

Raport: Praca nad Symulatorem TIAGo w ROS2, O3DE i integracja z TASKeR

OPRACOWANIE RAPORTU DOTYCZĄCEGO IMPLEMENTACJI SYMULATORA DZIAŁAJĄCEGO W ros 2 DO
TESTOWANIA SYSTEMU ZARZĄDZANIA ZADANIAMI ROBOTA AYSYSTUJĄCEGO

Dzieło realizowane w ramach grantu badawczego przyznanego przez Centrum Badawcze POB sztuczna
inteligencja i bobotyka Politechniki Warszawskiej, Projekt "Inicjatywa Doskonałości - Uczelnia Badawcza".

Autor: Adam Krawczyk



Wstęp

Celem pracy było stworzenie symulatora robota TIAGo w środowisku O3DE wykorzystując ROS2. Symulator powinien być w stanie symulować działanie robota w różnych scenariuszach, takich jak poruszanie się po środowisku. W ramach pracy nad symulatorem należało zaimplementować funkcjonalności robota, takie jak sterowanie ruchem, oddziaływanie ze środowiskiem (model fizyczny). Ponadto należało zintegrować symulator z ROS2, tak aby możliwe było wykorzystanie symulatora do testowania algorytmów sterowania robota. Interfejs symulatora powinien pokrywać się z interfejsem rzeczywistego robota i symulacji w Gazebo, aby możliwe było wykorzystanie symulatora do testowania rozwiązań (digital twin).

Przegląd Projektu

Zdecydowano o wyborze środowiska symulacyjnego bazującego na O3DE ponieważ w przeciwieństwie do Gazebo, O3DE jest silnikiem gier, który oferuje wiele funkcjonalności, które mogą być wykorzystane w symulacji robotów. Głównym atrybutem jest symulacja fizyki, która jest niezbędna do symulacji robota. O3DE wykorzystuje silnik fizyczny PhysX, który jest uważany za jeden z najlepszych silników fizycznych na rynku. Zapewnia on symulację fizyki w czasie rzeczywistym, co jest niezbędne do symulacji robota. Ponadto O3DE oferuje wiele funkcjonalności, które mogą być wykorzystane do symulacji robota, takich jak:

- Wysokiej jakości grafika
- Wsparcie dla akceleracji sprzętowej (Nvidia CUDA)
- Edytor scen który pozwala na łatwą konfigurację środowiska symulacyjnego (robota i otoczenia)
- Otwarta architektura, która pozwala na łatwe rozszerzanie funkcjonalności silnika (open source)

Specyfikacje Techniczne

Robot TIAGo Robot TIAGo, stworzony przez PAL Robotics, jest zaawansowanym robotem mobilnym przeznaczonym do zastosowań w badaniach i edukacji. Jego nazwa, TIAGo, jest akronimem od "Take It And Go", co podkreśla jego wszechstronność i mobilność. Oto kilka kluczowych cech robota TIAGo:

- modułowa konstrukcja
- napęd różnicowy (differential drive)
- obrotowa kamera RGB-D
- skaner laserowy (LiDAR)
- IMU (Inertial Measurement Unit)
- pomiar odometrii (encoder)

Wersje ROS2 i O3DE użyte w projekcie

Nr.	Nazwa	Wersja
1	System	Ubuntu 22.04
2	ROS2	Humble
3	O3DE	2310.0
4	O3DE-Extras	2310.0

Środowisko Symulacyjne

Wymagania Sprzętowe

Minimalne wymagania sprzętowe do uruchomienia symulatora TIAGo w O3DE:

Nr.	Nazwa	Specyfikacja
1	System	Ubuntu 22.04
2	CPU	Quad-core (4 rdzeni) 64-bit x86 z wsparciem dla SSE 4.1 SIMD
3	GPU	Nvidia GeForce GTX 1060 lub lepsza (2 GB pamięci RAM)
4	RAM	16 GB
5	Dysk	50 GB wolnego miejsca

Rekomendowane wymagania sprzętowe do uruchomienia symulatora TIAGo w O3DE:

Nr.	Nazwa	Specyfikacja
1	System	Ubuntu 22.04
2	CPU	Hexa-core (6 rdzeni) 64-bit x86 z wsparciem dla SSE 4.1 SIMD
3	GPU	Nvidia GeForce GTX serii 16 lub lepsza (6 GB pamięci RAM)
4	RAM	32 GB
5	Dysk	SSD 50 GB wolnego miejsca

Wymagania Software'owe

Wymagane oprogramowanie do uruchomienia symulatora TIAGo w O3DE:

