RoboMVP

Instrukcja obsługi i opis systemu

Robot Humanoidalny Unitree G1 EDU · ROS2 Humble · Ubuntu 22.04

Wersja dokumentu: 1.0 · Projekt: RoboMVP v0.2.0
Politechnika Rzeszowska / Laboratorium Robotyki

1. Wprowadzenie i cel projektu

RoboMVP to minimalna, w pełni funkcjonalna aplikacja ROS2 demonstrujaca scenariusz manipulacji dla robota humanoidalnego Unitree G1 EDU. Nazwa MVP pochodzi od "Minimum Viable Product" – projektu, który robi dokładnie to co potrzeba, bez zbędnej złożoności.

Celem demonstracji jest wykonanie przez robota autonomicznej sekwencji: podejście do stołu, wykrycie pudełka przez markery AprilTag, podniesienie go, obrót o 180°, przejście do drugiego stołu i odłożenie pudełka. Cały scenariusz jest deterministyczny – robot nie "myśli", wykonuje zaprogramowaną choreografię, korygowaną przez wizję komputerową.

1.1. Co odróżnia tę wersję od poprzedniej?

Wersja 0.2.0 dodaje pełną komunikację ROS2, której brakowało w MVP:

- Action Server dla sekwencji ruchów (/robomvp/manipulation_task) – klient widzi postęp, może anulować
- Serwisy operatorskie: /robomvp/pause, /robomvp/emergency_stop, /robomvp/reset
- Odometria dead-reckoning – pozycja robota dostępna w nav_msgs/Odometry i TF
- Statyczne transformacje TF – drzewo układów współrzędnych dla RViz i Nav2
- Diagnostyki systemu – DiagnosticArray co 1s dla rqt_robot_monitor
- Węzeł teleop – ręczne sterowanie przez /cmd_vel (joystick, klawiatura)

2. Sprzęt i wymagania systemowe

2.1. Robot

Parametr	Wartość
Model	Unitree G1 EDU
Stopnie swobody	43 DOF (nogi 12 DOF, ramiona 12 DOF, ręce, głowa)
Masa	~35 kg
Połączenie	Ethernet (RJ45) – interfejs eth0 / enp3s0
SDK	unitree_sdk2py (DDS przez CycloneDDS)
Kamery	2× USB (ciała: 640×480, głowy: 1280×720)

2.2. Komputer sterujący

Wymaganie	Specyfikacja
System operacyjny	Ubuntu 22.04 LTS (Jammy Jellyfish)
ROS2	Humble Hawksbill
Python	3.10+
Karta sieciowa	1× Ethernet do robota + opcjonalnie WiFi do komputera
RAM	min. 4 GB (zalecane 8 GB)
CPU	x86_64, min. 4 rdzenie (ROS2 + OpenCV)

2.3. Środowisko sceny

- Dwa stoły o wysokości ~0.7–0.8 m, ustawione naprzeciwko siebie w odległości ~2.5 m
- Pudełko (~20×20×15 cm) z przyklejonym markerem AprilTag ID=10 (tag36h11, 10 cm)
- Marker ID=21 na stole startowym, ID=22 na stole docelowym, ID=30 na ścianie/słupku
- Oświetlenie: min. 300 lx, bez bezpośredniego refleksu słonecznego na markerach
- Przestrzeń robocza: min. 1×4 m wolnej podłogi wzdłuż trasy robota

3. Instalacja krok po kroku

3.1. ROS2 Humble

Jeśli ROS2 nie jest jeszcze zainstalowany, wykonaj:

```
# Konfiguracja repozytorium ROS2
sudo apt install software-properties-common curl -y
sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key \
  -o /usr/share/keyrings/ros-archive-keyring.gpg
echo "deb [arch=$(dpkg --print-architecture) signed-by=...] \
  http://packages.ros.org/ros2/ubuntu $(. /etc/os-release && echo $UBUNTU_CODENAME) main"
\
  | sudo tee /etc/apt/sources.list.d/ros2.list > /dev/null

# Instalacja ROS2 Humble (pełna wersja desktop)
sudo apt update && sudo apt install ros-humble-desktop -y

# Narzędzia budowania
sudo apt install python3-colcon-common-extensions python3-rosdep -y
sudo rosdep init && rosdep update
```

