad_joint_3:.4f}\n")
```

Zmodyfikowany węzeł wypisywał każdorazowo zmierzoną wartość opóźnienia w terminalu ROS, co umożliwiało analizę statystyczną zgromadzonych wyników w późniejszych etapach badań. Fragment pliku tekstowego z zapisanymi danymi:

Czas: 0.359 s | joint_1=-0.3600, joint_3=-0.1200

Czas: 0.559 s | joint_1=-0.4000, joint_3=-0.1200

Czas: 0.559 s | joint_1=-0.3600, joint_3=-0.1200

Czas: 0.359 s | joint_1=-0.3200, joint_3=-0.1200

Czas: 0.359 s | joint_1=-0.2800, joint_3=-0.1200

Czas: 0.360 s | joint_1=-0.2400, joint_3=-0.1200

Czas: 0.359 s | joint_1=-0.2000, joint_3=-0.1200

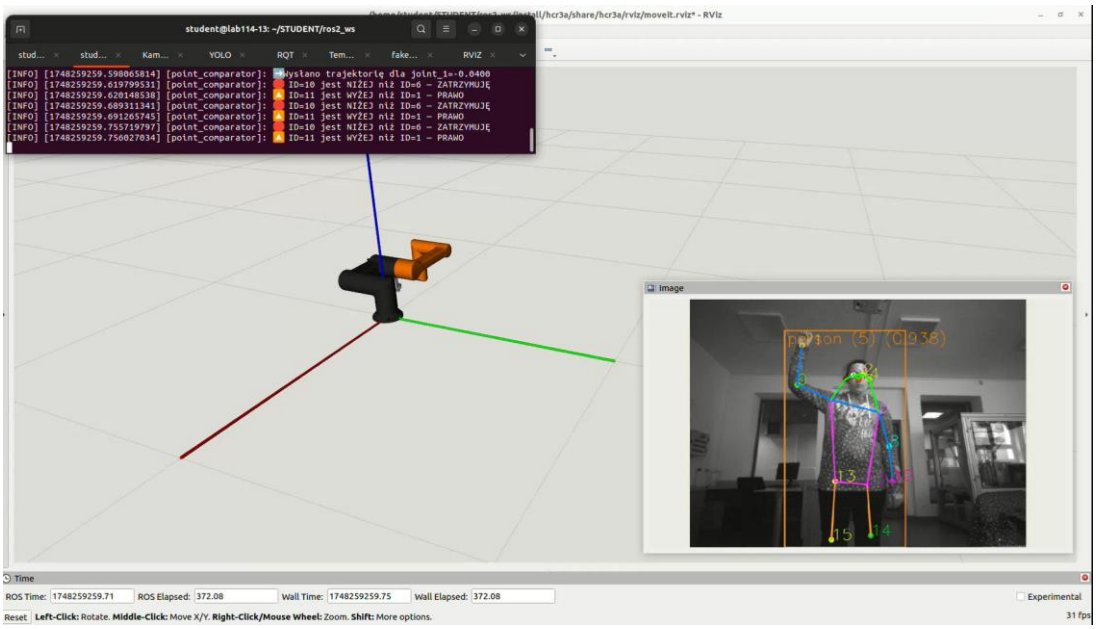
Czas: 0.359 s | joint_1=-0.1600, joint_3=-0.1200

Czas: 0.359 s | joint_1=-0.1200, joint_3=-0.1200

Wyniki były także zapisywane do pliku tekstowego za pomocą funkcji f.write, co ułatwiło ich późniejszą agregację i wizualizację.

5.1. Warunki testowe

Warunki testowe były ściśle kontrolowane i ujednolicone we wszystkich próbach. Testy przeprowadzono w zamkniętym pomieszczeniu laboratoryjnym przy stałym oświetleniu sztucznym, aby wyeliminować wpływ warunków zewnętrznych na jakość detekcji obrazu. Kamera RGB o rozdzielczości 640x480 pikseli pracowała z częstotliwością 30 klatek na sekundę i była zamocowana na statywie w odległości od operatora o około 0,5 metra dla pierwszej serii oraz około 1,2 metra dla drugiej serii (Zdjęcie 14). Model robota wyposażony w sześć stopni swobody był połączony z systemem ROS 2 za pomocą kontrolera joint_trajectory_controller, a jego trajektorie wizualizowano w czasie rzeczywistym przy użyciu narzędzia RViz. Dodatkowo, uruchomiony był panel RQT do monitorowania tematów ROS i rejestracji parametrów systemowych.



Zdjęcie 14 Praca programu dla odległości 1,2 metra operatora od kamery (wykonanie własne)

Dla każdej serii testów wykonano sekwencję ruchów:

- lewa ręka wymusza podniesienie ramienia (na czas około 2 sekund i następuje opuszczenie ręki na martwą strefę na 2 sekundy)
- lewa ręka wymusza opuszczenie ramienia
- prawa ręka wymusza obrót ramienia w prawo
- prawa ręka wymusza obrót ramienia w lewo
- lewa ręka wymusza podniesienie ramienia i prawa ręka wymusza obrót ramienia w prawo
- lewa ręka wymusza opuszczenie ramienia i prawa ręka wymusza obrót ramienia w lewo
- lewa ręka wymusza opuszczenie ramienia i prawa ręka wymusza obrót ramienia w prawo
- lewa ręka wymusza podniesienie ramienia i prawa ręka wymusza obrót ramienia w lewo

Każdy ruch był wykonywany przez około 2 sekundy a następnie obie ręce były kierowane do „martwej strefy” czyli poniżej punktów 4 i 6 aby nie wysyłać żadnych wiadomości do ramienia.

W każdej próbie mierzono czas od momentu rejestracji pozycji punktów kluczowych do publikacji trajektorii oraz obserwowano, czy wykonany ruch był zgodny z zamierzoną komendą. W przypadku nieprawidłowości analizowano przyczyny błędu, które najczęściej wynikały z chwilowego braku detekcji punktów (np. jeśli dłoń była częściowo zasłonięta lub poza polem widzenia kamery). Ponadto, analizowano również sytuacje, w których różnica wysokości pomiędzy punktami była zbyt mała, by spełnić warunek logiczny do podjęcia decyzji ruchowej – co skutkowało brakiem reakcji robota, mimo że operator wykonał gest.

5.2. Dane z testów

Każda linijka z plików trajectory_log zamierała informacje na temat czas wykonania, aktualnej pozycji stawu 1 oraz stawu 3. W celu wydobycia danych liczbowych z tego pliku oraz umieszczeniu ich w formie tabeli korzystano z programu Excel. Stworzono tabele z 4 kolumnami: „Np.” – numer próbki, „Czas [s]” – czas trwania akcji, „j1 kierunek” – kierunek obrotu stawu 1 (joint 1) i , „j3 kierunek” – kierunek obrotu stawu 3 (joint 3). Dodatkowo sformatowano wartości czasu trwania akcji za pomocą skali kolorów co ułatwia rozpoznanie się w wynikach.

Pierwsza seria danych (Tabela 5.1 I Seria danych z testowania kodu źródłowego w odległości 0,5 metra od operatora (wykonanie własne)Tabela 5.1) polegała na zbieraniu danych w mniejszej odległości operatora od kamery, około 0,5 metra. Z przeprowadzonych 57 próbek, YOLO poprawnie wykryło wszystkie wymagane punkty kluczowe we wszystkich przypadkach co odpowiada skuteczności na poziomie 100%. W przypadku próbki numer 1, 14, 30, 42 i 48 występują ruchy manipulatora, które początkowo wyglądają na anomalie. Powodem ich zarejestrowania jest jednak ruch operatora i zbyt wczesne wykrycie punktów kluczowych. Na przykład dla podniesienia ramienia (joint 3) lewa ręka operatora (punkt kluczowy 10) przeszła przez pole między punktami kluczowymi 1 i 6 wysyłając prośbę o opuszczenie ramienia (próbka 1).