- cmake
- Clang i biblioteka GNU C++ (libstdc++-12-dev clang)
- Vulkan support (libglu1-mesa-dev libxcb-xinerama0 libxcb-xinput0 libxcb-xinput-dev libxcb-xf86dev libxcb-xkb-dev libxkbcommon-dev libxkbcommon-x11-dev libfontconfig1-dev libpcre2-16-0 zlib1g-dev mesa-common-dev libunwind-dev libzstd-dev)
- Ninja build system (ninja-build)
- Git Large File Storage (git-lfs)
- Python 3.8 lub nowszy (python3.8-dev)
- ROS2 Humble (ros-humble-desktop)

Konfiguracja ROS2

Docelowa konfiguracja ROS2 jest zależna od ustawień systemowych nie symulatora. Środowisko ROS2 powinno być skonfigurowane zgodnie z instrukcjami dostępnymi na stronie [ROS2](#). Przetestowano symulator z domyślną konfiguracją DDS (FastRTPS).

Konfiguracja O3DE

Konfiguracja środowiska jest domyślna. Wymagane jest jedynie zainstalowanie zależności wymienionych w sekcji [Wymagania Software'owe](#).

Rozwój Symulatora

Do stworzenia symulatora TIAGo w O3DE wykonano następujące kroki:

1. Instalacja O3DE i O3DE-Extras
2. Instalacja ROS2 Humble
3. Stworzenie projektu w O3DE na podstawie szablonu ([ROS2ProjectTemplate](#))
4. Stworzenie modelu robota na podstawie modelu z Gazebo (format URDF/XACRO)
5. Doskonalenie modelu robota (dodanie modelu fizycznego, kolizji, sterowanie ruchem)
6. Dodanie sensorów robota (kamera RGB-D, skaner laserowy, IMU, odometria)
7. Skonfigurowanie symulacji (odpowiednie nazwy topiców)

Integracja TIAGo z ROS2

W modelu symulacji robota TIAGo wykorzystano komponenty z O3DE-Extras, które są odpowiedzialne za integrację symulatora z ROS2. Wykorzystano następujące sensory:

- ROS2 Camera Sensor (kamera RGB-D)
- ROS2 IMU Sensor (IMU)
- ROS2 Odometry Sensor (odometria)
- ROS2 Lidar Sensor (skaner laserowy)

