Roboty mobilne i kooperacyjne

Wprowadzenie do ROS

Materiały
http://staff.uz.zgora.pl/ipajak
http://staff.uz.zgora.pl/gpajak

ROS

Wprowadzenie

ROS

Robot Operating System (ROS)

- mimo nazwy "system operacyjny" ROS nie jest systemem operacyjnym
- otwartoźródłowa (open-source) platforma programistyczna
- zapewnia komunikację między procesami i wsparcie dla nisko-poziomowej kontroli urządzeń
- zestaw bibliotek programistycznych i narzędzi do tworzenia oprogramowania sterowania robotów

    od sterowników po najnowocześniejsze algorytmy oraz potężne narzędzia dla programistów pozwalających na sterowanie i symulację pracy robota

WikipediA

ROS 1 – Dystrybucje

Nr	Dystrybucja	Data wydania
1	ROS 1.0	
2	Box Turtle	02.03.2010
3	C Turtle	02.08.2010
4	Diamondback	02.03.2011
5	Electric Emys	30.08.2011
6	Fuerte Turtle	23.04.2012
7	Groovy Galapagos	31.12.2012
8	Hydro Medusa	04.08.2013
9	Indigo Igloo	22.06.2014
10	Jade Turtle	23.05.2015
11	Kinetic Kame	23.05.2016
12	Lunar Loggerhead	23.05.2017
13	Melodic Morenia	23.05.2018
14	Noetic Ninjemys	23.05.2020

- wersje ROS 1 poza pierwszą dystrybucją (ROS 1.0) mają nazwy rozpoczynające się od kolejnych liter alfabetu (podobnie jak Ubuntu i Android)
- każda wersja ma ilustrację w formie plakatu oraz ikonę żółwia
- symbol żółwia w ROS wywodzi się z edukacyjnego języka programowania Logo (opracowanego w latach 60)
- w MIT w 1969 stworzono robota żółwia opartego na Logo, który potrafił poruszać się po podłodze i rysować obrazy zgodnie z instrukcjami podawanymi przez komputer
- na podstawie tego robota w ROS opracowano program turtlesim
- ROS 1 nie będzie rozwijany dalej – wsparcie dla ostatniej wersji Noetic Ninjemys kończy się w 2025
- obecnie rozwijany jest ROS2

ROS 2 – Dystrybucje

Nr	Dystrybucja	Data wydania
1	Ardent Apalone	08.12.2017
2	Bouncy Bolson	02.06.2018
3	Crystal Clemmys	14.12.2018
4	Dashing Diademata	31.05.2019
5	Eloquent Elusor	22.11.2019
6	Foxy Fitzroy	05.06.2020
7	Galactic Geochelone	23.05.2021
8	Humble Hawksbill	23.05.2022
9	Iron Irwini	23.05.2023
10	Jazzy Jalisco	23.05.2024
11	Kilted Kaiju	23.05.2025

- wsparcie dla aplikacji czasu rzeczywistego
- zaawansowane funkcje bezpieczeństwa:
  - szyfrowana komunikacja między węzłami
  - kontrola dostępu do danych i usług
  - uwierzytelnianie użytkowników i urządzeń
- wieloplatformowy, ze wsparciem dla Linux, Windows, mac OS, systemów wbudowanych
- poprawione wsparcie dla narzędzi symulacyjnych (Gazebo, RViz2)
- obsługa nowych technologii i standardów
  - systemy oparte na chmurze
  - współpraca robotów w środowiskach IoT
  - nowoczesne standardy komunikacji i przetwarzania danych

ROS Industrial

rozszerzenie Robot Operating System (ROS), zaprojektowane specjalnie do zastosowań przemysłowych, daje wsparcie dla robotów przemysłowych (zawiera metapakiety dla wielu robotów przemysłowych)

3M, **ABB**, ADLINK, AF ManTech, Alias Robotics, ARC Specialties, Advanced Remanufacturing and Technology Centre (ARTC), ARM Institute, American Society for Testing and Materials (ASTM), AutomationSG, Bastian Solutions, Boeing, Bosch, BMW, Caterpillar, Delta Electronics, dormakaba, Fraunhofer IPA, FZI, Georgia Tech Research Institute, Harting, IAV GmbH, IHI, INC, Infineon, INESC TEC, Intel, Intrinsic, CNR - ITIA, JM Vistec Joeannum Research, Johnson & Johnson, Lely, Kabam Robotics, Keba, Kyocera, Kion Group, Lockheed Martin, Magna, Megazo Technologies, Mitsubishi Electric Asia Pte. Ltd, MTC, Nanyang Technological University, NIST - National Institute of Standards and Technology, NRC-CNRC, University of Texas at Austin - Nuclear Robotics Group, Numurus, NGEE ANN Polytechnic, The Ohio State University - College of Engineering, Omron, Open Source Robotics Foundation, Pepperl + Fuchs, Phoenix Contact, PickNik, PlusOne Robotics, PPM, PushCorp, Rensselaer Center for Automation Technologies and Systems, Republic Polytechnic (RP), Rethink Robotics, Saxion, SEC, Singapore University of Technology and Design (SUTD), Steel Founders' Society of America (SFSA), Schaeffler, SIAA, Siemens, Singapore Polytechnic (SP), Spirit Aerosystems, Southwest Research Institute (SwRI), Stogl Robotics, Synapticon, Tardec, Tecnologico de Monterrey, The Manufacturing Technology Center (MTC), Tormach, TrendMicro, Traclabs, T-Systems, TU Delft, Ubuntu by Canonical, **Universal Robots**, Volvo Group, Lincoln Automation, Yaskawa, Yokogawa