Tabela 5.1 I Seria danych z testowania kodu źródłowego w odległości 0,5 metra od operatora (wykonanie własne)

Np.	Czas [s]	j1 kierunek	j3 kierunek
1	0,43		DÓŁ
2	0,53		GÓRA
3	0,33		GÓRA
4	0,53		GÓRA
5	0,23		GÓRA
6	0,33		GÓRA
7	0,531		DÓŁ
8	0,33		DÓŁ
9	0,33		DÓŁ
10	0,33		DÓŁ
11	0,33		DÓŁ
12	0,53		DÓŁ
13	0,528		DÓŁ
14	0,43	LEWO	
15	0,33	PRAWO	
16	0,33	PRAWO	
17	0,33	PRAWO	
18	0,33	PRAWO	
19	0,33	PRAWO	
20	0,33	PRAWO	
21	0,53	PRAWO	
22	0,23	LEWO	
23	0,33	LEWO	
24	0,33	LEWO	
25	0,332	LEWO	
26	0,33	LEWO	
27	0,33	LEWO	
28	0,33	LEWO	
29	0,33	LEWO	

Np.	Czas [s]	j1 kierunek	j3 kierunek
30	0,23		DÓŁ
31	0,53	PRAWO	GÓRA
32	0,33	PRAWO	GÓRA
33	0,53	PRAWO	GÓRA
34	0,53	PRAWO	GÓRA
35	0,33	PRAWO	GÓRA
36	0,53	LEWO	DÓŁ
37	0,43	LEWO	DÓŁ
38	0,33	LEWO	DÓŁ
39	0,33	LEWO	DÓŁ
40	0,53	LEWO	DÓŁ
41	0,33	LEWO	DÓŁ
42	0,23	LEWO	
43	0,33	PRAWO	DÓŁ
44	0,33	PRAWO	DÓŁ
45	0,33	PRAWO	DÓŁ
46	0,33	PRAWO	DÓŁ
47	0,531	PRAWO	DÓŁ
48	0,23		DÓŁ
49	0,33		GÓRA
50	0,33		GÓRA
51	0,33		GÓRA
52	0,33		GÓRA
53	0,33	LEWO	GÓRA
54	0,53	LEWO	GÓRA
55	0,53	LEWO	GÓRA
56	0,33	LEWO	GÓRA
57	0,33	LEWO	GÓRA

W przypadku drugiej serii danych (Tabela 5.2), odległość operatora od kamery wynosiła około 1,2 metra. Z przeprowadzonych 54 próbek, YOLO poprawnie wykryło wszystkie wymagane punkty kluczowe co również odpowiada skuteczności na poziomie 100% tak jak w przypadku I serii. Kolejnym podobieństwem między seriami jest występowanie zbyt wczesnych wykryć pozycji operatora.

Tabela 5.2 II Seria danych z testowania kodu źródłowego w odległości 1,2 metra od operatora (wykonanie własne)

np.	Czas [s]	j1 kierunek	j3 kierunek	np.	Czas [s]	j1 kierunek	j3 kierunek
1	0,298		DÓŁ	28	0,298	PRAWO	GÓRA
2	0,298		GÓRA	29	0,298	PRAWO	GÓRA
3	0,498		GÓRA	30	0,298	PRAWO	GÓRA
4	0,298		GÓRA	31	0,298	PRAWO	GÓRA
5	0,298		GÓRA	32	0,298	LEWO	DÓŁ
6	0,298		GÓRA	33	0,398		DÓŁ
7	0,498		DÓŁ	34	0,298	LEWO	DÓŁ
8	0,298		DÓŁ	35	0,298	LEWO	DÓŁ
9	0,498		DÓŁ	36	0,298	LEWO	DÓŁ
10	0,298		DÓŁ	37	0,298	LEWO	DÓŁ
11	0,298		DÓŁ	38	0,298	LEWO	DÓŁ
12	0,298		DÓŁ	39	0,298	LEWO	DÓŁ
13	0,598	LEWO		40	0,598	LEWO	
14	0,298	PRAWO		41	0,499	PRAWO	DÓŁ
15	0,298	PRAWO		42	0,298	PRAWO	DÓŁ
16	0,3	PRAWO		43	0,298	PRAWO	DÓŁ
17	0,298	PRAWO		44	0,298	PRAWO	DÓŁ
18	0,298	PRAWO		45	0,298	PRAWO	DÓŁ
19	0,298	LEWO		46	0,298	PRAWO	DÓŁ
20	0,498	LEWO		47	0,498		DÓŁ
21	0,298	LEWO		48	0,298		GÓRA
22	0,298	LEWO		49	0,298	LEWO	GÓRA
23	0,498	LEWO		50	0,298	LEWO	GÓRA
24	0,298	LEWO		51	0,298	LEWO	GÓRA
25	0,298	LEWO		52	0,298	LEWO	GÓRA
26	0,298	LEWO	DÓŁ	53	0,298	LEWO	GÓRA
27	0,496	PRAWO	GÓRA	54	0,298	LEWO	GÓRA

Dla pełniejszej analizy otrzymanych wyników posegregowano otrzymane dane (Tabela 5.3) pod względem wysłanych poleceń do ramienia robota.

Tabela 5.3 Fragment danych z I serii

np.	Czas [s	j1 kierun	j3 kierune
22	0,23	LEWO	
42	0,23	LEWO	
23	0,33	LEWO	
24	0,33	LEWO	
26	0,33	LEWO	
27	0,33	LEWO	
28	0,33	LEWO	
29	0,33	LEWO	
25	0,332	LEWO	
14	0,43	LEWO	
15	0,33	PRAWO	
16	0,33	PRAWO	
17	0,33	PRAWO	
18	0,33	PRAWO	
19	0,33	PRAWO	
20	0,33	PRAWO	
21	0,53	PRAWO	
30	0,23		DÓŁ
48	0,23		DÓŁ
8	0,33		DÓŁ
9	0,33		DÓŁ
10	0,33		DÓŁ
11	0,33		DÓŁ
1	0,43		DÓŁ

5.3. Analiza

Obie serie przeanalizowano pod kątem średniego czasu reakcji dla zadanego gestu i obliczono odchylenie standardowe.

Tabela 5.4 Średni czas reakcji oraz odchylenie standardowe zebranych wyników dla serii I (opracowanie własne)

Joint 1	Joint 3	Śr. Czas	Odchylenie standardowe
LEWO		0,3202	0,0568
PRAWO		0,3586	0,0756
	DÓŁ	0,3799	0,1177
	GÓRA	0,3633	0,1000
LEWO	DÓŁ	0,4133	0,0983
PRAWO	DÓŁ	0,3702	0,0899
LEWO	GÓRA	0,4100	0,1095
PRAWO	GÓRA	0,4500	0,1095

Śr. Czas	0,3756

Tabela 5.5 Średni czas reakcji oraz odchylenie standardowe zebranych wyników dla serii II (opracowanie własne)

Joint 1	Joint 3	Śr. Czas	Odchylenie standardowe
LEWO		0,4091	0,1364
PRAWO		0,2984	0,0009
	DÓŁ	0,3758	0,0972
	GÓRA	0,3313	0,0816
LEWO	DÓŁ	0,2980	0,0000
PRAWO	DÓŁ	0,3315	0,0821
LEWO	GÓRA	0,2980	0,0000
PRAWO	GÓRA	0,3376	0,0885

Śr. Czas	0,3406

Tabela 5.6 Średni czas reakcji oraz odchylenie standardowe zebranych wyników dla obu serii (opracowanie własne)

Joint 1	Joint 3	Śr. Czas	Odchylenie standardowe
LEWO		0,3623	0,1094
PRAWO		0,3335	0,0639
	DÓŁ	0,3779	0,1055
	GÓRA	0,3505	0,0914
LEWO	DÓŁ	0,3474	0,0850
PRAWO	DÓŁ	0,3491	0,0837
LEWO	GÓRA	0,3489	0,0907
PRAWO	GÓRA	0,3938	0,1110

Śr. Czas	0,3586

Otrzymane wyniki średniego czasu reakcji dla analizowanych gestów wskazują na stosunkowo szybkie i stabilne działanie systemu. Średnia całkowita wartość czasu reakcji (Tabela 5.6) wyniosła 0,3586 s, co świadczy o wysokiej responsywności opracowanego algorytmu detekcji oraz sterowania i potwierdza założoną wcześniej hipotezę o czasie reakcji w okolicach 0,5 s. Ogólny czas reakcji był dłuższy w przypadku bliższej odległości od kamery (Tabela 5.4) niż w dalszej odległości (Tabela 5.5).