3.2. Zależności Pythona

```
# Biblioteki percepcji wizualnej
pip install opencv-python>=4.5 apriltag>=0.0.16 numpy>=1.21 pyyaml>=6.0

# Mostek ROS2-OpenCV
sudo apt install ros-humble-cv-bridge -y

# TF2 Python
sudo apt install ros-humble-tf2-ros ros-humble-tf2-geometry-msgs -y

# Unitree SDK (TYLKO na komputerze połączonym z robotem)
pip install unitree_sdk2py
```

3.3. Budowanie pakietu ROS2

```
cd ~/RoboMVP/ros2_ws

# Instalacja zależności ROS2 przez rosdep
rosdep install --from-paths src --ignore-src -r -y

# Budowanie pakietu
colcon build --symlink-install

# Załadowanie środowiska (dodaj do ~/.bashrc żeby działało permanentnie)
source install/setup.bash
echo "source ~/RoboMVP/ros2_ws/install/setup.bash" >> ~/.bashrc
```

> ⚠ **UWAGA:** Po każdej zmianie pliku .msg lub .action należy przebudować pakiet komendą colcon build.

4. Uruchomienie systemu

4.1. Sprawdzenie połączenia z robotem

Przed uruchomieniem upewnij się że robot jest włączony i połączony przez Ethernet:

```
# Sprawdź interfejs sieciowy
ip link show
# Powinieneś zobaczyć eth0 (lub enp3s0) w stanie UP

# Ping do robota (domyślny adres G1 EDU)
ping 192.168.123.161

# Sprawdź czy SDK widzi robota
python3 -c "from unitree_sdk2py.core.channel import ChannelFactoryInitialize; print('SDK
OK')"
```

4.2. Uruchomienie scenariusza

Podstawowe uruchomienie (bez robota – tryb demo):

```
cd ~/RoboMVP
source ros2_ws/install/setup.bash

# Uruchom cały system (tryb demo – bez fizycznego robota)
ros2 launch robomvp demo.launch.py \
  head_camera_device:=-1 \
  require_robot_connection:=false
```

Uruchomienie z robotem:

```
ros2 launch robomvp demo.launch.py \
  network_interface:=eth0 \
  body_camera_device:=0 \
  head_camera_device:=1 \
  require_robot_connection:=true
```