ROS Industrial – Universal Robots

Download the ROS/ROS2 Drivers

In order to use the ROS and ROS2 drivers, you must have ROS and/or ROS2 installed. For more guidance on this process, and information on the differences between the two frame works, visit the ROS Get Started page.

ROS Driver

In order to download and use the ROS driver, follow the instructions from this GitHub Repo.

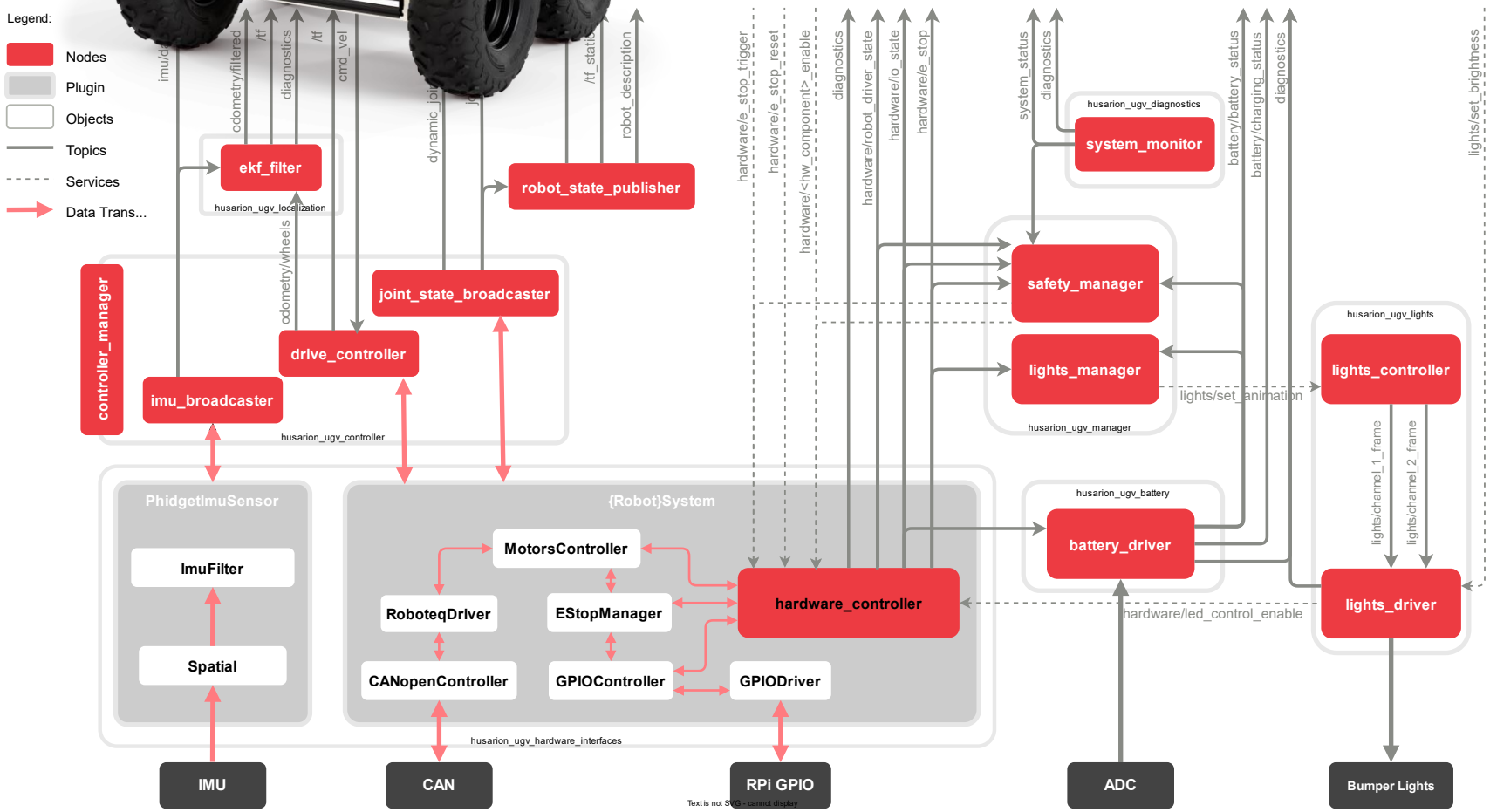
Develop with ROS Driver

ROS2 Driver

Here are the resources needed to get started with the ROS2 driver. This driver has built-in support with MoveIt!