Dodatkowo, przeprowadzono analizę zmienności danych, opartą na obliczeniu odchylenia standardowego, które utrzymywało się na poziomie 0,0965 s dla pierwszej serii oraz 0,0881 s dla drugiej serii. Tak niskie wartości odchylenia sugerują, że system reaguje w sposób przewidywalny i powtarzalny, co ma istotne znaczenie w kontekście jego zastosowania w

środowisku robotycznym wymagającym precyzji i niezawodności. Stabilność czasów reakcji może wynikać zarówno z dobrze dobranych parametrów sterowania ruchem, jak i efektywnego przetwarzania danych wejściowych pochodzących z detekcji gestów. Uzyskane rezultaty stanowią podstawę do pozytywnej oceny skuteczności proponowanego rozwiązania w zakresie interpretacji gestów oraz szybkiego przekładania ich na działania manipulatora.

Na podstawie przeprowadzonych eksperymentów stwierdzono, że system jest w stanie interpretować gesty użytkownika z wysoką skutecznością i działa z akceptowalnym opóźnieniem. 100% przypadków, w których gest został rozpoznany, zakończyło się poprawnym wykonaniem trajektorii przez manipulator. Opóźnienia czasowe były praktycznie niezauważalne dla użytkownika, co wskazuje na możliwość użycia systemu w kontekście sterowania w czasie rzeczywistym. Czas od momentu wykonania gestu do reakcji manipulatora był na tyle krótki, że interakcja była płynna i intuicyjna. System wykazał się dużą odpornością na zakłócenia i niewielkie błędy detekcji. W przypadku braku jasnego odczytu gestu (np. bliska odległość punktu kluczowego 10 i 1), trajektoria nie zostawała wysyłana do wykonania. Oznacza to, że zaprojektowana logika warunkowa (sprawdzająca relację punktów 1 i 10 lub 11) poprawnie reaguje w przypadku niepewnych danych wejściowych.

5.4. Wnioski

Na podstawie przeprowadzonych badań i analiz eksperymentalnych można sformułować szereg istotnych wniosków dotyczących skuteczności i przydatności zaprojektowanego systemu do sterowania ramieniem robota na podstawie gestów użytkownika. Przede wszystkim, system wykazał się bardzo wysoką dokładnością w interpretacji gestów, osiągając skuteczność rozpoznania na poziomie 100%. Tak wysoka wartość potwierdza trafność doboru metod przetwarzania obrazu, opartych na sieci YOLOv8 przystosowanej do detekcji punktów kluczowych ludzkiego ciała, oraz integracji tej detekcji z architekturą ROS 2 i kontrolerem trajektorii manipulatora. Detekcja gestów, oparta na porównywaniu współrzędnych punktów odpowiadających dłoniom, barkom i punktom na twarzy, pozwoliła na wyodrębnienie jednoznacznych sygnałów sterujących robotem.

Opóźnienie między odbiorem danych a reakcją manipulatora wynosiło średnio 0.36 sekundy, co mieści się w granicach akceptowalnych dla naturalnej i intuicyjnej interakcji człowieka z robotem. Z perspektywy użytkownika oznacza to brak zauważalnych opóźnień czy przestojów w odpowiedzi robota na wykonane gesty, co stanowi istotną zaletę w kontekście rzeczywistych zastosowań, szczególnie w interaktywnych środowiskach pracy. Modyfikacje kodu źródłowego umożliwiające rejestrację znaczników czasowych pozwoliły na precyzyjne zmierzenie tego parametru, a uzyskane dane potwierdzają, że system jest wystarczająco szybki, aby być wykorzystywany w dynamicznych aplikacjach.

Dodatkowo, zastosowanie mechanizmów zabezpieczających, takich jak wartości graniczne dla obrotu stawów robota, kontrola poprawności wykrycia wszystkich wymaganych punktów oraz warunków logicznych sterujących kierunkiem ruchu, skutecznie chroni system

przed błędnymi reakcjami, wynikającymi np. z chwilowego braku detekcji, złego ustawienia kamery lub błędnych danych wejściowych. W praktyce oznacza to, że robot nie reaguje nieprzewidywalnie lub przypadkowo, a jego ruchy są deterministyczne i bezpieczne. W ten sposób minimalizowane są ryzyka wynikające z nieprawidłowego działania systemu w środowisku, gdzie przebywa człowiek.

System wykazał się również wysoką odpornością na zakłócenia w warunkach testowych, obejmujących różne pozycje operatora, kąty widzenia kamery oraz zmiany odległości operatora od kamery. W żadnym z przypadków nie odnotowano niekontrolowanych lub błędnych ruchów robota, które mogłyby wskazywać na niestabilność działania algorytmu. Świadczy to o dobrze zaprojektowanej architekturze komunikacyjnej ROS 2 oraz skutecznej synchronizacji węzłów odpowiedzialnych za odbiór danych, ich interpretację oraz generowanie trajektorii sterujących. W praktyce przekłada się to na stabilność i przewidywalność działania systemu w dłuższej perspektywie czasowej.