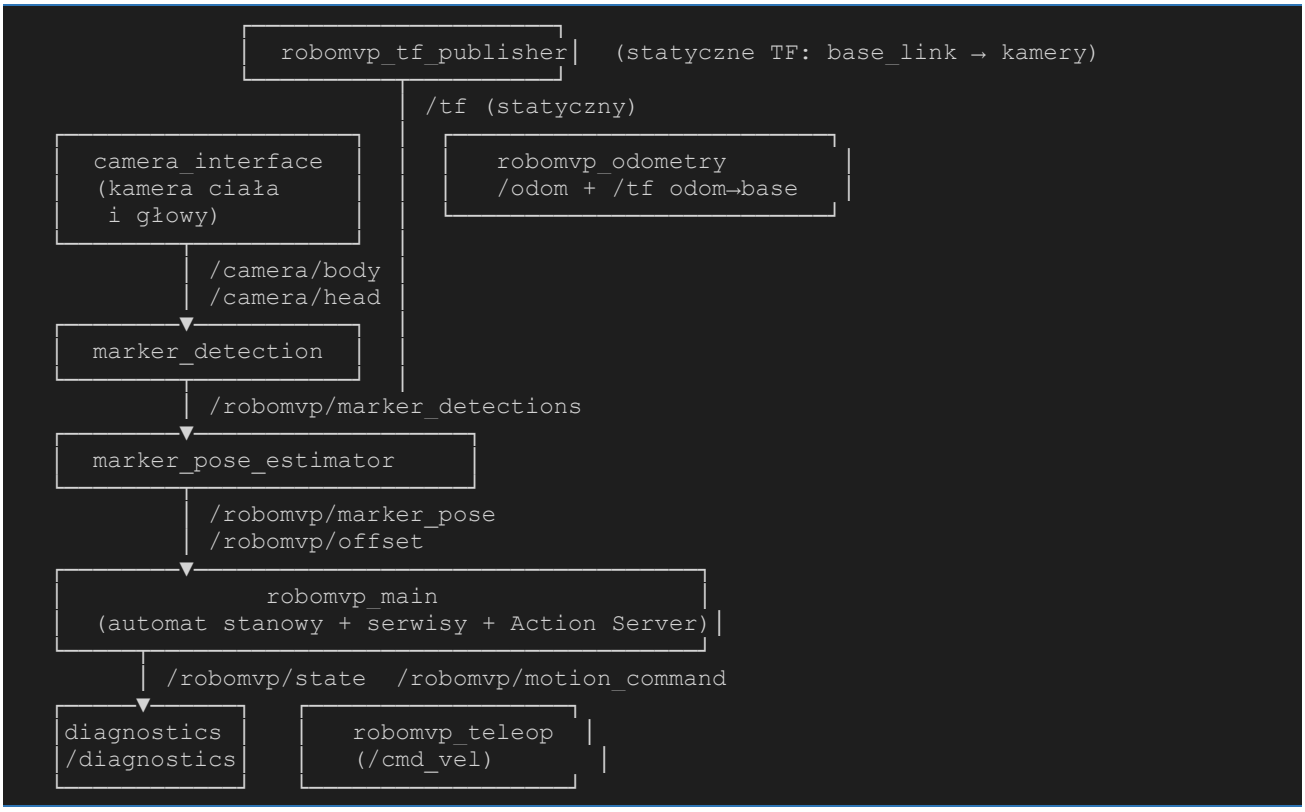
4.3. Parametry uruchomienia

Parametr	Opis i wartości domyślne
`marker_type`	Typ markerów: apriltag (domyślnie) lub qr
`network_interface`	Interfejs Ethernet do robota – eth0
`require_robot_connection`	Błąd gdy brak robota: true/false (domyślnie false)
`body_camera_device`	Indeks /dev/videoX kamery ciała – 0
`head_camera_device`	Indeks kamery głowy; -1 = użyj kamery ciała – -1
`teleop_enabled`	Aktywuj ręczne sterowanie /cmd_vel: true/false – false
`step_period`	Okres kroku automatu [s] – 1.0

5. Architektura systemu, węzły i komunikacja

5.1. Graf węzłów ROS2

System składa się z 8 węzłów ROS2 komunikujących się przez 15 tematów, 3 serwisy, 1 Action Server i drzewo TF. Każdy węzeł ma pojedynczą odpowiedzialność.



5.2. Pełna lista tematów ROS2

Temat	Typ / Opis
/camera/body/image_raw	sensor_msgs/Image – klatka 640×480 z kamery ciała (10 Hz)
/camera/head/image_raw	sensor_msgs/Image – klatka 1280×720 z kamery głowy (10 Hz)
/robomvp/marker_detections	robomvp/MarkerDetection – ID markera + pozycja pikselowa
/robomvp/marker_pose	robomvp/MarkerPose – pozycja 3D markera w układzie kamery
/robomvp/offset	robomvp/Offset – korekta dx/dy/dz dla wyrównania
/robomvp/state	robomvp/State – aktualny stan automatu (co 1s)
/robomvp/motion_command	std_msgs/String – nazwa aktualnie wykonywanej sekwencji

Temat	Typ / Opis
/odom	nav_msgs/Odometry – pozycja i prędkość (dead reckoning, 50 Hz)
/tf	tf2_msgs/TFMessage – drzewo transformacji układów współrzędnych
/tf_static	tf2_msgs/TFMessage – statyczne TF (kamera → base_link)
/diagnostics	diagnostic_msgs/DiagnosticArray – zdrowie komponentów (1 Hz)
/cmd_vel	geometry_msgs/Twist – ręczne komendy prędkości (teleop)
/robomvp/teleop_active	std_msgs/Bool – czy teleop jest aktualnie aktywny

5.3. Serwisy ROS2

Serwis	Typ / Opis
/robomvp/pause	std_srvs/SetBool – True = wstrzymaj, False = wznów scenariusz
/robomvp/emergency_stop	std_srvs/Trigger – natychmiastowe zatrzymanie + rozłączenie SDK
/robomvp/reset	std_srvs/Trigger – reset automatu do SEARCH_TABLE

5.4. Action Server

Action /robomvp/manipulation_task (typ: robomvp/ManipulationTask):

- Goal: sequence_name (nazwa sekwencji), apply_offset (czy korygować offsetem)
- Feedback: current_step, total_steps, progress (0.0–1.0), step_description, current_state
- Result: success, elapsed_seconds, final_state, message
- Obsługiwane sekwencje: approach_table, pick_box, rotate_180, walk_to_second_table, place_box, full_scenario

6. Szczegółowy opis węzłów

6.1. camera_interface

Odpowiada wyłącznie za pozyskiwanie klatek z kamer USB przez OpenCV VideoCapture i publikowanie ich jako sensor_msgs/Image. Kamera ciała jest wymagana – bez niej węzeł rzuca RuntimeError. Kamera głowy jest opcjonalna; gdy head_camera_device:=-1, obraz z kamery ciała jest duplikowany na temat głowy. Częstotliwość publikacji: 10 Hz (konfigurowane przez parametr publish_rate).

