



**Politechnika
Śląska**

Projekt
Bezzałogowe Obiekty Autonomiczne

Symulacja robota Unitree Go2 w środowisku ISSAC SIM

Skład sekcji: Filip Bazarnicki, Darian Bonk, Jakub Ligenza, Kacper Wach
Rok akademicki: 2024/25

Kierunek: Automatyka i Robotyka

Specjalizacja: Robotyka

Semestr: 1

1 Cel projektu

Celem projektu była instalacja środowiska ISSAC SIM oraz jego odpowiednia konfiguracja, przygotowanie cyfrowego bliźniaka robota Unitree Go2, przeprowadzenie przykładowych symulacji w ISAAC SIM oraz nawiązanie połączenia między nim, a środowiskiem ROS2.

2 Wstęp teoretyczny

2.1 Czym jest środowisko ISAAC SIM

NVIDIA Isaac Sim to zaawansowane środowisko symulacyjne oparte na silniku Omniverse, stworzone z myślą o rozwoju i testowaniu algorytmów dla robotyki. Umożliwia realistyczną symulację fizyki, czujników oraz środowisk 3D, co pozwala na szybkie prototypowanie i walidację systemów autonomicznych bez konieczności fizycznego dostępu do sprzętu. Symulator ten wspiera modelowanie robotów zgodne z formatem URDF (Unified Robot Description Format) oraz pozwala na integrację z popularnymi bibliotekami takimi jak ROS/ROS2, PyTorch, czy Isaac SDK. Dzięki realistycznemu odwzorowaniu działania sensorów (np. kamer RGB-D, LiDAR, IMU), możliwe jest testowanie algorytmów percepcji, lokalizacji, mapowania, planowania ruchu czy sterowania w warunkach zbliżonych do rzeczywistych. Środowisko to odgrywa kluczową rolę w symulacji „cyfrowych bliźniaków” robotów, co znaczaco obniża koszty i ryzyko związane z rzeczywistymi testami.

2.2 Czym jest środowisko ROS2

ROS 2 (Robot Operating System 2) to otwartoźródłowy system operacyjny dla robotów nowej generacji, zaprojektowany jako następca klasycznego ROS 1. Stanowi zaawansowaną platformę programistyczną do tworzenia, testowania i wdrażania oprogramowania dla systemów robotycznych, umożliwiając modularne projektowanie aplikacji z wykorzystaniem tzw. węzłów (nodes), które komunikują się ze sobą poprzez tematy (topics), usługi (services) oraz akcje (actions).

Jedną z kluczowych zmian w ROS 2 względem poprzednika jest zastosowanie standardu DDS (Data Distribution Service) jako warstwy komunikacyjnej, co zapewnia deterministyczne i wydajne przesyłanie danych między węzłami – również w sieciach rozproszonych.

ROS 2 współpracuje z popularnymi narzędziami robotycznymi i symulatorami (np. Gazebo, Isaac Sim, MoveIt 2), wspiera wiele języków programowania (C++, Python) i znajduje zastosowanie w robotyce przemysłowej, mobilnej, medycznej i badawczej.

2.3 Czym jest robot Unitree Go2

Unitree GO2 to zaawansowany, autonomiczny czteronożny robot mobilny (quadruped robot) opracowany przez firmę Unitree Robotics. Został zaprojektowany z myślą o wszechstronnych zastosowaniach w dziedzinach takich jak badania naukowe, eksploracja terenów trudnodostępnych, edukacja, patrolowanie oraz rozwój algorytmów sztucznej inteligencji i percepcji środowiska. Robot GO2 jest sterowany za pomocą systemu ROS 2, co umożliwia integrację z istniejącymi narzędziami i frameworkami robotycznymi. Dzięki zastosowaniu silników o wysokim momencie obrotowym i zaawansowanym algorytmom stabilizacji, GO2 potrafi poruszać się stabilnie w zmiennych warunkach terenowych.

3 Instalacja i konfiguracja środowiska ISSAC SIM oraz ROS2

W projekcie korzystaliśmy z systemu Ubuntu 22.04, wykorzystaliśmy system Linux ze względu na potrzebę wykorzystania środowiska ROS2 w projekcie, co jest prostsze w systemach linuxowych.