Analiza statystyczna wyników pokazała również, że system działa z dużą powtarzalnością – odchylenia standardowe czasu reakcji były niewielkie. Sugeruje to, że system może być skalowalny i przystosowany do współpracy z różnymi użytkownikami, bez konieczności personalizacji modelu lub dostrajania parametrów dla każdej osoby. Zwiększa to jego użyteczność w kontekście wdrażania w środowiskach produkcyjnych lub usługowych.

Na podstawie zebranych danych i przeprowadzonej analizy można stwierdzić, że połączenie zaawansowanych metod komputerowego rozpoznawania obrazu z technologiami ROS 2 i MoveIt pozwala na stworzenie responsywnego, precyzyjnego i bezpiecznego interfejsu do sterowania manipulatorem w czasie rzeczywistym. Projekt zrealizowany w ramach niniejszej pracy magisterskiej stanowi dowód koncepcji, który z powodzeniem może być rozwijany i adaptowany do bardziej złożonych scenariuszy interakcji człowiek–robot. Dalsze kierunki rozwoju mogą obejmować rozszerzenie systemu o dodatkowe gesty, integrację kamery głębi, wprowadzenie adaptacyjnych algorytmów uczenia maszynowego oraz testy z udziałem wielu użytkowników i w zmiennych warunkach środowiskowych.

5.5. Możliwości poprawy systemu

Pomimo uzyskanych zadowalających wyników, istnieje szereg możliwości dalszego rozwoju systemu w celu zwiększenia jego niezawodności i adaptacji do bardziej wymagających środowisk pracy. Przede wszystkim, implementacja mechanizmów śledzenia ruchu (ang. tracking) pozwoliłaby na redukcję błędów chwilowej utraty punktów kluczowych przez YOLO, szczególnie w przypadku dynamicznych gestów lub wyjścia punktu poza pole widzenia kamery. Dodatkowo, rozszerzenie analizy o dane zawierające głębię obrazu umożliwiłoby poprawne rozpoznanie pozycji rąk również w przestrzeni trójwymiarowej, co pozwoliłoby na dokładniejsze mapowanie gestów na trajektorie robota.

Kolejnym kierunkiem poprawy jest zastosowanie uczenia maszynowego w warstwie decyzyjnej, np. poprzez klasyfikację gestów w oparciu o sekwencje czasowe z wykorzystaniem rekurencyjnych sieci neuronowych. Takie podejście zwiększyłoby odporność systemu na fluktuacje w danych i umożliwiłoby interpretację bardziej złożonych gestów niż tylko statycznych pozycji dłoni. W aspekcie komunikacyjnym możliwe jest również dodanie warstwy priorytetowania i weryfikacji poleceń, np. poprzez żądanie potwierdzenia gestu lub zastosowanie podwójnych gestów (gest + utrzymanie), co zmniejszyłoby ryzyko przypadkowego uruchomienia trajektorii.

W warstwie sterowania robota warto rozważyć zastosowanie dynamicznego planowania trajektorii z uwzględnieniem predykcji ruchu człowieka, co umożliwiłoby płynniejsze i bardziej naturalne reakcje systemu. Z punktu widzenia bezpieczeństwa, istotne byłoby również zaimplementowanie monitorowania otoczenia robota (np. z użyciem czujników LiDAR lub systemów wizyjnych) w celu natychmiastowego zatrzymania ruchu w razie wykrycia przeszkody lub zbliżenia się operatora do przestrzeni roboczej. Zastosowanie powyższych rozwiązań pozwoliłoby przekształcić obecny system w pełni funkcjonalne, bezpieczne i inteligentne środowisko współpracy człowieka z robotem.

6. PODSUMOWANIE

Podsumowując całość przeprowadzonej pracy magisterskiej, należy stwierdzić, że zrealizowano wszystkie zaplanowane cele i zadania badawcze, które obejmowały zarówno przegląd literatury oraz istniejących rozwiązań, jak również projekt, implementację, integrację oraz testowanie systemu umożliwiającego sterowanie robotem współpracującym na podstawie gestów człowieka. Praca została podzielona na pięć głównych etapów, z których każdy stanowił logiczne rozwinięcie poprzedniego i był niezbędny do osiągnięcia pełnej funkcjonalności i niezawodności zaprojektowanego rozwiązania.

Pierwszym etapem było przeprowadzenie przeglądu literatury oraz istniejących rozwiązań w zakresie rozpoznawania gestów i ich wykorzystania w systemach sterowania robotami. Analizie poddano aktualne trendy w dziedzinie Human-Robot Interaction, technologie detekcji pozycji człowieka oraz rozwiązania oparte na sztucznej inteligencji i głębokim uczeniu. Szczególną uwagę poświęcono systemom wykorzystującym sieci neuronowe do analizy obrazu w czasie rzeczywistym, takim jak OpenPose, MediaPipe oraz różne wersje modelu YOLO (You Only Look Once). Dokonano wyboru podejścia opartego na modelu YOLOv8, który – po odpowiednim dostosowaniu – okazał się najbardziej efektywny w kontekście detekcji punktów kluczowych ciała użytkownika w czasie rzeczywistym, z zachowaniem wysokiej wydajności obliczeniowej. Przegląd rozwiązań uwzględniał również dostępne platformy programistyczne, z których na potrzeby pracy wybrano ROS 2 (Robot Operating System 2), charakteryzujący się modularną architekturą, dużą społecznością oraz zgodnością z popularnymi narzędziami do symulacji i wizualizacji.