Develop with ROS2 Driver

ROS Industrial – Husarion



ROS Industrial – intel REALSENSE

intel REALSENSE | PRODUCTS | DEVELOPERS | USE CASES | BLOG | SUPPORT | STORE

Documentation | ROS1

- Odroid installation
- FIRMWARE UPDATE
- Overview
- Firmware releases D400
- Intel® RealSense™ Documentation
- Firmware releases L500
- Firmware Update Tool (rs-fw-update)
- SDK 2.0 CODE SAMPLES WRAPPERS AND LANGUAGES
- Overview
- C/C++
- Python
- ROS - Robot Operating System
- ROS1
  - Starting camera node
  - PointCloud ROS Examples
  - Align Depth
  - Multiple Cameras
  - T265 Examples
  - D400+T265 ROS examples
  - Object Analytics
  - Open from File ROS example
  - Multiple cameras showing a semi-

ROS1

The ROS Wrapper for Intel® RealSense™ cameras allows you to use Intel® RealSense™ cameras with ROS1.

These are the **ROS (1)** supported Distributions:
📌 **Note:** The latest ROS (1) release is version 2.3.2.

- Noetic Ninjemys (Ubuntu 20.04 Focal)
- Melodic Morenia (Ubuntu 18.04 Bionic)
- Kinetic Kame (Ubuntu 16.04 Xenial)

ROS Documentation and Installation instructions can be found at: https://docs.ros.org

📌 **Note:** Check the ROS Release documentation at ROS official distributions
📌 **Note:** Check https://www.intelrealsense.com/message-to-customers/ for supported cameras.