3.1 Instalacja środowiska ISAAC SIM

Ponieważ nie da się już pobrać ISAAC SIM bezpośrednio przy użyciu Omniverse Launcher, pobraliśmy ręcznie archiwum ze strony NVIDIA:

- Strona: <https://developer.nvidia.com/isaac-sim>
- Wersja: Isaac Sim 4.5.0

Po pobraniu rozpakowaliśmy plik i możliwe było już uruchomienie środowiska ISAAC SIM. Można to zrobić, wchodząc do katalogu, w którym umieszczone zostały rozpakowane pliki i wpisując ./isaac-sim.sh lub alternatywnie isaac-sim.selector.sh.

3.2 Instalacja środowiska ROS2

Źródłem wykonanej instalacji była oficjalna dokumentacja ROS 2 dla wersji Humble (Ubuntu Development Setup):

<https://docs.ros.org/en/humble/Installation/Alternatives/Ubuntu-Development-Setup.html>

3.2.1 Krok 1: Konfiguracja lokalizacji UTF-8

Na początku upewniliśmy się, że system ma skonfigurowane środowisko lokalizacyjne obsługujące UTF-8.

```
sudo locale-gen en_US en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LANG=en_US.UTF-8
```

Sprawdziliśmy, czy ustawienia lokalne są prawidłowe:

```
locale
```

3.2.2 Krok 2: Instalacja zależności systemowych

Zainstalowaliśmy wymagane pakiety narzędziowe i kompilacyjne:

```
sudo apt update && sudo apt install -y
build-essential cmake git wget curl
gnupg lsb-release python3-colcon-common-extensions
python3-pip python3-vcstool python3-rosdep
```

Zainicjowaliśmy narzędzie rosdep:

```
sudo rosdep init
rosdep update
```

3.2.3 Krok 3: Utworzenie przestrzeni roboczej i pobranie kodu ROS 2

Stworzyliśmy katalog roboczy i pobraliśmy plik repozytoriów:

```
mkdir -p ~/ros2_humble/src  
cd ~/ros2_humble  
wget https://raw.githubusercontent.com/ros2/ros2/humble/ros2.repos  
vcs import src < ros2.repos
```

3.2.4 Krok 4: Instalacja zależności pakietów ROS 2

Zainstalowaliśmy wszystkie zależności wymagane do komplikacji (pomijając rti-connext, którego nie potrzebujemy).

```
rosdep install -from-paths src -ignore-src -r -y  
-skip-keys "fastcdr rti-connext-dds-6.0.1 urdfdom_headers"
```

3.2.5 Krok 5: Kompilacja ROS 2

Rozpoczęliśmy komplikację całego środowiska z wykorzystaniem colcon:

```
cd ~/ros2_humble  
colcon build -symlink-install  
Ten proces trwał kilkanaście minut.
```

3.2.6 Krok 6: Konfiguracja środowiska

Po zakończeniu komplikacji dodaliśmy ROS 2 do zmiennych środowiskowych:

```
echo "source ~/ros2_humble/install/setup.bash" >> ~/.bashrc  
source ~/.bashrc
```

Dzięki temu komenda ros2 była dostępna z każdego terminala.

3.2.7 Weryfikacja działania ros2

Przeprowadziliśmy podstawowy test komunikacji między dwoma węzłami:

Terminal 1:

```
ros2 run demo_nodes_cpp talker
```

Terminal 2:

```
ros2 run demo_nodes_cpp listener
```

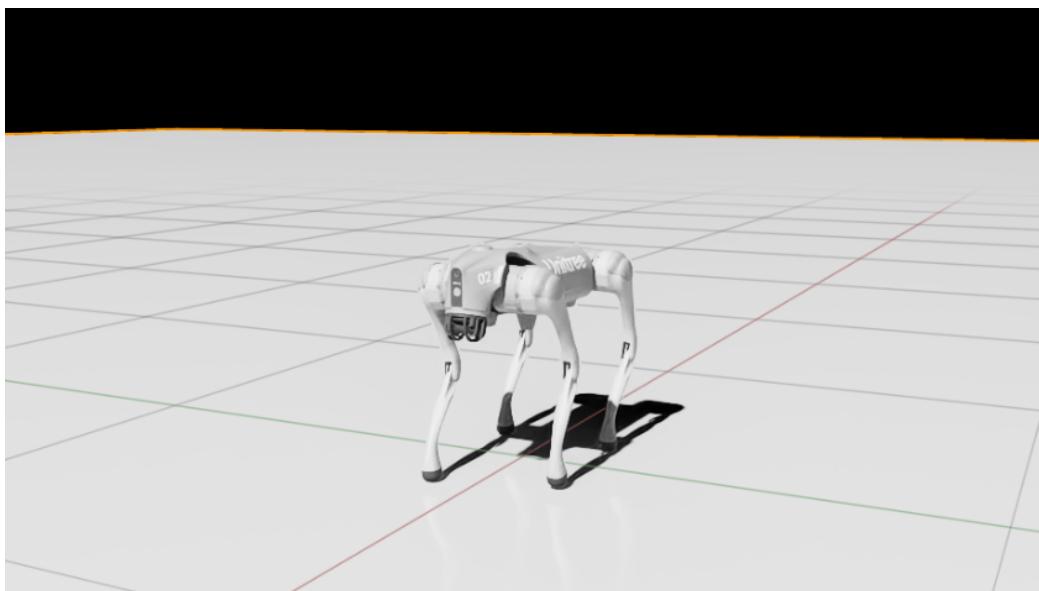
Terminal listener poprawnie odbierał wiadomości wysyłane przez talker, co potwierdziło, że środowisko działa.