Drugim zadaniem było uruchomienie wybranego algorytmu rozpoznawania gestów w czasie rzeczywistym. W tym celu przygotowano środowisko programistyczne, bazujące na ROS 2 Humble oraz bibliotece PyTorch, który umożliwił integrację modelu YOLOv8 z systemem ROS poprzez odpowiednie węzły komunikacyjne. Zaimplementowano adapter przekształcający dane wyjściowe sieci detekcyjnej do formatu wiadomości ROS typu DetectionArray, zawierającego pozycje wybranych punktów kluczowych ciała (m.in. głowy, nadgarstków, łokci, barków). Przygotowano również kamerę RGB zgodną z systemem ROS, która dostarczała obraz w rozdzielczości i częstotliwości umożliwiającej płynną detekcję. Przeprowadzono testy jednostkowe komponentów wizji komputerowej, weryfikując jakość detekcji w różnych warunkach oświetleniowych oraz przy różnych odległościach operatora od kamery.

W kolejnym etapie pracy zrealizowano integrację systemu rozpoznawania gestów z systemem sterowania ramieniem robota współpracującego. Wykorzystano środowisko MoveIt oraz narzędzie RViz do konfiguracji modelu manipulatora, planowania trajektorii oraz wizualizacji ruchów robota. Na potrzeby sterowania napisano własny węzeł ROS o nazwie point_comparator, który odbierał dane detekcyjne, analizował pozycje punktów kluczowych i na tej podstawie wyznaczał nowe pozycje dwóch kluczowych przegubów ramienia robota. Zmiany pozycji odbywały się stopniowo (z krokiem 0,04 radiana), a ich wartość ograniczano do zadanych zakresów, co zapewniało bezpieczeństwo ruchu. Węzeł wykorzystywał również dane o aktualnym

stanie manipulatora z kontrolera JointTrajectoryController, umożliwiając śledzenie stanu wykonania trajektorii. Wprowadzono zabezpieczenia programowe zapobiegające wysyłaniu nadmiarowych poleceń oraz eliminujące fałszywe reakcje na niepełne lub błędne dane wejściowe.

Czwartym zadaniem było przeprowadzenie badań systemu w warunkach testowych. W tym celu dokonano modyfikacji kodu sterującego, umożliwiającej pomiar czasu od momentu detekcji gestu do momentu rozpoczęcia ruchu ramienia. Dokładność pomiaru pozwoliła na precyzyjną ocenę czasu reakcji systemu. Eksperymenty prowadzono w środowisku kontrolowanym, z zachowaniem stałych warunków oświetleniowych oraz ustalonego dystansu między operatorem a kamerą. W trakcie testów analizowano skuteczność interpretacji trzech podstawowych gestów: podnoszenia ręki, obniżania ręki oraz zatrzymania ruchu. Przeprowadzono testów w różnych konfiguracjach początkowych, co pozwoliło na wyciągnięcie wiarygodnych wniosków statystycznych. W ramach testów mierzono również częstość występowania fałszywych detekcji, stabilność pracy węzła oraz poprawność wykonania trajektorii przez manipulator.

Wyniki badań wskazały, że system charakteryzuje się wysoką dokładnością interpretacji gestów oraz niskim opóźnieniem reakcji (średnio 0,36 s). Dzięki wbudowanym mechanizmom zabezpieczającym system był odporny na przypadkowe zmiany pozycji ciała operatora oraz chwilowe zakłócenia w obrazie. Analiza jakościowa wykazała, że ruchy robota były płynne, proporcjonalne do gestów i łatwe do kontrolowania, co czyni system intuicyjnym i naturalnym w obsłudze. Co ważne, system nie wymagał stosowania markerów ani specjalnych ubrań – wystarczała zwykła kamera RGB. Uzyskane wyniki potwierdziły hipotezy badawcze mówiące, że możliwe jest stworzenie responsywnego, stabilnego i bezpiecznego systemu sterowania robotem na podstawie gestów rozpoznawanych w czasie rzeczywistym.

W ostatnim etapie pracy – wnioskowaniu – stwierdzono, że system może znaleźć praktyczne zastosowanie w środowiskach przemysłowych i usługowych, szczególnie tam, gdzie operatorzy potrzebują sterować robotem w sposób bezdotykowy, szybki i intuicyjny. Może on również służyć jako podstawa do dalszych prac badawczo-rozwojowych, takich jak rozszerzenie zestawu gestów, wprowadzenie interfejsów głosowych, adaptacyjne dostosowanie ruchów do indywidualnych cech operatora, czy zastosowanie kamer głębi dla zwiększenia precyzji. Dzięki otwartości platformy ROS 2 oraz modularności systemu, możliwa jest jego dalsza rozbudowa i adaptacja do zmiennych warunków rzeczywistego środowiska pracy.

