Mateusz BORKOWSKI, Krystian ŁYGAS

MODEL ROBOTA SZEREGOWEGO TYPU SCARA W ŚRODOWISKU ROS

W artykule ukazano zastosowanie modułu Rviz do wizualizacji stanu manipulatora typu Scara. W tym celu opracowano URDF dla rzeczywistego modelu robota. Dokonano wizualizacji stanu robota przy pomocy modułu Rviz. Uwzględniono parametry bezwładnościowe manipulatora przez co uzyskano model dynamiczny możliwy do uruchomienia przy pomocy modułu Gazebo. Dzięki zastosowanej metodzie możliwe jest programowanie manipulatora w trybie offline czyli bez fizycznego dostępu do niego. Umożliwia to testowanie programów narzędziowych oraz algorytmów sterowania robota bez narażenia na uszkodzenie urządzenia.

WSTĘP

Większość rodzin manipulatorów przemysłowych wraz z możliwością programowania online z panelu operatorskiego „teach-pendant', posiada dedykowane oprogramowanie symulacyjne. Przykładowymi może być Roboguide – firmy Fanuc, K-Roset – firmy Kawasaki. Powstanie takiego oprogramowania wyniknęło z zapotrzebowania projektowania fabryk, zmian parametrów produkcji czy automatyzacji tradycyjnych procesów technologicznych, które było czasochłonne i wiązało się z dużym nakładem sił i środków. W celu poszukiwania oszczędności opracowywano metody pozwalające na szybsze przezbrajania oraz zmienianie konfiguracji pracy robotów, gdyż długie postoje generowały straty fabrykom. Poszukiwano więc narzędzi które nie tylko by rozwiązywały problem przeprogramowywania dużej liczby kontrolerów robotów, ale również takiego które umożliwiłoby wirtualne projektowanie całych fabryk.

Środowiska dedykowane do manipulatorów mają głownie pomóc projektantowi/programiście osiągnąć zamierzony cel bez dostępu do fizycznego manipulatora. Daje również wiele informacji, dzięki czemu może uchronić przed wieloma błędami poprzez walidację programu w środowisku symulacyjnym. Zapewnia więc ochronę urządzenia przed zniszczeniem na fazie uruchamiania linii produkcyjnej, gdyż umożliwia wykrycie błędów programowych bez poniesienia konsekwencji. Oprogramowanie to ma za zadanie wspomóc pracę programisty, poprzez przygotowanie 'layoutu' wykonywanego procesu. Umożliwia on między innymi zdefiniowanie punktów odniesienia, określenie lokalizacji newralgicznych punktów pracy. Ważnym aspektem jest również przeprowadzenie testu koncepcji, który pozwala dobrać prawidłowy manipulator pod względem czasu cyklu pracy.

Tworząc własny manipulator nie posiadamy zalet wcześniej wspomnianych. O ile manipulator pracuje w środowisku ROS (robot operating system) istnieje możliwość użycia węzła Rviz. Jest to węzeł (podprogram) systemu operacyjnego ROS, umożliwiający wizualizację stanu manipulatora. Aby móc go użyć należy posiadać plik URDF (universal robot description file) opisujący manipulator.

W artykule we wstępie teoretycznym przedstawiono strukturę systemu operacyjnego robotów (ROS), przybliżono moduł wizualizacji robota (RVIZ), opisano zunifikowany format opisu robota (URDF). Wszystkie powyższe narzędzia służą do przeprowadzenia symulacji oraz sterowania robotów. Przedstawiono również wykonany URDF oraz wynik symulacji kompatybilności manipulatora robotycznego typu Scara w module wizualizacyjnym środowiska ROS.

1. ROS

System operacyjny robotów (ROS) z ang. Robotic operating system jest narzędziem, strukturą służącą do pisania oprogramowania robotów. Jest zbiorem bibliotek, narzędzi i konwencji, które w swej idei mają za zadanie uprościć zadanie tworzenia samych robotów jak również ich otoczenia pracy. ROS może być wykorzystywany w celach naukowych oraz komercyjnych za darmo dzięki temu ,że został wydany na licencji BSD. Środowisko wykorzystuje platformę Linux, która z racji zaliczania do tzw. otwartego oprogramowania z ang. open source jest wykorzystywane platformą w robotyce i automatyce przemysłowej i nieprzemysłowej [1]. Programista ma szereg możliwości od pisania programów, po możliwość wykonywania testów robotów w module wizualizacyjnym RVIZ z ang. Robot Visualizer, a także w integracji ze środowiskiem Gazebo przeznaczonym do testów symulacyjnych dynamiki robotów. ROS także umożliwia współpracę robotów z systemami wizyjnymi z ang. vision system wykorzystując bibliotekę funkcji przetwarzania obrazu OpenCV. Dzięki czemu stosowany jest zarówno w przemyśle, jak i robotyce naukowej.