4 Cyfrowy bliźniak robota Unitree Go2

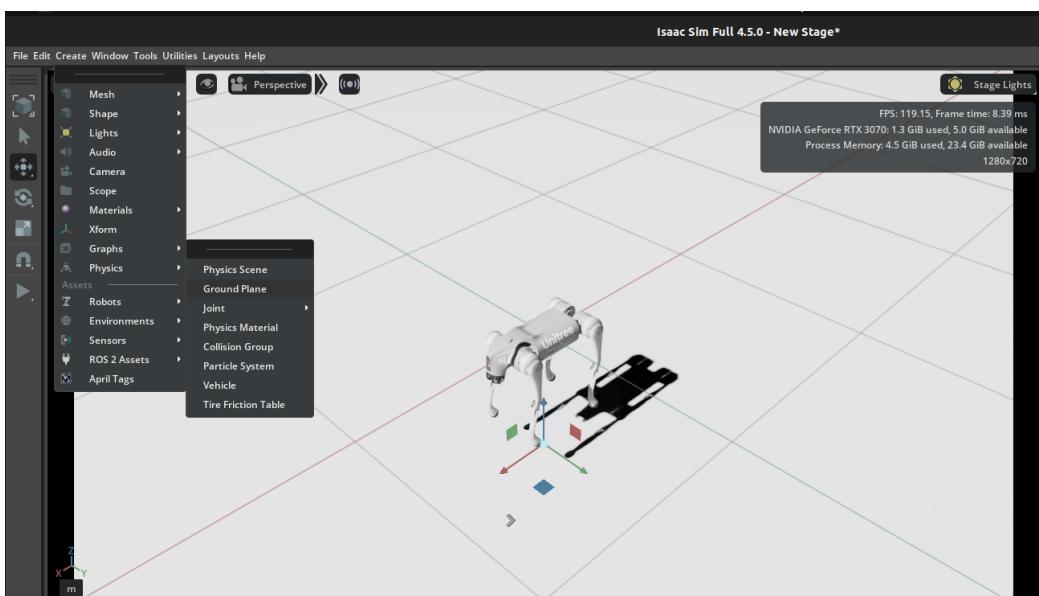
W środowisku ISSAC SIM znajduje się gotowy model cyfrowego bliźniaka robota Unitree Go2, w celu umieszczenia go na scenie otwartego programu należy odnaleźć go w oknie Isaac Sim Assets i dodać go, wciskając przycisk Load as Reference lub przeciągając jego ikonę na scenę. Widok okna Isaac Sim Assets przedstawiony jest na rysunku 1. Widok umieszczonego w symulacji robota przedstawiony jest na rysunku 2. Na rysunku 3 pokazane jest, jak dodać podłogę w scenie, należy wybrać w górnym pasku Create, następnie Physics, a na końcu wcisnąć Ground Plane.



Rys. 1: Biblioteka dostępnych robotów Isaac Sim Assets.



Rys. 2: Cyfrowy bliźniak robota Unitree Go2 w środowisku ISAAC SIM.