6.2. marker_detection

Subskrybuje obrazy z obu kamer i wykrywa markery AprilTag (rodzina tag36h11) lub kody QR (fallback gdy brak biblioteki apriltag). Każde wykrycie publikuje MarkerDetection z ID markera

i jego pozycją pikselową. Kluczowa poprawka względem oryginału: typ detektora śledzony przez osobną flagę _use_apriltag, nie przez hasattr() – eliminuje pomyłkę gdy cv2.QRCodeDetector ma metodę detect() o innej sygnaturze niż apriltag.Detector.

6.3. marker_pose_estimator

Przetwarza MarkerDetection na pozycję 3D (MarkerPose) metodą pinhole: głębokość obliczona z proporcji rozmiaru markera w pikselach do rzeczywistego, pozycje X/Y z odchylenia środka od centrum obrazu. Korekcja dystorsji soczewki przez cv2.undistortPoints. Publikuje też Offset (dx, dz = negacja x, y kamery) używany przez automat do wyrównania robota z pudełkiem.

6.4. robomvp_main (główny)

Centralny orkiestrator całego systemu. Utrzymuje automat stanowy (StateMachine), uruchamia sekwencje ruchów w osobnych wątkach (nie blokuje executora ROS2), obsługuje trzy serwisy operatorskie i Action Server. Używa MultiThreadedExecutor z ReentrantCallbackGroup – dzięki temu serwisy i timer automatu mogą działać równolegle nawet gdy Action Server wykonuje długą sekwencję.

6.5. robomvp_diagnostics

Publikuje DiagnosticArray co 1 sekundę na temat /diagnostics. Monitoruje "zdrowie" automatu stanowego (czy /robomvp/state jest aktualne) i pipeline ruchu (ostatnia komenda). Komponent STALE = brak wiadomości przez >3s. Widoczny w rqt_robot_monitor jako kolorowy panel statusu.

6.6. robomvp_odometry

Generuje nav_msgs/Odometry metodą dead reckoning – całkuje komendy prędkości po czasie (Euler, 50 Hz). Publikuje też transformację TF odom → base_link. Dokładność: ~5–10 cm na metr marszu. Docelowo: podłączyć EKF (robot_localization) łączący odometrię z IMU robota dla lepszej estymacji.

6.7. robomvp_tf_publisher

Publikuje statyczne transformacje TF: base_link → body_camera_frame i base_link → head_camera_frame. Pozycje kamer są parametryzowane (body_camera_x/y/z, head_camera_pitch_deg). Transformacja uwzględnia konwersję układu kamery optycznej (z = do przodu) na układ ROS (x = do przodu) zgodnie z REP-102.

6.8. robomvp_teleop

Subskrybuje /cmd_vel i przekazuje komendy do LocoClient.SetVelocity. Aktywny tylko gdy parametr enabled:=true I scenariusz nie jest w toku. Watchdog: jeśli przez 0.5s nie przyjdzie nowa komenda, robot zatrzymuje się (zabezpieczenie przed utratą połączenia z joystickiem). Typowe użycie: ros2 run teleop_twist_keyboard teleop_twist_keyboard.

7. Automat stanowy – co się dzieje krok po kroku

Automat stanowy (FSM – Finite State Machine) to "mózg" scenariusza. System może znajdować się dokładnie w jednym z ośmiu stanów. Przejście do następnego stanu następuje gdy spełniony jest konkretny warunek lub upłynął timeout awaryjny. Poniżej opisano każdy stan z jego warunkami wejścia, działaniem i warunkami wyjścia.

7.1. SEARCH_TABLE (Stan 0)

Cel: Robot szuka markera stołu startowego obracając lub skanując przestrzeń.

Wejście: Stan startowy po uruchomieniu lub resecie.

Warunek przejścia: Wykryto marker ID=21 (pickup_table_marker). Timeout awaryjny: 20 s → przejście do DETECT_MARKER mimo braku potwierdzenia.

Akcja w main_node: Brak sekwencji ruchu – robot stoi.

7.2. DETECT_MARKER (Stan 1)

Cel: Robot podchodzi do stołu i szuka markera pudełka.

Wejście: Przejście z SEARCH_TABLE.

Warunek przejścia: Wykryto marker ID=10 (box_marker_id) z ważną pozycją 3D.