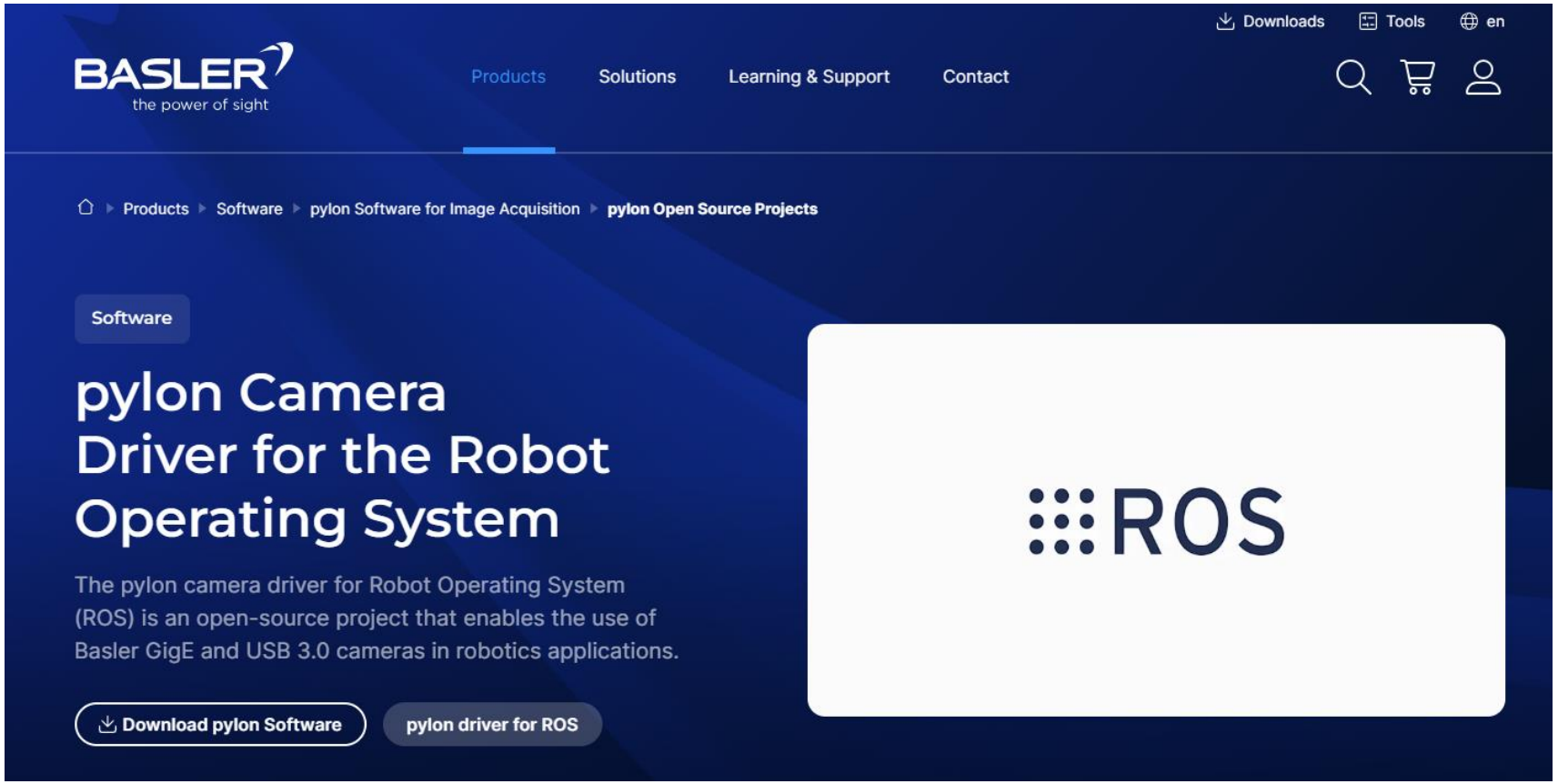
Getting Started

- ROS1 installation Instructions and distributions - link

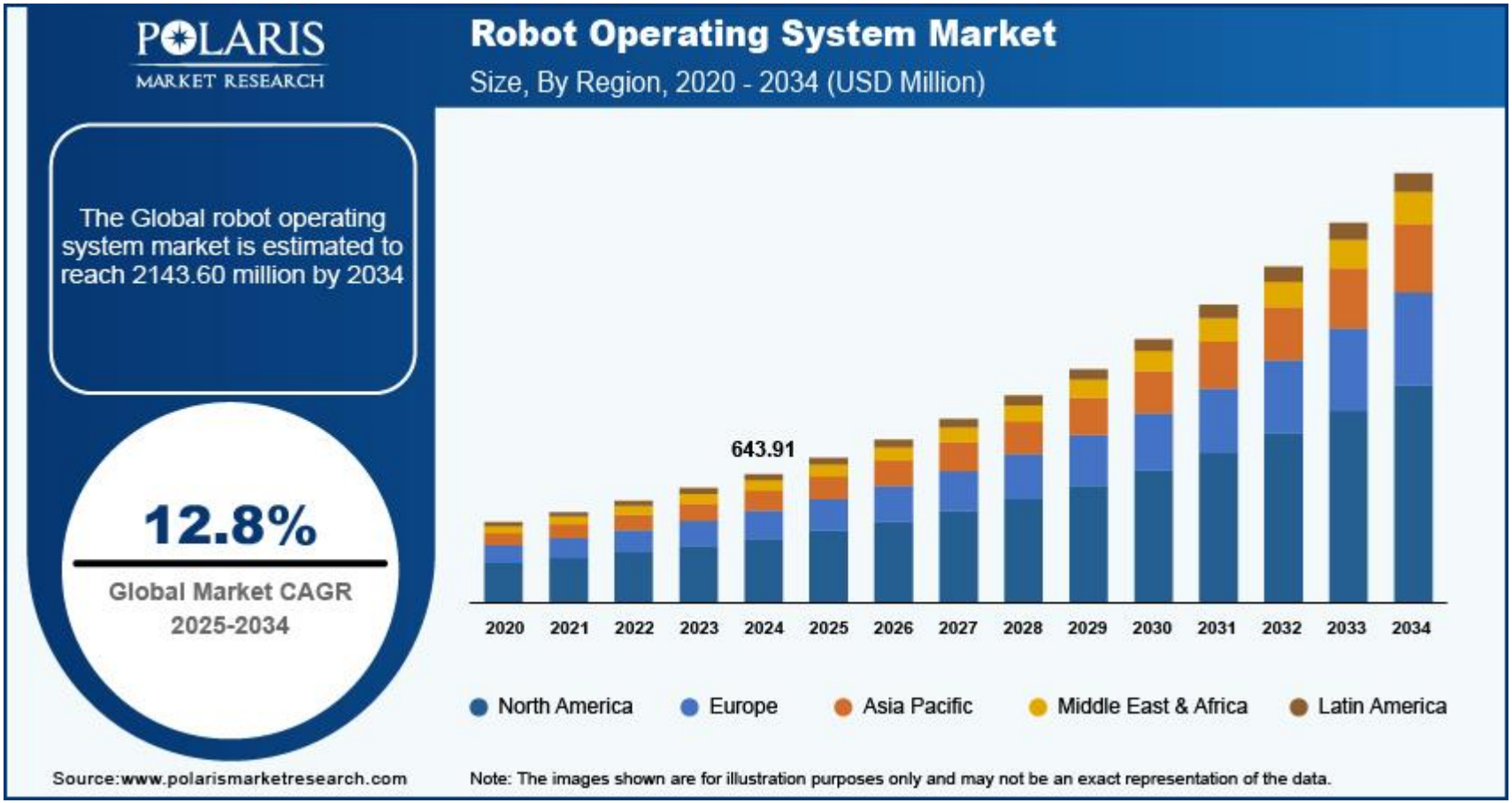
ROS Camera nodes examples

Link to Example	Experience level	Supported Cameras	Link to GitHub
Starting camera node	★	D400, L500, T265	rs_camera.launch
PointCloud visualization	★	D400	demo_pointcloud.launch rs_rgbd.launch rs_d435_camera_with_model.launch
Align Depth	★	D400	rs_aligned_depth.launch

ROS Industrial – BASLER



ROS


POLARIS
MARKET RESEARCH
Robot Operating System Market
Size, By Region, 2020 - 2034 (USD Million)
The Global robot operating system market is estimated to reach 2143.60 million by 2034
12.8%
Global Market CAGR 2025-2034
643.91
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
North America
Europe
Asia Pacific
Middle East & Africa
Latin America
Source:www.polarismarketresearch.com
Note: The images shown are for illustration purposes only and may not be an exact representation of the data.

ROS

Podstawy

Pakiety

Pakiet (ang. package)

- oprogramowanie w ROS jest zorganizowane w pakiety, pakiet może zawierać:
  - węzły ROS (kody źródłowe, skrypty, dokumentację, inne zasoby)
  - biblioteki niezależne od ROS
  - zestaw danych
  - pliki konfiguracyjne
- celem pakietów jest dostarczenie użytecznej funkcjonalności w sposób łatwy do wykorzystania
- pakiety mają zwykle strukturę podobną strukturę:

katalog	opis
msg	definicje typów komunikatów
src	pliki źródłowe
srv	definicje usług
scripts	skrypty (np. w pythonie)
launch	pliki .launch (służą do uruchamiania węzłów ROS i konfiguracji systemu)