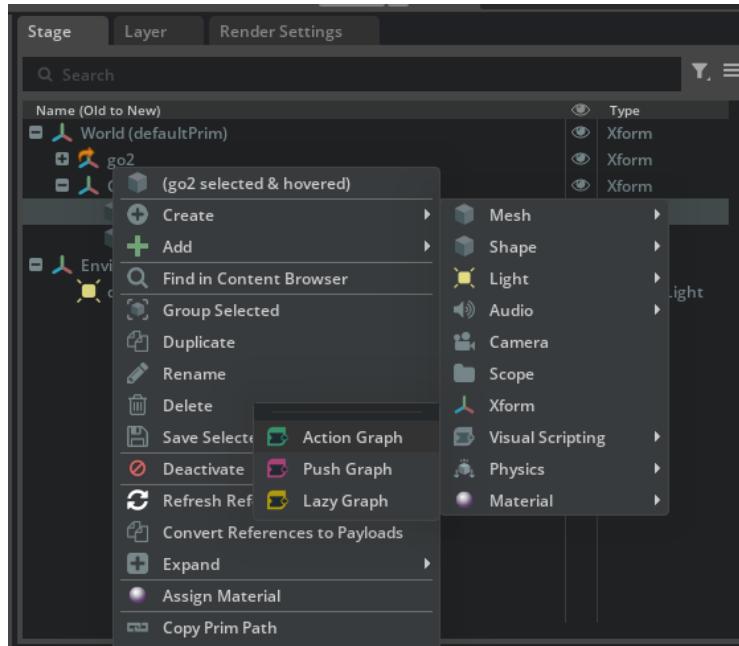
Rys. 3: Dodanie podłoża w ISSAC SIM.

5 Sterowania robotem przy pomocy środowiska ROS2

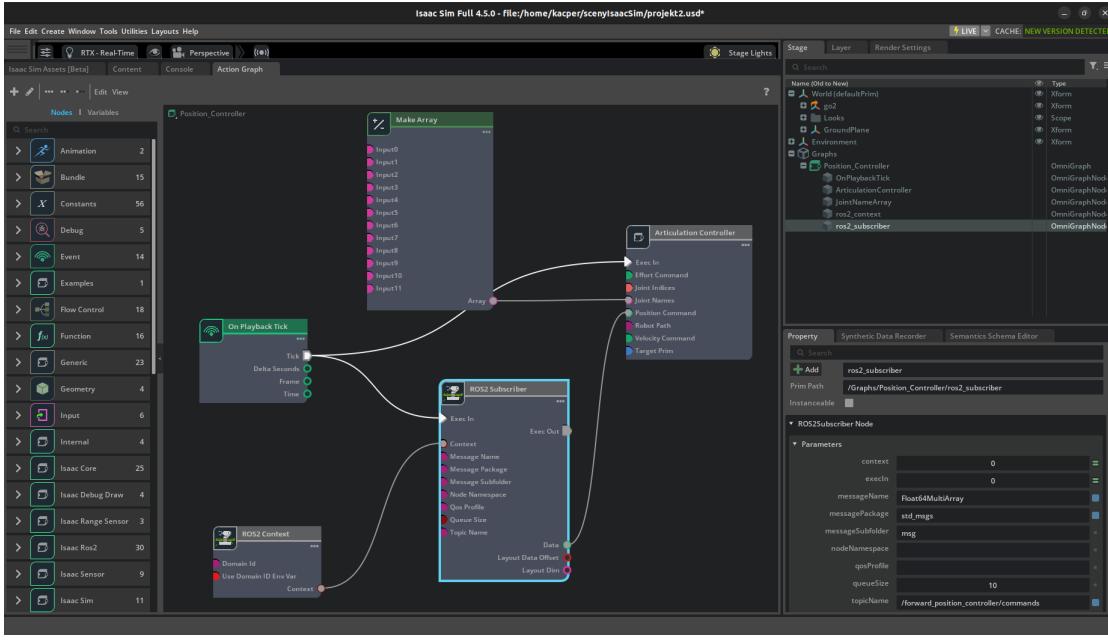
Wszystkie potrzebne pliki źródłowe do sterowania sterowania robotem znajdują się w linku poniżej.

<https://drive.google.com/drive/folders/1fdtBgbXUCxln3wdQgWUAGIBdBf-2Qv9z>

Żeby możliwe było sterowanie robotem Unitree Go2 przy pomocy środowiska ROS2, wymagane jest dodanie do projektu specjalnego grafu z instrukcjami w formie bloków (Action Graph). Na rysunku 4 pokazane jest jak dodać Action Graph, na rysunku 5 widoczne są potrzebne do sterowania robotem bloki oraz połączenia między nimi.



Rys. 4: Dodanie Action Graph do cyfrowego bliźniaka Unitree Go2



Rys. 5: Action Graph do nawiązania komunikacji między ROS2 oraz ISAAC SIM.