Akcja: Sekwencja approach_table (4 waypoints, ~3 s). Uruchamiana w osobnym wątku (nie blokuje callbacków ROS2).

7.3. ALIGN_WITH_BOX (Stan 2)

Cel: Robot wyrównuje się boczne z pudełkiem na podstawie offsetu z kamery.

Warunek przejścia: |dx| < 0.05 m i |dz| < 0.05 m (alignment_threshold). Timeout awaryjny: 10 s.

Akcja: Brak sekwencji – automat czeka na zaktualizowanie offsetu przez marker_pose_estimator. W przyszłości: drobne ruchy korekcyjne w pętli.

7.4. PICK_BOX (Stan 3)

Cel: Robot podnosi pudełko z stołu.

Specjalność: Stan sekwencyjny – automat CZEKA w tym stanie dopóki main_node nie wywoła notify_sequence_done() po zakończeniu sekwencji. To kluczowa poprawka v0.2.0 – w poprzedniej wersji automat przechodził natychmiast, pomijając wykonanie sekwencji.

Akcja: Sekwencja pick_box (4 waypoints: ramię w dół → chwyt → uniesienie).

7.5. ROTATE_180 (Stan 4)

Cel: Robot obraca się o 180° trzymając pudełko.

Specjalność: Stan sekwencyjny – czeka na notify_sequence_done().

Akcja: Sekwencja rotate_180 (3 waypoints: 0° → 90° → 180°). Podzielone na 2 kroki po 90° dla bezpieczeństwa.

7.6. NAVIGATE_TO_TARGET_MARKER (Stan 5)

Cel: Robot idzie do drugiego stołu wykrywając marker ID=30 lub ID=22.

Warunek przejścia: Wykryto marker docelowy i głębokość z < 0.3 m (stop_distance_threshold). Timeout awaryjny: 25 s → FINISHED (awaryjne zakończenie bez odkładania).

Akcja: Sekwencja walk_to_second_table (4 waypoints, ~8 s marszu).

7.7. PLACE_BOX (Stan 6)

Cel: Robot odkłada pudełko na drugi stół.

Specjalność: Stan sekwencyjny – czeka na notify_sequence_done().

Akcja: Sekwencja place_box (4 waypoints: ramię w dół → odłożenie → ramię w górę).

7.8. FINISHED (Stan 7)

Cel: Scenariusz zakończony. Timer automatu zatrzymany.

Powrót: Wywołaj serwis /robomvp/reset aby zresetować do SEARCH_TABLE.

8. Sterowanie operatorskie – serwisy i Action

8.1. Wstrzymanie i wznowienie scenariusza

```
# Wstrzymaj scenariusz (robot stoi, automat nie robi kroków)
ros2 service call /robomvp/pause std_srvs/SetBool "{data: true}"

# Wznów scenariusz
ros2 service call /robomvp/pause std_srvs/SetBool "{data: false}"
```

8.2. Emergency stop

```
# Natychmiastowe zatrzymanie – zatrzymuje robota i rozłącza SDK
ros2 service call /robomvp/emergency_stop std_srvs/Trigger "{}"

# Po e-stop: zresetuj i ponownie połącz
ros2 service call /robomvp/reset std_srvs/Trigger "{}"
```

 WAŻNE: Emergency stop rozłącza SDK. Robot wejdzie w tryb Damp (serwomechanizmy rozluźnione). Przed wznowieniem upewnij się że robot stoi stabilnie.

8.3. Wykonanie pojedynczej sekwencji przez Action

```
# Wykonaj sekwencję pick_box z korekcją offsetu (i obserwuj feedback)
ros2 action send_goal --feedback \
  /robomvp/manipulation_task \
  robomvp/action/ManipulationTask \
  "{sequence_name: 'pick_box', apply_offset: true}"

# Anuluj aktualną sekwencję
ros2 action cancel <goal_id>
```

8.4. Ręczne sterowanie teleop

```
# W osobnym terminalu: uruchom system z teleop_enabled
ros2 launch robomvp demo.launch.py teleop_enabled:=true

# W kolejnym terminalu: sterowanie z klawiatury
ros2 run teleop_twist_keyboard teleop_twist_keyboard
# Klawisze: i=przód, ,=tył, j=lewo, l=prawo, u/o=skręt
```