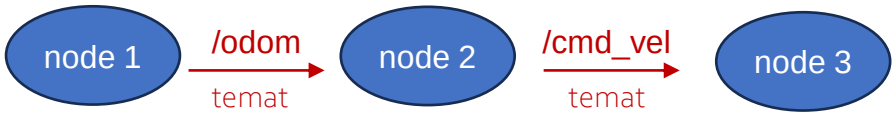
Pakiet turtlesim

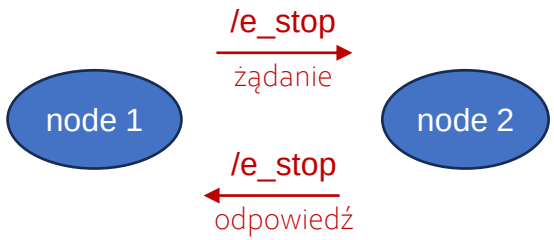
```
turtlesim/
├── CMakeLists.txt    # plik odpowiedzialny za konfigurację pakietu
├── package.xml       # plik konfiguracyjny pakietu ROS z tzw. metadanymi
├── launch/           # katalog zawierający pliki launch
├── msg/              # katalog zawierający definicje komunikatów
│   └── Pose.msg      # komunikat zawierający dane o pozycji żółwia
├── src/              # katalog źródłowy z plikami implementacji węzła
│   └── turtlesim.cpp # główny plik implementacji węzła turtlesim
├── srv/              # katalog zawierający definicje usług
│   ├── Kill.srv      # definicja usługi usunięcia żółwia
│   ├── Spawn.srv     # definicja usługi utworzenia żółwia
│   └── TeleportAbsolute.srv # definicja usługi teleportacji żółwia
└── scripts/          # katalog zawierający skrypty, np. w pythonie
    └── turtle_teleop_key.py # skrypt do sterowania żółwiem z klawiatury
```

ROS

Podstawy

- opiera się na idei tworzenia współbieżnie działających **węzłów** (ang. nodes) reprezentujących pojedyncze procesy i będących w istocie programami o dużym stopniu autonomii*
- węzły komunikują się między sobą poprzez ciągłe asynchroniczne rozgłaszanie i nasłuchiwanie **komunikatów** o konkretnym typie i **temacie** (ang. topic)
- węzły mogą komunikować się za pomocą **serwisów** (ang. service) synchronicznie w modelu żądanie-odpowiedź: jeden węzeł wysyła żądanie, drugi odpowiada na żądanie, komunikacja sprowadza się do wywołania funkcji i oczekiwania na odpowiedź