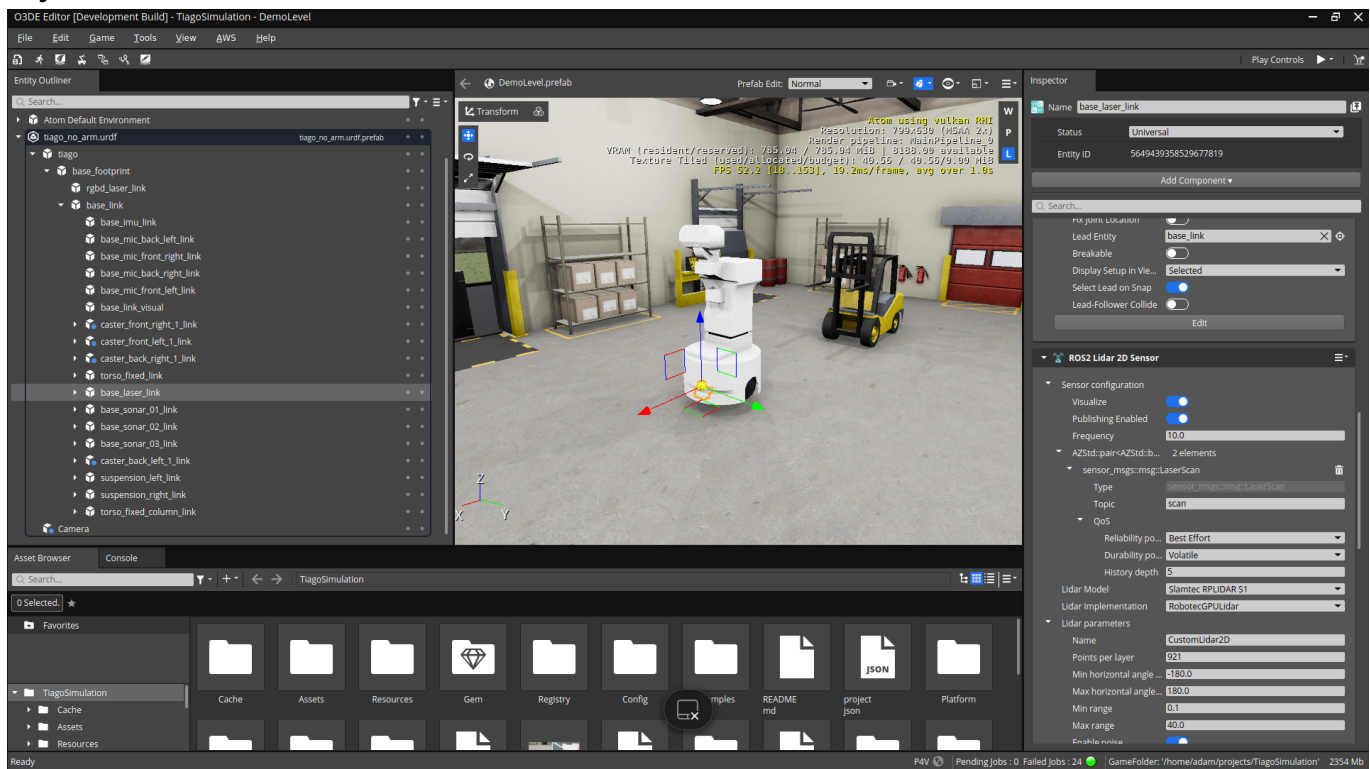
Implementacja Funkcjonalności

Dla założonego celu projektu, stworzono następujące funkcjonalności:

- sterowanie ruchem robota (prędkość liniowa i kątowa)
- model fizyczny robota (PhysX)
- model kolizji robota (PhysX)
- osensorowanie robota (kamera RGB-D, skaner laserowy, IMU, odometria)
- opis uruchomienia symulatora
- przykładowe scenariusze symulacji (nawigacja z [nav2](#))

Zrzuty Ekranu z Symulatora

Edytor O3DE:





Symulacja:



Atom using vulkan RHI
Resolution: 799x539 (MSAA 2x)
Render pipeline: MainPipeline 0
VRAM (resident/reserved): 785.17 / 785.17 MiB | 8188.00 available
Texture Tiled (used/allocated/budget): 40.56 / 40.56/0.00 MiB
FPS 59.6 [33..138], 16.8ms/frame, avg over 1.0s




ROS2 Lidar 2D Sensor


▼

Sensor configuration

Visualize

☒

Publishing Enabled

☒

Frequency

10.0


▼

AZStd::pair<AZStd::b...

2 elements

▼

sensor_msgs::msg::LaserScan



Type

sensor_msgs::msg::LaserScan

Topic

scan

▼

QoS

Reliability po...

Best Effort

▼

Durability po...

Volatile

▼

History depth

5

Lidar Model

Slamtec RPLIDAR S1

▼

Lidar Implementation

RobotecGPULidar

▼

▼

Lidar parameters

Name

CustomLidar2D

Points per layer

921

Min horizontal angle ...

-180.0

Max horizontal angle...

180.0

Min range

0.1

Max range

40.0

Enable noise

☒

▼

Noise parameters

Angular noise std ...

0.0

Distance noise std...



0.02

Distance noise std...