ROS zawiera wiele bibliotek pozwalających na sterowanie oraz symulację w skład których wchodzą:

- obsługujące urządzenia sprzętowe,
- sterowanie niskopoziomowe,
- implementację wykonywania typowych funkcji
- komunikację między-wątkową,
- zarządzanie pakietami.

Podstawowym językiem programowania w środowisku programistycznym ROS jest język programowania obiektowego C++ bądź Python, wybór języka programowania, zależy wyłącznie od preferencji programisty. Ros dostępny jest na następujących systemach operacyjnych: ubuntu, debian, eksperymentalnie na: osX (homebrew), gentoo, openEmbedded/Yocto [2]. Posiada wiele gotowych platform robotycznych, wśród których można zaliczyć roboty z następujących kategorii: szeregowych, kroczących, latających, mobilnych (kołowych), hybrydowych. Posiada również moduł Ros-Industrial służący do sterowania robotów przemysłowych. Składa się on z następujących pakietów [3]:

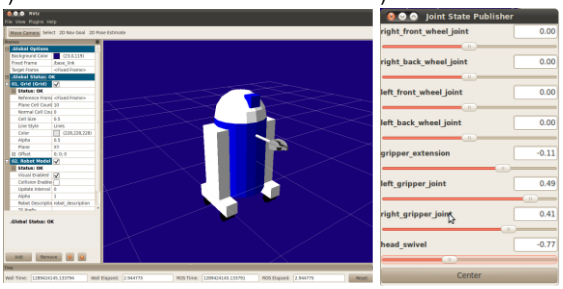
- ROS GUI- pakiet służący do tworzenia interfejsów graficznych, w skład którego wchodzi moduł Rviz, służący do wizualizacji stanu robota,
- Movelt Layer – pakiet służący do planowania trajektorii ruchu, kinematyki, aplikacji pick and place,

– ROS-I GUI – pakiet służący do symulacji standardowych przemysłowych teach-pendant'ów (paneli operatorskich robotów), będąca jeszcze w fazie rozwoju,
– ROS-I Interface Layer- jest to biblioteka klienta służąca do komunikacji kontrolera robota przemysłowego który wspiera komunikację po socket (Ethernet), służy do wymiany informacji, więc do zadania ruchu robota jak również odczytu jego stanów wewnętrznych,
– ROS-I Simple Message Layer – definiuje typ wiadomości służącej do połączenia, jak również protokół komunikacyjny kontrolera robota przemysłowego,
– ROS-I controler Layer –warstwa pełniąca funkcję kontrolera robota przemysłowego.
– ROS-Industrial wspiera następujące marki robotów: ABB, Adept, Fanuc, Kuka, Motoman, Robotiq, Universal Robots.

2. RVIZ

RVIZ jest modułem wizualizacyjnym środowiska ROS, umożliwia on za pomocą interfejsu graficznego wizualizację robota opisanego w pliku URDF [3]. Odzwierciedlona zostaje również struktura kinematyczna robota. Środowisko Rviz umożliwia pozycjonowanie członów robota, poprzez zadawanie kąta rotacji (rotation angle) w radianach bądź współrzędnych przesunięcia za pomocą panelu sterowania (joint_state_publisher) (Rys. 1.b)[4]. Za pomocą kolorów zdefiniowanych przez programistę mogą zostać zaznaczone kolejne człony manipulatora reprezentowane poprzez modele bryłowe CAD (Rys. 1.a). Proste bryły geometryczne mogą zostać również wygenerowane bezpośrednio w pliku URDF.

a) b)



Rys. 1. *Przykładowa wizualizacja graficzna pliku URDF modelu robota wraz z panelem sterowania kinematyką robota [4]*