*WikipediA

Komunikat /cmd_vel

Znaczenie

- **/cmd_vel** jest wykorzystywany do publikowania poleceń sterujących prędkościami robota
- typ **/cmd_vel** to **geometry_msgs/Twist**, pozwala opisać prędkości liniowe oraz kątowe robota w trzech osiach

linear
  x
  y
  z
angular
  x
  y
  z

Komunikat /odom

Znaczenie

- **/odom** jest wykorzystywany do publikowania informacji o aktualnej pozie i prędkości robota (dane są szacowane na podstawie odczytów z czujników, takich jak enkodery w kołach, czujniki optyczne LIDAR, kamer)
- typ **/odom** to **nav_msgs/Odometry**, składa się z 3 głównych elementów:

  header (typ: **std_msgs/Header**) – nagłówek

  pose (typ: **geometry_msgs/PoseWithCovariance**) – pozycja i orientacja robota

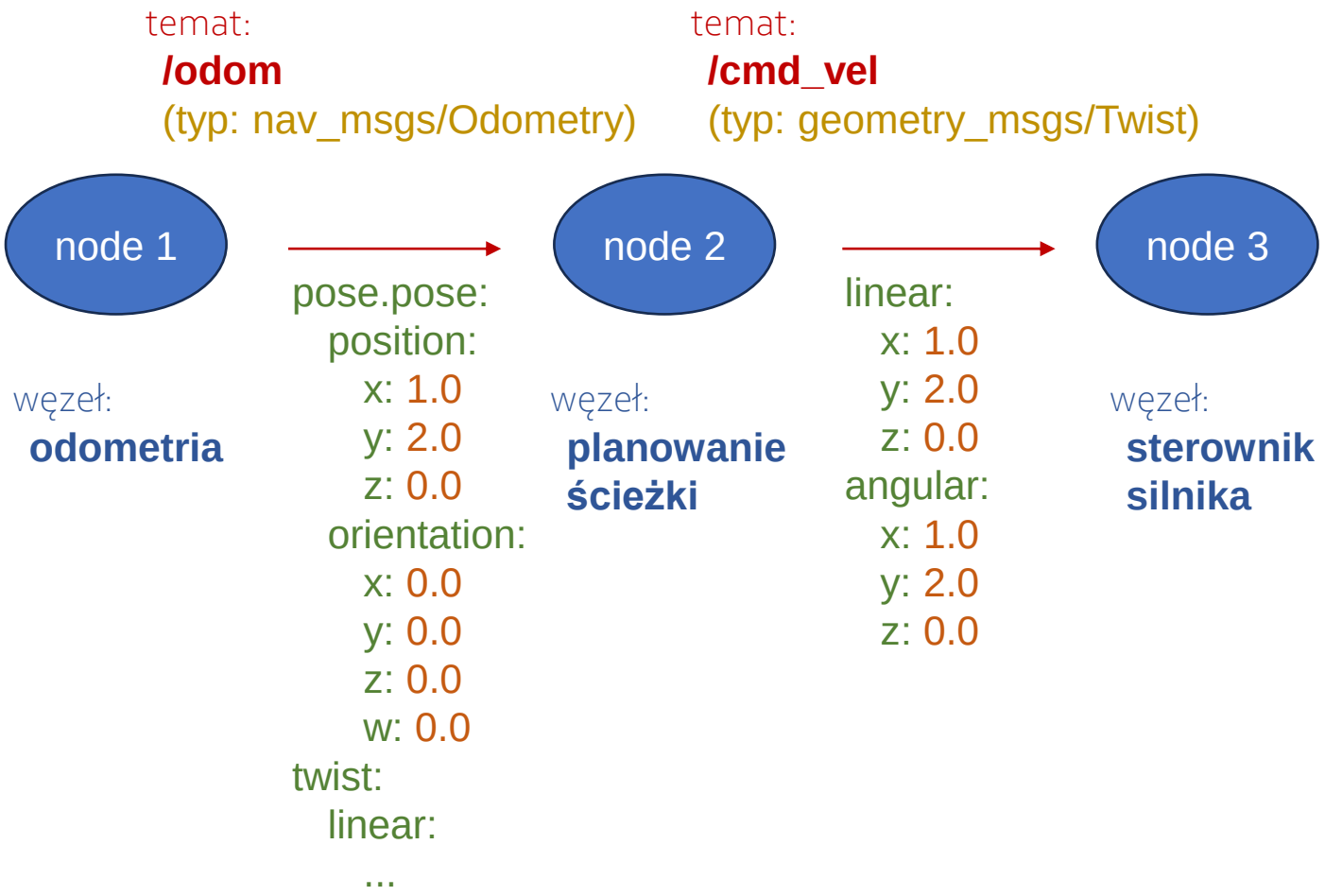
  twist (typ: **geometry_msgs/TwistWithCovariance**) – prędkość liniowa i kątowa

```
header
  seq
  stamp
  frame_id
```