Blok On Playback Tick uruchamia się w każdej klatce symulacji i pozwala na wykonywanie się pozostałych bloków synchronicznie. Blok ROS2 Context inicjalizuje ROS2 w środowisku ISAAC SIM. Articulation Controller służy do przekazywania danych do poszczególnych członów robota. W bloku Make Array należy wstawić poszczególne nazwy członów robota, nazwy te można odczytać z obiektu robota go2. Blok Make Array przekazuje nazwy do bloku Articulation Controller po to żeby wiadomo było jakie człony mają być wysterowane na jaką wartość. Wstawione nazwy członów w bloku Make Array zostały pokazane na rysunku 6 ważne jest żeby kolejność nazw członów była taka sama jak na rysunku ze względu na fakt, że w takiej kolejności są odbierane z bloku ROS2Subscribe.

Blok ROS2Subscriber służy do otrzymywania danych z programu środowiska ROS2, tak zwanego subskrybowania danych, na rysunku 7 pokazano jak poprawnie skonfigurować blok ROS2Subscriber żeby dane były odpowiednio pobierane. Otrzymane dane są przesyłane do bloku Articulation Controller.

W celu uruchomienia programu sterującego, po wcześniejszym wypakowaniu potrzebnych plików, należy przejść do folderu `/quadruped_robot_ROS2` i w terminalu wpisać (żeby uruchomić terminal w danym folderze należy będąc w aplikacji pliki w danym folderze wcisnąć prawy przycisk myszy i ‘otwórz w terminalu’):

`colcon build`

Następnie w nowym terminalu wpisać:

```
source quadruped_robot_ROS2/install/setup.bash
ros2 launch robot_control robot_control.launch.py
```

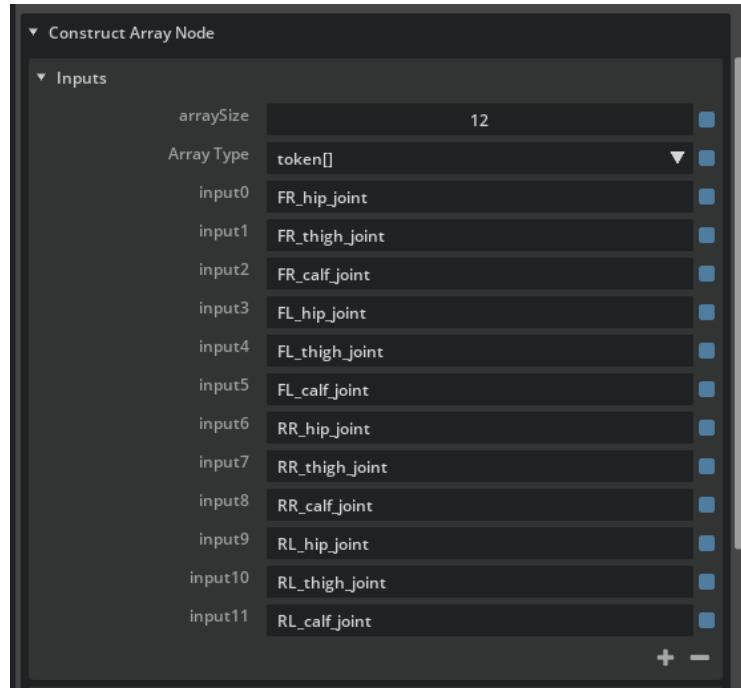
Po czym przechodzi się do folderu `/quadruped_robot_ROS2/UI` uruchamia się nowy terminal w tym folderze i wpisuje:

```
python3 controller.py
```

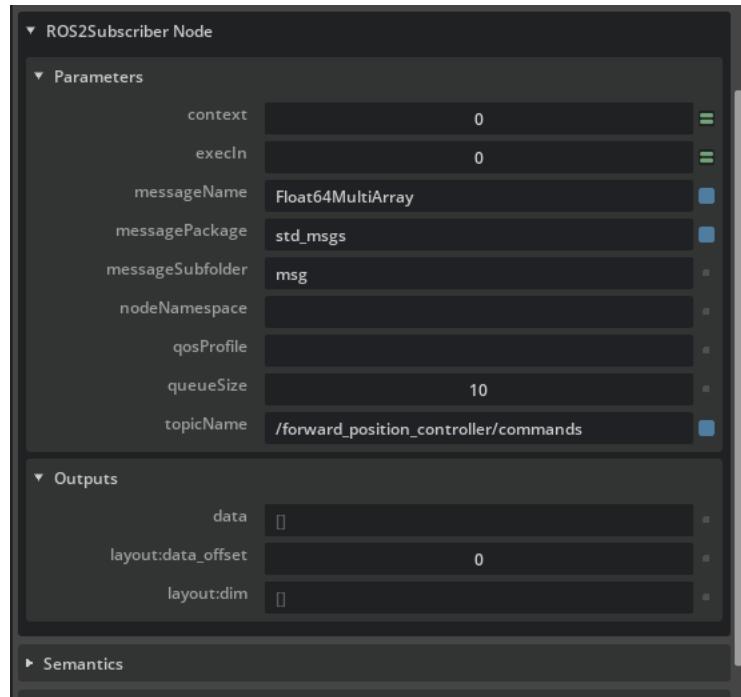
Na rysunku 8 oraz na rysunku 9 widoczne jest przykładowe sterowanie cyfrowym bliźniakiem robota Unitree Go2. Sterowanie odbywa się przy pomocy okna z kontrolerem widocznego na obu rysunkach, okno ma nazwę controller.