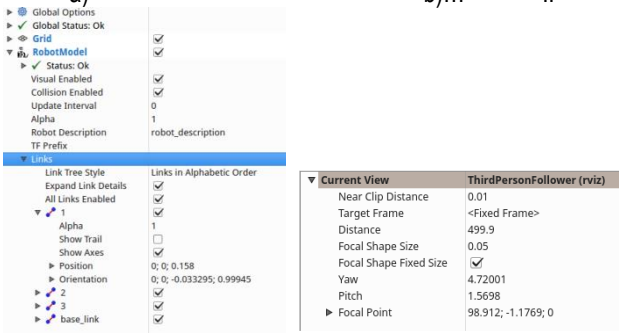
Suwakami są zadawane kolejne pozycje kątowe członów robota (Rys. 1.b)**.** Przycisk C*enter* umożliwia powrót do położenia początkowego zdefiniowanego w pliku URDF. W wyniku zadania odpowiednich pozycji kątowych i współrzędnych przesunięcia dla przegubów obrotowych z ang. revolute joint oraz pryzmatycznych z ang. prismatic joint robot zmienia orientację do zdefiniowanego przez programistę położenia. Środowisko umożliwia analizę kinematyczną ruchu robota oraz wyznaczanie przestrzeni roboczych, w tym stref kolizyjnych. Taka wiedza umożliwia projektowanie procesów zrobotyzowanych w sposób precyzyjny i uniwersalny dla wielu rodzin robotów przemysłowych co posiada zalety w przeciwieństwie do stosowania środowisk programistycznych dedykowanych dla jednej rodziny manipulatorów przemysłowych.

Rys. 2 *Schemat blokowy przedstawiający komunikację między-wątkową w środowisku programistycznym ROS.*

Komunikacja pomiędzy węzłami (node) odbywa się przy pomocy bloków tematu (topics), które przesyłają informację (message).

Taka architektura komunikacji umożliwia szybką wymianę danych w środowisku programistycznym ROS.

a) b)… ..

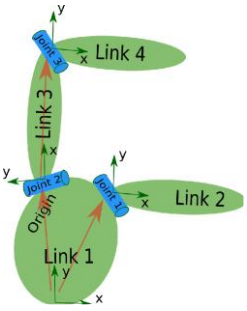


Rys. 3. *Panele graficzne modułu Rviz; a) okno parametryczne interfejsu graficznego modułu Rviz umożliwiające nadzór nad wizualizacją robota; b) okno parametryczne ustawień kamery interfejsu graficznego [4]*

Interfejs graficzny środowiska Rviz umożliwia dokładną wizualizację w pięciu trybach roboczych: (FPS, Orbit, ThirdPersonFollower, TopDownOrtho, XYorbit) oraz dostarcza informacji o obecnej pozycji kamery (Distance, Position, Pitch, Yaw)(Rys. 2.b). Umożliwia on również tymczasową edycję graficzną środowiska oraz modelu, bez konieczności edycji kodu pliku URDF (Rys 2.a). Panel ustawień parametrów wyświetlania umożliwia edycję siatki współrzędnych, wizualizacji modelu, kolizji, osi, a także wyświetla pozycję oraz orientację poszczególnych linków. Parametry te są zdefiniowane w pliku URDF, bądź wynikają z aktualnej pozycji robota w przestrzeni roboczej.

3. URDF

Zwiększona popularność użycia środowisk symulacyjnych robotów oraz bibliotek programowych spowodowała, że zaistniała potrzeba normalizacji opisu robotów. URDF czyli Unified Robot Description Format jak sama nazwa wskazuje jest to zunifikowany format opisu robota który służy do określania kinematyki i dynamiki, oraz wizualizacji reprezentacji (pozy) i modelu kolizji robota [3]. Plik ten jest wykorzystywany przez moduł wizualizacji Rviz oraz środowisko symulacji dynamiki robotów Gazebo. Na podstawie danych zawartych w pliku URDF możliwe jest obliczenie przestrzennej pozy robota oraz wykrycie ewentualnych błędów programistycznych środowiska kolidującego z robotem. Plik ten jest napisany wyjątkowo w języku programowania HTML. Aby programista mógł dobrze opisać robota za pomocą pliku URDF, potrzebny jest jego schemat ideowy na podstawie, którego robot zostaje zaimplementowany we właściwy sposób. Niektóre oprogramowanie CAD przy użyciu dedykowanych wtyczek umożliwia generowanie pliku URDF. Są to rozwiązania dla systemów firm Dassault Systemes (URDF Exporter) Solidworks oraz Autodesk (URDF Conventer) Inventor.