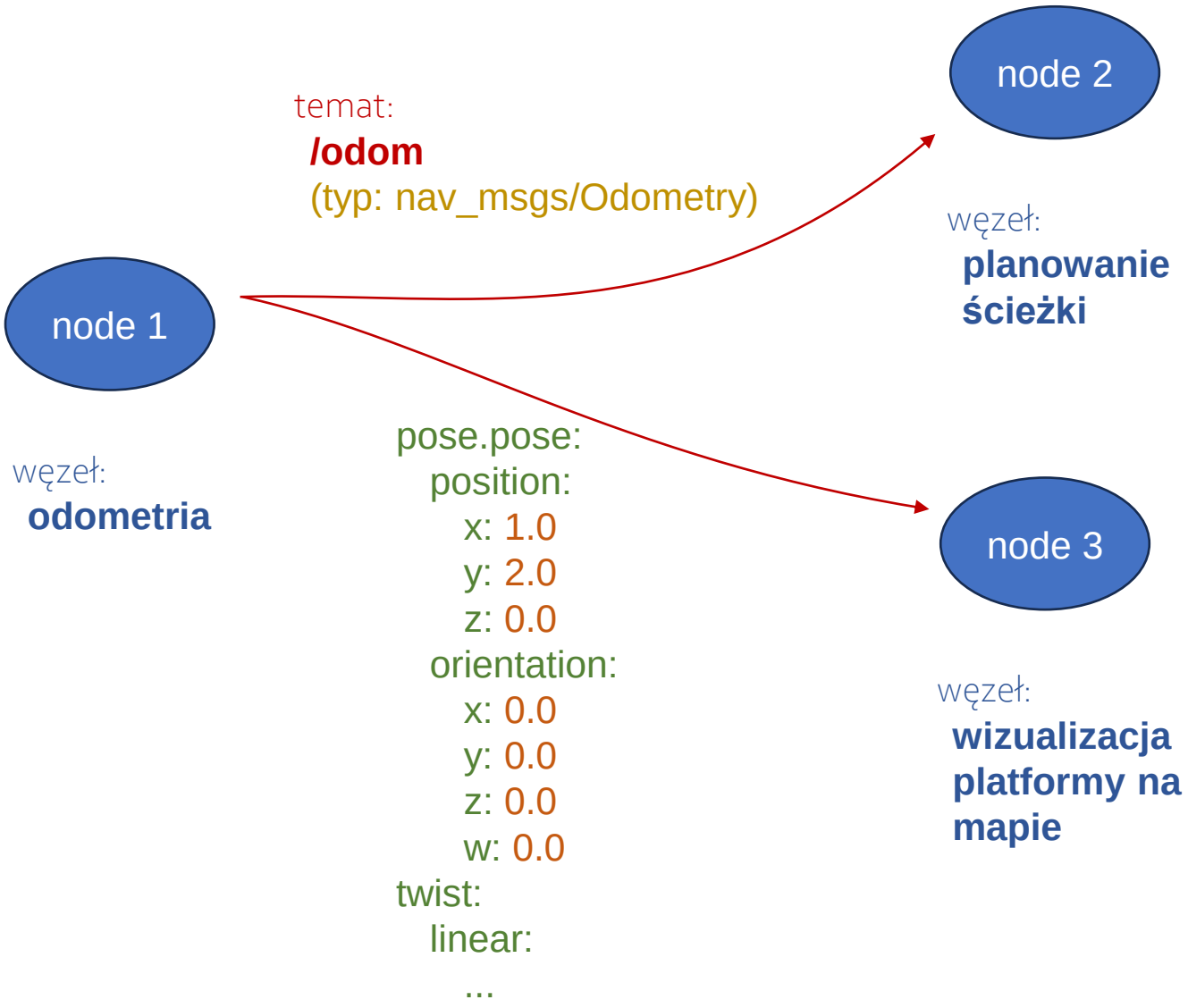
```
pose
  pose
    position
      x
      y
      z
    orientation
      x
      y
      z
      w
  covariance
```

```
twist
  twist
    linear
      x
      y
      z
    angular
      x
      y
      z
      w
  covariance
```