Do wykonania tej części projektu korzystaliśmy z poniższego filmu.

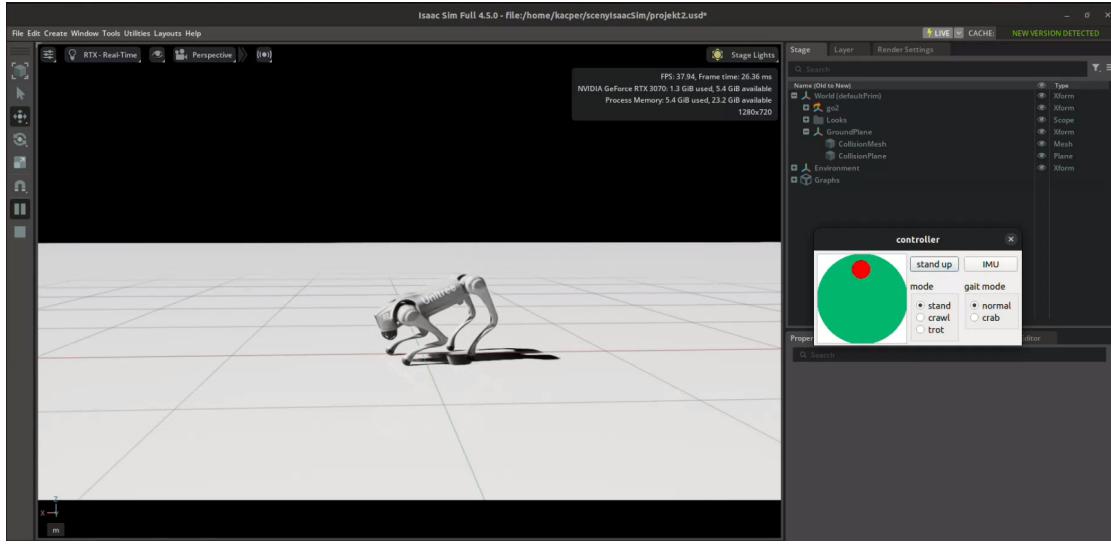
Link do filmu



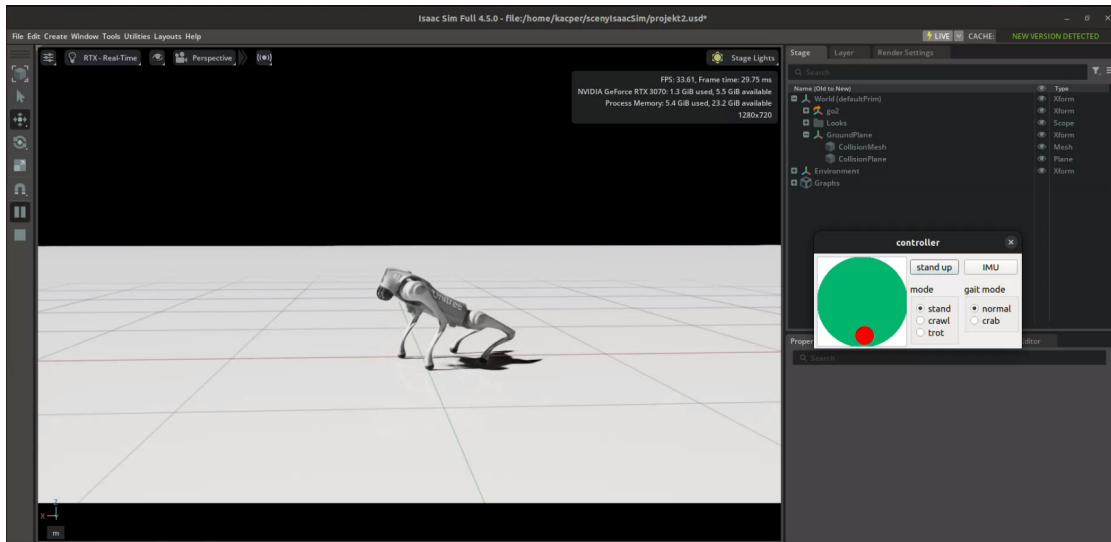
Rys. 6: Ustawienia bloku Make Array



Rys. 7: Ustawienia bloku ROS2 Subscriber



Rys. 8: Robot Unitree Go2 pochylający się do przodu.



Rys. 9: Robot Unitree Go2 pochylający się do tyłu.

6 Jazda robotem kołowym z użyciem ROS2 i teleop_twist_keyboard

Pakiet teleop_twist_keyboard umożliwia ręczne sterowanie robotem mobilnym w systemie ROS2 z użyciem klawiatury. Wysyła komunikaty typu geometry_msgs/msg/Twist na zadany temat (domyślnie /cmd_vel). Pozwala sterować prędkością liniową oraz kątową- co jest typowym sposobem poruszania się robotów mobilnych (np. typu differential drive).

Klawisze sterujące

Podstawowe sterowanie odbywa się z użyciem następujących klawiszy:

Klawisz Działanie

- i Do przodu
- k Zatrzymanie(zerowanie prędkości)
- , Do tyłu
- j Skręt w lewo(obrót w miejscu)
- l Skręt w prawo(obrót w miejscu)
- u Do przodu + w lewo
- o Do przodu + w prawo
- m Do tyłu + w lewo
- . Do tyłu + w prawo