Podsumowując, niniejsza praca magisterska udowodniła, że możliwe jest skuteczne połączenie technik rozpoznawania obrazu opartych na głębokim uczeniu z systemami sterowania robotami w architekturze ROS 2. Stworzony system spełnił wszystkie założenia projektowe i potwierdził swoją użyteczność w praktyce. Osiągnięte rezultaty otwierają nowe możliwości rozwoju interfejsów człowiek–robot, wspierających zarówno efektywność, jak i bezpieczeństwo pracy. Zrealizowane zadania, począwszy od przeglądu rozwiązań, poprzez uruchomienie

algorytmu detekcji, integrację z robotem, badania i analizę wyników, dowodzą, że obrany kierunek był zasadny, a końcowy rezultat w pełni zadowalający.

BIBLIOGRAFIA

[1] N. Corporation, *trt_pose: Real-Time Human Pose Estimation for Jetson Platform,* -: GitHub repository, 2019.

[2] Z. Cao, „Realtime multi-person 2d pose estimation using part affinity fields.," w *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[3] B. H. W. a. Y. W. Xiao, „Simple baselines for human pose estimation and tracking.," w *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[4] Ultralytics, *YOLOv8 by Ultralytics,* GitHub repository, 2023.

[5] V. V. K. V. S. S. Swapnil Patil, „Advances and perspectives in collaborative robotics: a review of key," *Discover Mechanical Engineering,* tom 2:13, nr 2:13, 2023.

[6] IFR, „Welcome to the presentation of World Robotics 2022," c/o VDMA Robotics + Automation, 60528 Frankfurt Main, Germany, 2022.

[7] IFR, „Welcome to the presentation of World Robotics 2024," c/o VDMA Robotics + Automation, 60528 Frankfurt Main, Germany, 2024.

[8] I. Maurtua, A. Ibarguren, J. Kildal, L. Susperregi i B. Sierra, „Human–robot collaboration in industrial applications: Safety, interaction and trust," *International Journal of Advanced Robotic Systems,* pp. 1-10, July-August 2017.

[9] A. M. Zanchettin, N. M. Ceriani, P. Rocco, H. Ding i B. Matthias, „Safety in human-robot collaborative manufacturing environments: metrics and control," *IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING,,* pp. 99-112, April 2015.

[10] „www.universal-robots.com," 15 02 2024. [Online]. Available: https://www.universal-robots.com/products/ur16-robot/.

[11] M. Peshkin i J. Colgate, „Cobot architecture,” *IEEE Transactions on Robotics and Automation,* tom 17, nr 4, pp. 377 - 390, August 2001.

[12] J. E. C. M. Peshkin, „Cobots,” *Industrial Robot,* tom 5, nr 26, pp. 335-341, 1999.

[13] ABB, „https://new.abb.com/products/robotics/robots/collaborative-robots/yumi/dual-arm,” ABB, [Online]. Available: https://global.abb/group/en . [Data uzyskania dostępu: 4 11 2024].

[14] P. Kohut i K. Skop, „Vision Systems for a UR5 Cobot on a Quality Control Robotic Station,” *Applied Sciences,* nr 14, 2024.

[15] Z. Z. J. H. S. C. B. C. M. Ü. X. Z. Xingyu Yang, „Automation of SME production with a Cobot system powered by learning-based vision,” Department of Mechanical and Production Engineering, Aarhus University, Katrinebjergvej 89 G-F, Aarhus, 8200, Denmark, 2023.

[16] M. Faszczewski, „https://iautomatyka.pl/miekka-robotyka-przyklady-zastosowan-soft-robotics/,” iAutomatyka, 9 12 2017. [Online]. Available: https://iautomatyka.pl. [Data uzyskania dostępu: 8 11 2024].

[17] O. Sokolov, V. Andrusyshyn, A. Iakovets i V. Ivanov, „Intelligent Human–Robot Interaction Assistant for Collaborative Robots,” *Electronics,* nr 14, 2025.

[18] G. A. S. K. N. A. M. Rajeswari Packianathan, „Machine Vision and Industrial Robotics in Manufacturing,” w *Machine Vision and Industrial Robotics in Manufacturing* , 2024, p. 15.

[19] S. M. ,. K. T. ,. O. T. ,. R. J. Moor Madis, „AI functionalities in cobot-based manufacturing for performance improvement in quality control application,” *Journal of Machine Engineering,* tom 24, pp. 44-55, 2024.

[20] A. A. L. A. Y. A. F. A. H. H. A. A. Y. Z. Mohamad Halwani, „A novel vision-based multi-functional sensor for normality and position measurements in precise robotic manufacturing,” *Precision Engineering,* tom 88, pp. 367-381, 2024.

[21] A. S. C. d. S. A. e. a. Santos, „Integration of Artificial Vision and Image Processing into a Pick and Place Collaborative Robotic System,” *J Intell Robot Syst,* tom 110, May 2024.

[22] D. N. Yenjai N, „Optimizing pick-place operations: Leveraging k-means for visual object localization and decision-making in collaborative robots.,” *J Appl Res Sci Tech,* tom 23, 2024.

[23] KUKA, „https://www.kuka.com/en-de/industries/health-care/kuka-medical-robotics/lbr-med,” KUKA, [Online]. Available: https://www.kuka.com. [Data uzyskania dostępu: 10 10 2024].