9. Monitorowanie i debugowanie systemu

9.1. Podgląd stanu systemu

```
# Lista aktywnych węzłów
ros2 node list

# Lista aktywnych tematów z typami
ros2 topic list -t

# Aktualny stan automatu (na żywo)
ros2 topic echo /robomvp/state

# Diagnostyki (zdrowie komponentów)
ros2 topic echo /diagnostics

# Odometria
ros2 topic echo /odom --field pose.pose.position
```

9.2. Narzędzia graficzne

- rqt_graph: wizualizacja grafu węzłów i tematów (ros2 run rqt_graph rqt_graph)
- rqt_robot_monitor: panel diagnostyk z kolorowymi statusami komponentów
- RViz2: wizualizacja TF, odometrii i obrazów kamer (ros2 run rviz2 rviz2)
- rqt_plot: wykresy wartości numerycznych (np. offset dx/dy w czasie)

9.3. Częste problemy i rozwiązania

Problem	Rozwiązanie
Węzeł main rzuca RuntimeError przy starcie	Dodaj require_robot_connection:=false lub sprawdź kabel Ethernet
Marker nie jest wykrywany (brak /robomvp/marker_detections)	Sprawdź oświetlenie, rozmiar markera (10 cm), orientację (prostopadle do kamery)
Offset jest zawsze duży (nie maleje w ALIGN_WITH_BOX)	Skalibruj kamerę (camera.yaml: fx, fy, cx, cy) i sprawdź marker_size w scene.yaml
Robot nie reaguje na SetVelocity	Upewnij się że LocoClient.Start() zostało wywołane (tryb Sport Mode)
Emergency stop nie zatrzymuje robota fizycznie	Sprawdź połączenie sieciowe; e-stop wymaga działającego SDK do StopMove
colcon build nie znajduje config/*.yaml	Sprawdź ścieżkę w setup.py: os.path.dirname(__file__) musi wskazywać na setup.py
Diagnostyki pokazują STALE dla automatu	Węzeł main nie publikuje /robomvp/state – sprawdź log błędów węzła

10. Kalibracja kamer

Dokładność estymacji odległości markera zależy od poprawnych parametrów kamery. Domyślne wartości (fx=fy=600, cx=320, cy=240 dla kamery ciała) są przybliżone i mogą dawać błąd do 15% w szacowaniu odległości.

10.1. Kalibracja przez planszetę szachownicy (zalecane)

```
# Wydrukuj planszę 8×6 szachownicy (rozmiar kwadratu: 25 mm)
# Zbierz 15–20 zdjęć z różnych kątów

# Kalibracja przez ROS2 camera_calibration
sudo apt install ros-humble-camera-calibration -y
ros2 run camera_calibration cameracalibrator \
  --size 8x6 --square 0.025 \
  image:=/camera/body/image_raw \
  camera:=/camera/body
```

Skopiuj wynikowe wartości (fx, fy, cx, cy, k1, k2, p1, p2) do config/camera.yaml.

> **INFO:** Po kalibracji przebuduj pakiet (colcon build) aby nowe wartości zostały zainstalowane w share/robomvp/config/camera.yaml.

11. Słownik pojęć

Pojęcie	Wyjaśnienie
Action Server	Mechanizm ROS2 do długotrwałych zadań z feedbackiem w trakcie i możliwością anulowania
AprilTag	System markerów wizualnych (kwadratowe wzory bitowe) do lokalizacji obiektów
Dead reckoning	Szacowanie pozycji przez całkowanie prędkości – bez GPS, z narastającym błędem
DDS	Data Distribution Service – protokół komunikacji używany przez ROS2 i SDK Unitree
DOF	Degrees of Freedom – stopnie swobody (liczba niezależnych ruchów manipulatora)
FSM	Finite State Machine – automat stanowy; model systemu z skończoną liczbą stanów
LocoClient	Klasa SDK Unitree do sterowania lokomocją w Sport Mode (wysyła SetVelocity)
Kwaternion	Czterowymiarowa liczba zespolona reprezentująca obrót 3D (w, x, y, z)
Odometria	Szacowanie pozycji i orientacji robota na podstawie pomiarów ruchu
Pinhole camera	Model aparatu otworkowego: prosta relacja między pikselem, a punktem 3D w przestrzeni
Race condition	Błąd gdy dwa wątki jednocześnie modyfikują tę samą daną bez synchronizacji
REP-103	ROS Enhancement Proposal 103: standardowa konwencja układów współrzędnych w ROS
Sport Mode	Tryb wysokiego poziomu G1 EDU: robot sam zarządza krokami nóg na podstawie prędkości
TF2	Transform Library 2: system zarządzania układami współrzędnych w ROS2