Zmiana prędkości:

Klawisz Działanie

- q Zwiększa prędkość liniową i kątową
- z Zmniejsza prędkość liniową i kątową
- w Zwiększa tylko prędkość liniową
- x Zmniejsza tylko prędkość liniową
- e Zwiększa tylko prędkość kątową
- c Zmniejsza tylko prędkość kątową

Każde wciśnięcie odpowiedniego klawisza powoduje wysłanie wiadomości Twist na zadany temat, zgodnie z bieżącymi wartościami prędkości.

Instalacja pakietu

a) instalacja binarna zalecana

sudo apt update

sudo apt install ros-humble-teleop-twist-keyboard

b) Instalacja ze źródła

cd ~/ros2_ws/src

git clone https://github.com/ros2/teleop_twist_keyboard.git

cd ..

colcon build

source install/setup.bash

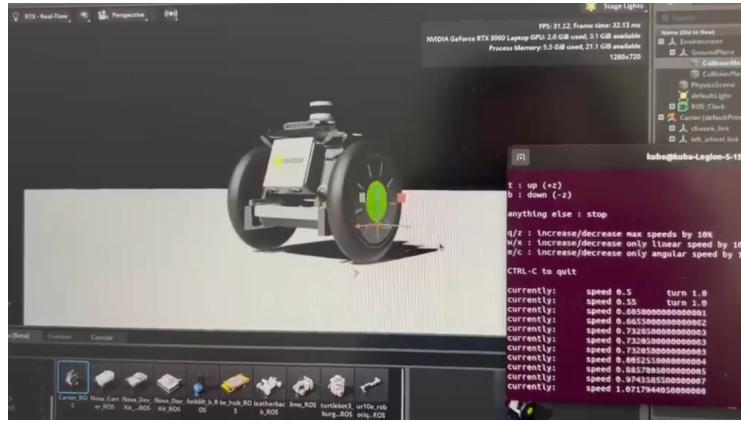
Uruchomienie

ros2 run teleop_twist_keyboard teleop_twist_keyboard

Do testów wykorzystano gotowy przykład robota mobilnego dostępnego w środowisku Isaac Sim 4.5.0, który został przygotowany do współpracy z systemem ROS2. Przykład ten zawiera wstępnie skonfigurowany zestaw węzłów (nodes) ROS2, które umożliwiają komunikację pomiędzy symulowanym robotem, a zewnętrznym środowiskiem ROS 2.