[24] Agrobot, „https://www.agrobot.com/e-series,” Agrobot, [Online]. Available: https://www.agrobot.com. [Data uzyskania dostępu: 27 11 2024].

[25] Universal Robots, „https://www.universal-robots.com/pl/produkty/robot-ur10e/,” Universal Robots, [Online]. Available: https://www.universal-robots.com. [Data uzyskania dostępu: 20 9 2024].

[26] ABB, „https://new.abb.com/news/pl/detail/27516/abb-przedstawia-yumir-pierwszego-na-swiecie-dwuramiennego-robota-ktory-naprawde-wspolpracuje-z-czlowiekiem,” ABB, 14 4 2015. [Online]. Available: https://new.abb.com. [Data uzyskania dostępu: 21 9 2024].

[27] Blue River, „https://www.bluerivertechnology.com/our-mission/,” Blue River, [Online]. Available: https://www.bluerivertechnology.com. [Data uzyskania dostępu: 3 12 2024].

[28] Figure, „https://www.figure.ai,” Figure, 20 02 2025. [Online]. Available: https://www.figure.ai/news/helix. [Data uzyskania dostępu: 06 2025].

[29] D. N.-A. Andrea Cherubini, „Sensor-Based Control for Collaborative Robots: Fundamentals, Challenges, and Opportunities,” *Frontiers in Neurorobotics,* tom 14, 07 January 2021.

[30] A. P. R. C. A. L. A. F. P. Cherubini, „Tytuł artykułu: Collaborative manufacturing with physical human-robot interaction,” *Czasopismo: Robotics and Computer-Integrated Manufacturing,* tom 40, p. 1–13, 2016.

[31] G. W. H. G. C. Z. Xinghao Chen, „Pose guided structured region ensemble network for cascaded hand pose estimation,” *Neurocomputing,* tom 395, pp. 138-149, 28 June 2020.

[32] K. Simonyan i A. Zisserman, „Very Deep Convolutional Networks for Large-Scale Image Recognition,” *arXiv,* pp. 88-113, 4 Sep 2014.

[33] G. H. T. S. S. -E. W. a. Y. S. Z. Cao, „OpenPose: Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields,” *IEEE Transactions on Pattern Analysis and Machine Intelligence,* tom 43, nr 1, pp. 172-186, 1 Jan 2021.

[34] J. Redmon, S. Divvala, R. Girshick i A. Farhadi, „You Only Look Once: Unified, Real-Time Object Detection,” w *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, 2016.

[35] L. M. Z. C. Y. Y. Jing Qi, „Computer vision-based hand gesture recognition for human-robot interaction: a review.,” *Complex & Intelligent Systems,* pp. 1581-1606, 19 July 2023.

[36] J. T. H. N. C. M. E. U. M. H. Camillo Lugaresi, „MediaPipe: A Framework for Perceiving and Processing Reality,” w *Third Workshop on Computer Vision for AR/VR at IEEE Computer Vision and Pattern Recognition (CVPR)*, Long Beach, California, 2019.

[37] T. S. R. C. V. R. R. Mupparaju Sohan, „A Review on YOLOv8 and Its Advancements,” w *Algorithms for Intelligent Systems Data Intelligence and Cognitive Informatics*, Springer Nature Singapore , 2024, pp. 529-545.

[38] R. A. Güler, N. Neverova i I. Kokkinos, „DensePose: Dense Human Pose Estimation in the Wild,” w *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, UT, USA, 2018.

[39] D. Oved, „https://medium.com,” TensorFlow, 07 05 2018. [Online]. Available: https://medium.com/tensorflow/real-time-human-pose-estimation-in-the-browser-with-tensorflow-js-7dd0bc881cd5. [Data uzyskania dostępu: 28 10 2024].

[40] T. M. B. W. R. G. Steve Macenski, „Robot Operating System 2: Design, architecture, and uses in the wild,” *Science Robotics,* tom 7, pp. 66-84, 11 May 2022.

[41] A. J. Lentin Joseph, "Robot Operating System for Absolute Beginners" Getting Started with Ubuntu Linux for Robotics, Berkely, CA, United States: Apress, 2018.

[42] H. Robotics, „https://hanwharobotics.com," [Online]. Available: https://hanwharobotics.com/robot/hcr-3a. [Data uzyskania dostępu: 08 12 2024].

[43] © 2019 Hanwha Precision Machinery Co., *HCR-3 Collaborative Robot User Manual, V 2.001,* Hanwha Precision Machinery R&D Center, 6, Pangyo-ro 319 beon-gil, Bundang-gu, Seongnam-si, Gyeonggi-do (13488), Republic of Korea, Aug 2019.

[44] Corobotics, „https://corobotics.pl," [Online]. Available: https://corobotics.pl/produkty/oprogramowanie/rodi-off-line/. [Data uzyskania dostępu: 08 12 2024].

[45] Intel® RealSense™, „https://www.intelrealsense.com," [Online]. Available: https://www.intelrealsense.com/depth-camera-d415/. [Data uzyskania dostępu: 08 12 2024].

[46] Intel®RealSense™, *Intel® RealSenseTM D400 Series Product Family,* Intel Corporation, Jan 2019.

[47] B. Aderinola, „https://www.theconstruct.ai," ROS Tutorials, 03 09 2018. [Online]. Available: https://www.theconstruct.ai/ros-5-mins-034-ros-action/. [Data uzyskania dostępu: 15 10 2024].

[48] T.-Y. L. e. al., „https://cocodataset.org," Common Objects in Context, [Online]. Available: https://cocodataset.org/#home. [Data uzyskania dostępu: 18 11 2024].

ZAŁĄCZNIKI

1. Kod źródłowy