Rys 4. *Graficzna ideowa reprezentacja robota trójprzegubowego o trzech członach dynamicznych [5].*

Na podstawie schematu ideowego robota powstaje plik URDF. W pliku tym możemy wyróżnić następujące moduły opisujące poszczególne parametry: [1]:

a) Robota:
   – w tym miejscu definiowana jest nazwa robota,
b) Inercji:
   – w tym miejscu definiowane są parametry bezwładnościowe brył sztywnych z których składa się model, według modelu z (Rys.4) są to elementy opisane nazwą „Link”,
c) Wizualizacyjne:
   – w tym miejscy definiowane są parametry odnoszące się do wizualizacji danego elementu (Linku), możliwe jest w nim wykonanie prostej bryły geometrycznej poprzez definicję, lub wczytanie gotowego modelu bryłowego w formacie STL, możliwe jest również zdefiniowanie koloru jak również przezroczystości bryły,
d) Współrzędnych układu:
   – w tym miejscu definiowana jest współrzędna początku układu,
e) Geometryczne”
   – w tym miejscu definiowane są parametry geometryczne brył, powodujące odpowiednie translacje układów współrzędnych,
f) Materiałowe:
   – w tym miejscu definiowane są materiały oraz kolor wyświetlania członów bądź tekstura
g) Przegubów:
   – w tym miejscy definiowane są rodzaje przegubów, według modelu z (Rys.4) są to elementy opisane nazwą Joint”, definicja określa możliwe zachowanie się wzajemne sąsiednich brył sztywnych (Linków),
   – definiowane są również ograniczenia występowania danego przegubu.

Fragment przykładowego kodu URDF został zaprezentowany na (Rys. 5).

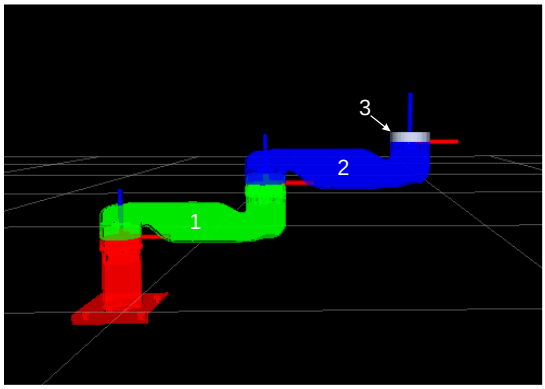
```
<?xml version="1.0" ?>
<robot name="materials">
  <link name="base_link">
    <visual>
      <geometry>
        <cylinder length="0.6" radius="0.2"/>
      </geometry>
      <material name="blue">
        <color rgba="0 0 .8 1"/>
      </material>
    </visual>
  </link>

  <link name="right_leg">
    <visual>
      <geometry>
        <box size="0.6 .2 .1"/>
      </geometry>
      <origin rpy="0 1.57075 0" xyz="0 0 -0.3"/>
      <material name="white">
        <color rgba="1 1 1 1"/>
      </material>
    </visual>
  </link>

  <joint name="base_to_right_leg" type="fixed">
    <parent link="base_link"/>
    <child link="right_leg"/>
    <origin xyz="0.22 0 .25"/>
  </joint>
```

Rys. 5. *Fragment przykładowego pliku URDF w języku XML robota o zdefiniowanej geometrii [4].*

Plik URDF można poddawać wielokrotnej edycji, aby osiągnąć określony przez programistę efekt wizualizacji bądź symulacji. Parametry bezwładnościowe pozwalają na odwzorowanie dynamiki robota, dzięki czemu można uznać że URDF jest opisem modelu robota.

4. WYNIK WIZUALIZACJI

W wyniku przeprowadzonej symulacji rzeczywistego manipulatora typu Scara w środowisku ROS przy wykorzystaniu modułu Rviz otrzymano graficzną wizualizację wraz z odwzorowaniem struktury kinematycznej rzeczywistego robota. Na (Rys. 6) przedstawiono schemat manipulatora, oraz zaznaczono człony ruchome robota. W (Tab.1) przedstawiono parametry bezwładnościowe robota, takie jak masa, położenia środka ciężkości, oraz masowe momenty bezwładności.