Robot ten subskrybuje wiadomości typu geometry_msgs/msg/Twist, publikowane na temacie /cmd_vel, który jest domyślnym kanałem komunikacyjnym wykorzystywanym przez pakiet teleop_twist_keyboard. Oz-

nacza to, że po uruchomieniu symulacji oraz włączeniu mostu ROS 2, komendy wysyłane z klawiatury mogą bezpośrednio sterować ruchem robota w symulacji.



Rys. 10: Robot kołowy w symulacji ISAAC SIM.

7 Przykładowe symulacje

W środowisku ISAAC SIM dostępnych jest wiele przykładowych symulacji do, których można uzyskać dostęp, wybierając w górnym pasku Window, następnie Examples, a na końcu Robotics Examples, pokazane jest to na rysunku 10. W dolnej części programu otwiera się zakładka Robotics Examples, w której znajdują się różnorodne przykłady umożliwiające sterowanie robotami przy pomocy klawiatury. W dalszej części zaprezentowano wybrane przykłady.



Rys. 11: Okno do dodawania dostępnych w ISAAC SIM przykładowych symulacji.

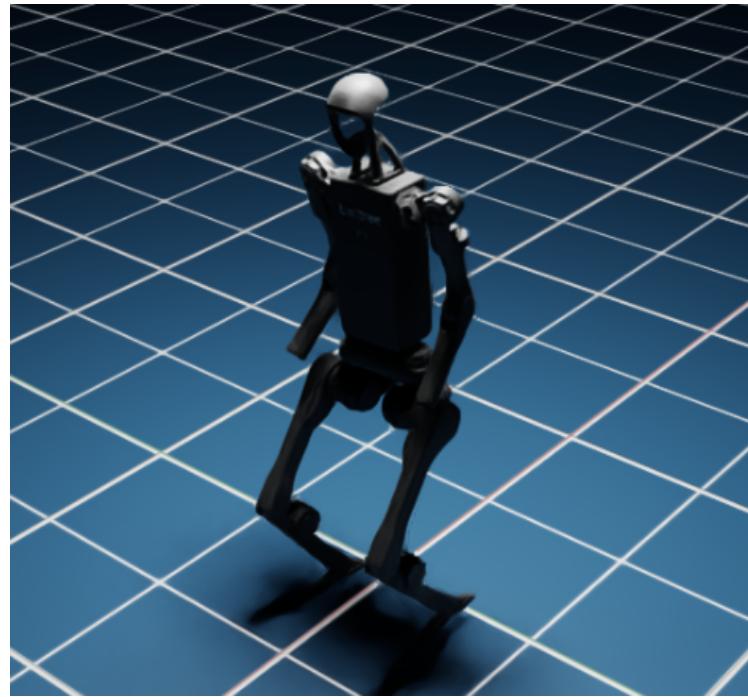
Na rysunku 11 przedstawiony jest widok na cyfrowego bliźniaka firmy Boston Dynamics. W przykładowej symulacji jest możliwe sterowanie nim przy pomocy strzałek i przycisków klawiatury. Podobnie na rysunku 12

pokazany jest humanoidalny robot firmy Unitree nim, również możliwe jest sterowanie w dostępnej przykładowej symulacji.

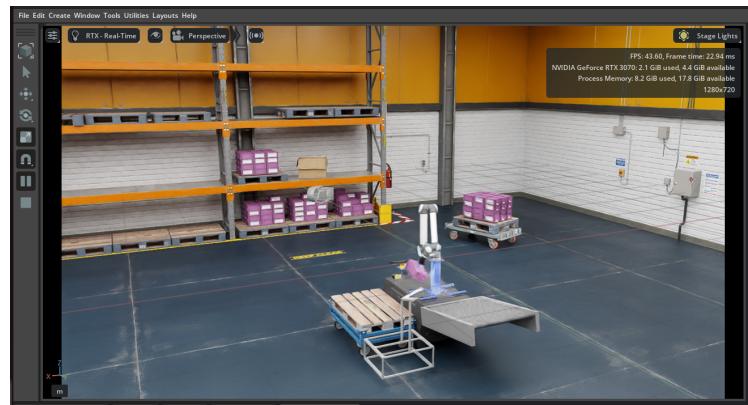


Rys. 12: Enter Caption

W środowisku ISAAC SIM dostępny jest też przykład z manipulatorem układającym kolejne dostarczane przez ruchomy taśmociąg palety. Okno z widoku tej przykładowej symulacji jest pokazane na rysunku 13.



Rys. 13: Przykład z ISAAC SIM z robotem humanoidalnym.



Rys. 14: Przykład z ISAAC SIM z manipulatorem i paletami.