```
import rclpy

import time

from rclpy.node import Node

from yolo_msgs.msg import DetectionArray

from trajectory_msgs.msg import JointTrajectory, JointTrajectoryPoint

from control_msgs.msg import JointTrajectoryControllerState


class PointComparator(Node):


    def __init__(self):

        super().__init__('point_comparator')


        self.timer = self.create_timer(0.1, self.timer_callback)  # co 0.1 sekundy


        self.subscription = self.create_subscription(

            DetectionArray,

            '/yolo/tracking',

            self.listener_callback,

            10)


        self.state_sub = self.create_subscription(

            JointTrajectoryControllerState,

            '/joint_trajectory_controller/state',

            self.state_callback,

            10)
```

```
        self.trajectory_pub = self.create_publisher(
            JointTrajectory,
            '/joint_trajectory_controller/joint_trajectory',
            10)

        self.executing = False
        self.new_trajectory_needed = False
        self.current_rad_joint_3 = 0.0
        self.current_rad_joint_1 = 0.0
        self.step = 0.04  # 0.002  - za wolno
        self.min_rad = -2.5
        self.max_rad = 2.5
        self.last_requested_rad_joint_3 = self.current_rad_joint_3
        self.last_requested_rad_joint_1 = self.current_rad_joint_1

        self.joint_names = ['joint_1', 'joint_2', 'joint_3', 'joint_4', 'joint_5', 'joint_6']

        self.last_sent_time = None
        self.log_file_path = '/home/student/STUDENT/ros2_ws/trajectory_log.txt'

        self.get_logger().info("PointComparator node started")

    def listener_callback(self, msg: DetectionArray):
        point_1_y = None
        point_10_y = None
        point_6_y=None
        point_11_y = None
        point_7_y=None
```

```
for detection in msg.detections:
    for keypoint in detection.keypoints.data:
        if keypoint.id == 1:
            point_1_y = keypoint.point.y
        elif keypoint.id == 10:
            point_10_y = keypoint.point.y
        elif keypoint.id == 6:
            point_6_y = keypoint.point.y
        elif keypoint.id == 11:
            point_11_y = keypoint.point.y
        elif keypoint.id == 7:
            point_7_y = keypoint.point.y


 #dla lewej ręki


if point_1_y is not None and point_10_y is not None and point_6_y is not None:
    if point_10_y < point_1_y:
        self.get_logger().info("ID=10 jest WYŻEJ niż ID=1 — PODNOSZĘ")
        #new_z = min(self.current_rad_joint_3 + self.step, self.max_rad)
        new_z = max(self.current_rad_joint_3 - self.step, self.min_rad)
        if not self.executing and abs(new_z - self.last_requested_rad_joint_3) > 1e-4:
            self.current_rad_joint_3 = new_z
            self.new_trajectory_needed = True


    if point_10_y > point_1_y and point_10_y < point_6_y:
       self.get_logger().info("ID=10 jest NIŻEJ niż ID=1 i wyżej niż ID=6 — OBNIŻAM")
        new_z = min(self.current_rad_joint_3 + self.step, self.max_rad)
```

```
        #new_z = max(self.current_rad_joint_3 - self.step, self.min_rad)
        if not self.executing and abs(new_z - self.last_requested_rad_joint_3) > 1e-4:
            self.current_rad_joint_3 = new_z
            self.new_trajectory_needed = True

    elif point_10_y > point_6_y:
        self.get_logger().info("ID=10 jest NIŻEJ niż ID=6 — ZATRZYMUJĘ")
else:
    self.get_logger().info("Nie znaleziono punktów (ID=1 i ID=10 i ID=6)")



#dla prawej ręki

if point_1_y is not None and point_11_y is not None and point_7_y is not None:
    if point_11_y < point_1_y:
        self.get_logger().info("ID=11 jest WYŻEJ niż ID=1 — PRAWO")
        #new_z = min(self.current_rad_joint_3 + self.step, self.max_rad)
        new_z = max(self.current_rad_joint_1 - self.step, self.min_rad)
        if not self.executing and abs(new_z - self.last_requested_rad_joint_1) > 1e-4:
            self.current_rad_joint_1 = new_z
            self.new_trajectory_needed = True

    if point_11_y > point_1_y and point_11_y < point_7_y:
        self.get_logger().info("ID=11 jest NIŻEJ niż ID=1 i wyżej niż ID=7 — LEWO")
        new_z = min(self.current_rad_joint_1 + self.step, self.max_rad)
        #new_z = max(self.current_rad_joint_3 - self.step, self.min_rad)
        if not self.executing and abs(new_z - self.last_requested_rad_joint_1) > 1e-4:
```

```
                self.current_rad_joint_1 = new_z
                self.new_trajectory_needed = True

        elif point_11_y > point_7_y:
            self.get_logger().info("ID=11 jest NIŻEJ niż ID=7 — ZATRZYMUJĘ")
    else:
        self.get_logger().info("Nie znaleziono punktów (ID=1 i ID=11 i ID=7)")



def send_trajectory(self):
    traj = JointTrajectory()
    traj.joint_names = self.joint_names

    point = JointTrajectoryPoint()
    point.positions = [self.current_rad_joint_1, 0.0, self.current_rad_joint_3, 0.0, 0.0, 0.0]
    #point.time_from_start.sec = 1  # czas wykonania trajektorii

    traj.points.append(point)

    self.trajectory_pub.publish(traj)
    self.executing = True
    self.last_requested_rad_joint_3 = self.current_rad_joint_3
    self.last_requested_rad_joint_1 = self.current_rad_joint_1
    self.last_sent_time = time.time()

    self.get_logger().info(f"Wysłano trajektorię dla joint_3={self.current_rad_joint_3:.4f}")
    self.get_logger().info(f"Wysłano trajektorię dla joint_1={self.current_rad_joint_1:.4f}")
```

```
    def state_callback(self, msg: JointTrajectoryControllerState):
        actual = msg.actual.positions
        desired = msg.desired.positions

        if len(actual) != len(desired):
            return

        tolerance = 0.01  # radiany

        # Czy wszystkie jointy osiągnęły pozycję docelową?
        if all(abs(a - d) < tolerance for a, d in zip(actual, desired)):
            if self.executing:
                elapsed = time.time() - self.last_sent_time if self.last_sent_time else -1
                self.get_logger().info(f"Trajektoria zakończona po {elapsed:.3f} sek.")

                with open(self.log_file_path, 'a') as f:
                    f.write(f"Czas: {elapsed:.3f} s | joint_1={self.current_rad_joint_1:.4f}, joint_3={self.current_rad_joint_3:.4f}\n")

                self.executing = False
            else:
                self.executing = True

    def timer_callback(self):
        if self.new_trajectory_needed and not self.executing:
            self.send_trajectory()
            self.new_trajectory_needed = False
```

```
def main(args=None):
    rclpy.init(args=args)
    node = PointComparator()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```