0.001

Excluded Entities

0 elements

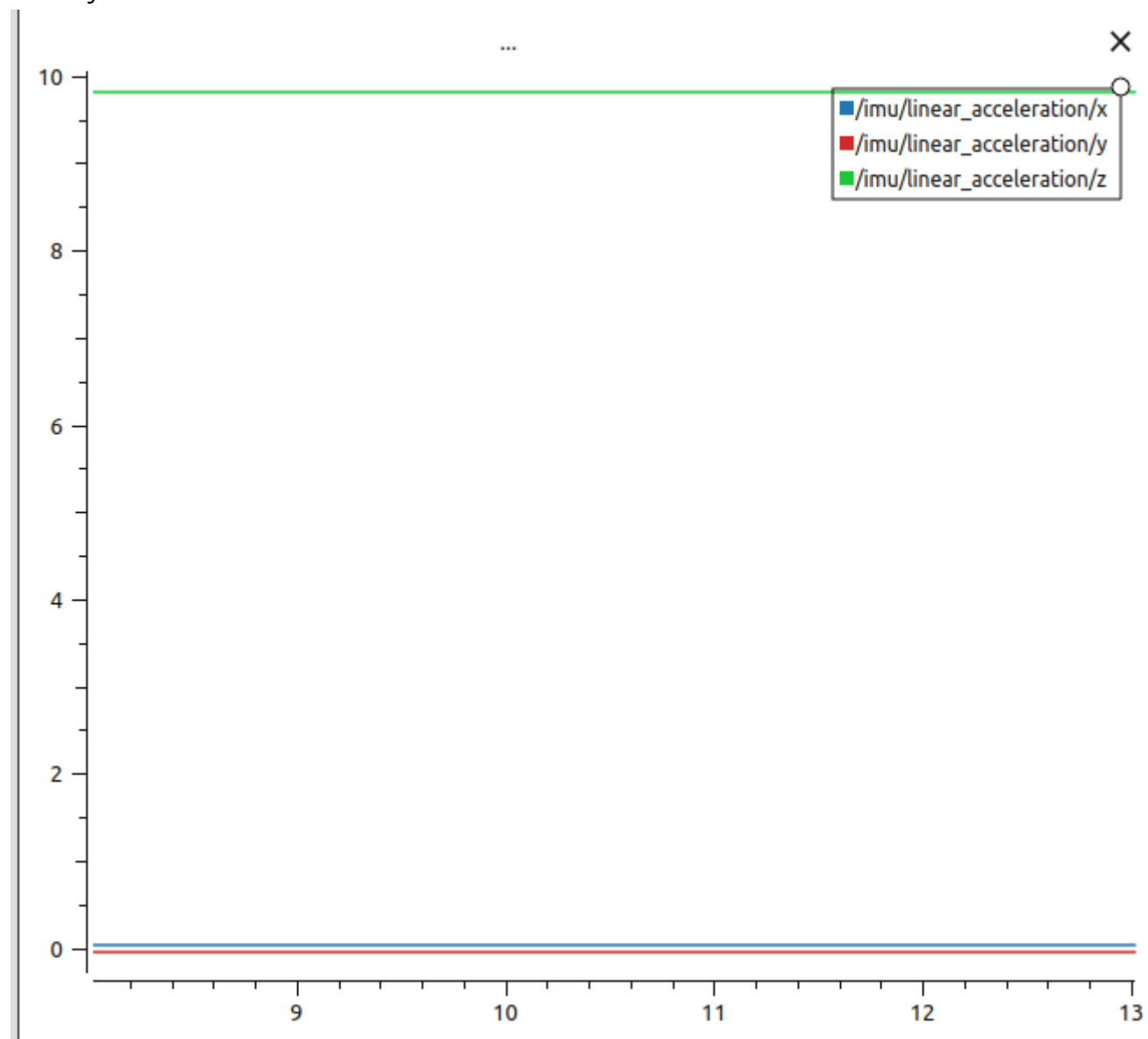
 

Points at Max

☐

Konfiguracja Komponentów:

Sensory:



/camera_image_color



0



10.00m



☐ camera_image_color_mouse_left

☐ Smooth scaling

0°



/camera_image_depth



0



10.00m



☐ camera_image_depth_mouse_left

☐ Smooth scaling

0°



Scenariusze Symulacji

Został przetestowany scenariusz w którym robot miał przeprowadzić SLAM (Simultaneous Localization and Mapping) w środowisku. W tym celu wykorzystano pakiet `slam_toolbox` oraz `nav2`. Robot był w stanie poruszać się w terenie na podstawie stworzonej mapy i zadawanych celów.

Repozytoria z Kodami

Linki do repozytoriów GitHub zawierających kod źródłowy projektu.

- [Kod źródłowy symulatora](#)
- [Kod źródłowy modelu robota](#)
- [Kod źródłowy komponentów O3DE-Extras](#)
- [Kod źródłowy komponentów O3DE](#)

Lista Komponentów Oprogramowania

Wykaz komponentów oprogramowania robota, które omówiliśmy na ostatnich spotkaniach.

| 1 | base_laser_link | Robotec GPU Lidar | `scan` "10hz Best Effort Volatile" | | 2 | base_sonar_x_link | Robotec GPU lidar | `pc` "5hz Best Effort Volatile" | | 3 | head_front_camera_rgb_frame | ROS2 Camera Sensor | `camera_image_color` "30hz Best Effort Volatile" | RGB and depth image | | 4 | base_imu_link | ROS2 IMU Sensor | `imu` "100hz Best Effort Volatile" | IMU data | | 5 | base_footprint | ROS2 Odometry Sensor | `odom` "10hz Best Effort Volatile" | Odometry data |

Nr.	Nazwa komponentu	Opis	Topic
1	ROS2 Camera Sensor	Kamera RGB-D	<code>camera_image_color camera_depth_image</code> "30hz Best Effort Volatile"
2	ROS2 IMU Sensor	IMU	<code>imu</code> "100hz Best Effort Volatile"
3	ROS2 Odometry Sensor	Odometria	<code>odom</code> "10hz Best Effort Volatile"
4	ROS2 Lidar Sensor	Skaner laserowy	<code>scan</code> "10hz Best Effort Volatile"
5	ROS2 Frame	Ramka odniesienia	-
6	PhysX Joint	Połączenie między elementami robota	-

Podsumowanie

Robot jest w stanie poruszać się w środowisku oraz wysyłać dane z sensorów na standardowe topics ROS 2.

Podziękowania

Podziękowania dla osób zaangażowanych w projekt, szczególnie dla P. Wojciecha Dudka.