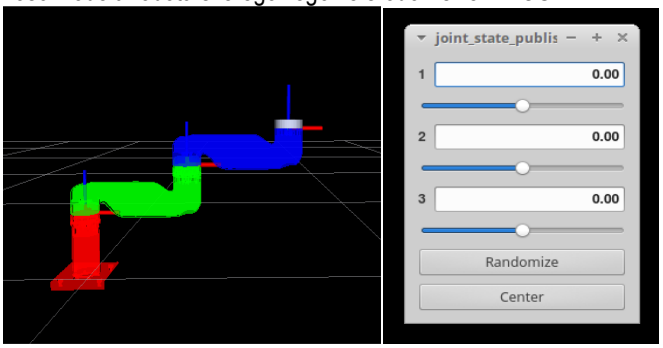


Rys. 6. *Schemat manipulatora typu Scara.*

Tab. 1. *Tabela parametrów robota typu Scara [6]*

Bryła	Parametr	Wartość
1)	Masa [kg]	1.9896
	Środek ciężkości (x,y,z) [m]	(0.24359,0,0.059265)
	I_{XX} [kg*m^2]	0.0041085
	I_{XY} [kg*m^2]	-8.195E-06
	I_{XZ} [kg*m^2]	0.0032956
	I_{YY} [kg*m^2]	0.025936
	I_{YZ} [kg*m^2]	5.9873E-06
	I_{ZZ} [kg*m^2]	0.023561
2)	Masa [kg]	0.79012
	Środek ciężkości (x,y,z) [m]	(0.15703,8.4232E-06, 0.039874)
	I_{XX} [kg*m^2]	0.0011636
	I_{XY} [kg*m^2]	-2.9234E-07
	I_{XZ} [kg*m^2]	0.0011521
	I_{YY} [kg*m^2]	0.013165
	I_{YZ} [kg*m^2]	3.4767E-09
	I_{ZZ} [kg*m^2]	0.013092
3)	Masa [kg]	0.11084
	Środek ciężkości (x,y,z) [m]	(0 0 0.01)
	I_{XX} [kg*m^2]	5.2573E-05
	I_{XY} [kg*m^2]	-1.1294E-21
	I_{XZ} [kg*m^2]	9.1496E-22
	I_{YY} [kg*m^2]	5.2573E-05
	I_{YZ} [kg*m^2]	6.7234E-22
	I_{ZZ} [kg*m^2]	9.7757E-05

Na (Rys. 7) przedstawiono wizualizację modelu zaimplementowanego w pliku URDF Podczas symulacji sprawdzono kompatybilność modelu robota szeregowego ze środowiskiem ROS.



Rys. 7.*Wynik testu kompatybilności robota ze środowiskiem ROS, zakończony wizualizacją w module Rviz.*

PODSUMOWANIE

Założeniem artykułu było opracowanie pliku URDF dla manipulatora typu Scara i jego zwizualizowanie w module wizualizacyjnym środowiska ROS, co miało potwierdzić prawidłowe wykonanie.

Sprawdzono w ten sposób kompatybilność modelu robota szeregowego z środowiskiem programistycznym ROS, którą uzyskano. Tak przygotowany model w formacie URDF umożliwia testowanie programów narzędziowych, oraz algorytmów sterowania robota bez narażenia na uszkodzenie urządzenia.

BIBLIOGRAFIA

1. M. Q. Smart Brian Gerkey, William D., *Programming Robots with ROS*. .
2. L. Joseph, *Learning Robotics using Python*. Birmingham: Packt Publishing - ebooks Account, 2015.
3. L. Joseph, *Mastering ROS for Robotics Programming*. Birmingham Mumbai: Packt Publishing - ebooks Account, 2015.
4. "urdf/Tutorials/Building a Movable Robot Model with URDF - ROS Wiki." [Online]. Available: http://wiki.ros.org/urdf/Tutorials/Buildng%20a%20Movable%20Robot%20Model%20with%20URDF. [Accessed: 28-Mar-2017].
5. "urdf/Tutorials/Create your own urdf file - ROS Wiki." [Online]. Available: http://wiki.ros.org/urdf/Tutorials/Create%20your%20own%20urdf%20file. [Accessed: 28-Mar-2017].
6. K. Łygas, "Metoda symulacyjna doboru parametrów napędu elektrycznego dla dedykowanej trajektorii ruchu," *Autobusy Tech. Eksploat. Syst. Transp.*, vol. R. 17, nr 12, 2016.

Model of manipulator Scara in ROS

The article shows the use of the Rviz module to visualize the state of the Scara manipulator. For this purpose, URDF has been developed for the actual robot model. The state of the robot has been visualized by the Rviz module. The inertia parameters of the manipulator were taken into account, resulting in a dynamic model that could be started with the Gazebo module. Thanks to the method used it is possible to programing offline of robot without physical access to it. This allows you to test tools and robot control algorithms without damaging the device.

Autorzy:
Mateusz Borkowski – Politechnika Lubelska, Wydział Mechaniczny, Katedra Automatyzacji, SKN „Feedback Control" mateusz.borkowski@pollub.edu.pl
mgr inż. Krystian Łygas – Politechnika Lubelska, Wydział Mechaniczny, Katedra Automatyzacji, k.lygas@pollub.pl