ROS – komunikacja asynchroniczna



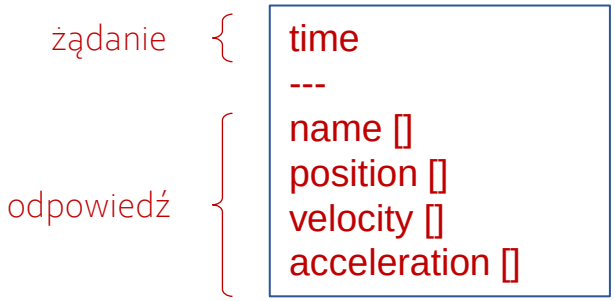
ROS – komunikacja asynchroniczna



Serwis /query_trajectory_state

Znaczenie

- **/query_trajectory_state** jest wykorzystywany w celu pobrania informacji o aktualnym lub planowanym stanie trajektorii
- typ **/query_trajectory_state** to **control_msgs/QueryTrajectoryState**, zawiera informacje o żądaniu (czas) dla której kontroler robota ma odpowiedzieć informacjami o stanie trajektorii (położenie, prędkość i przyspieszenie dla każdego połączenia robota)

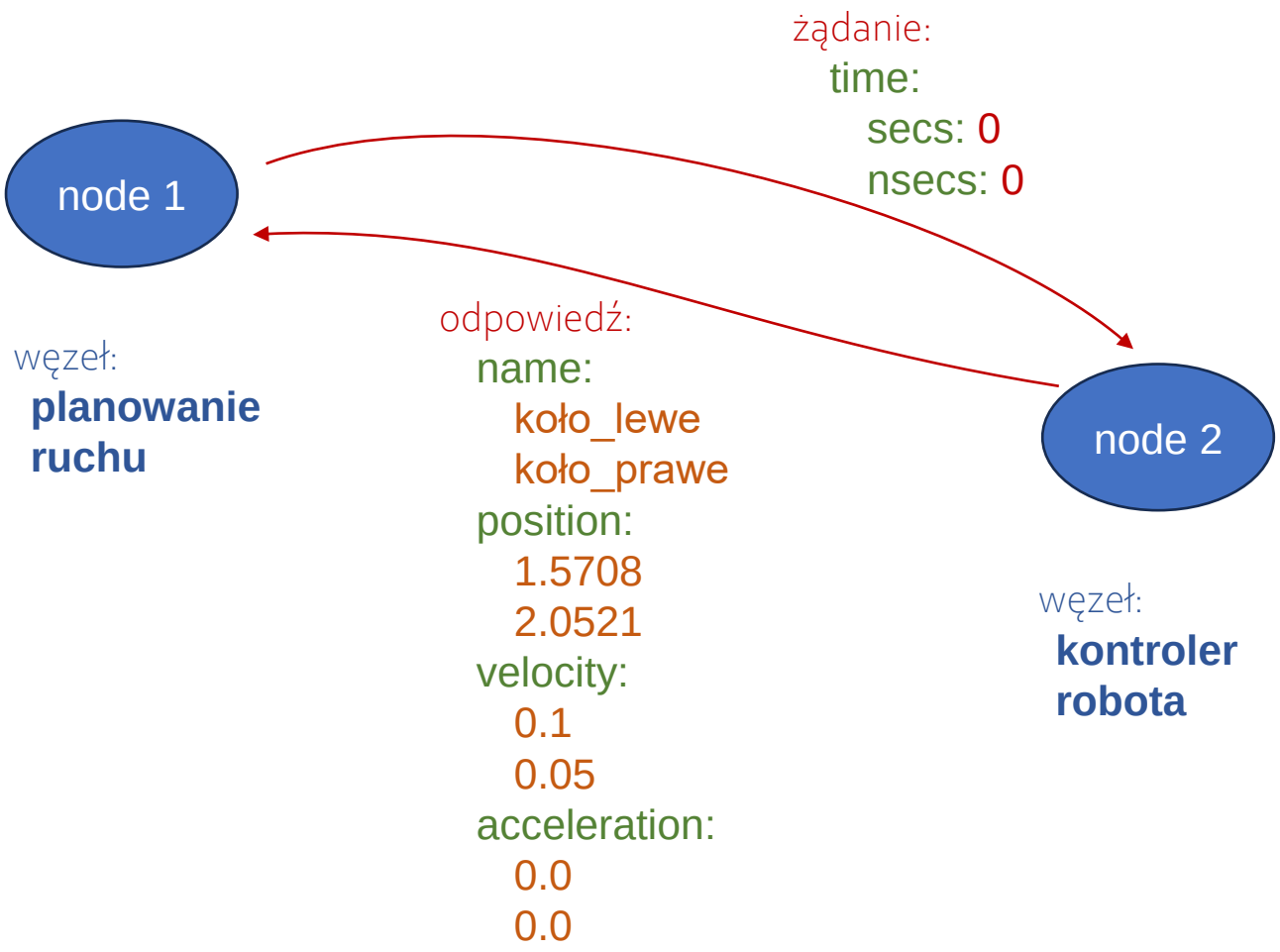


ROS – komunikacja asynchroniczna

serwis:

/query_trajectory_state

(typ: control_msgs/QueryTrajectoryState)



ROS Master

Własności i zadania

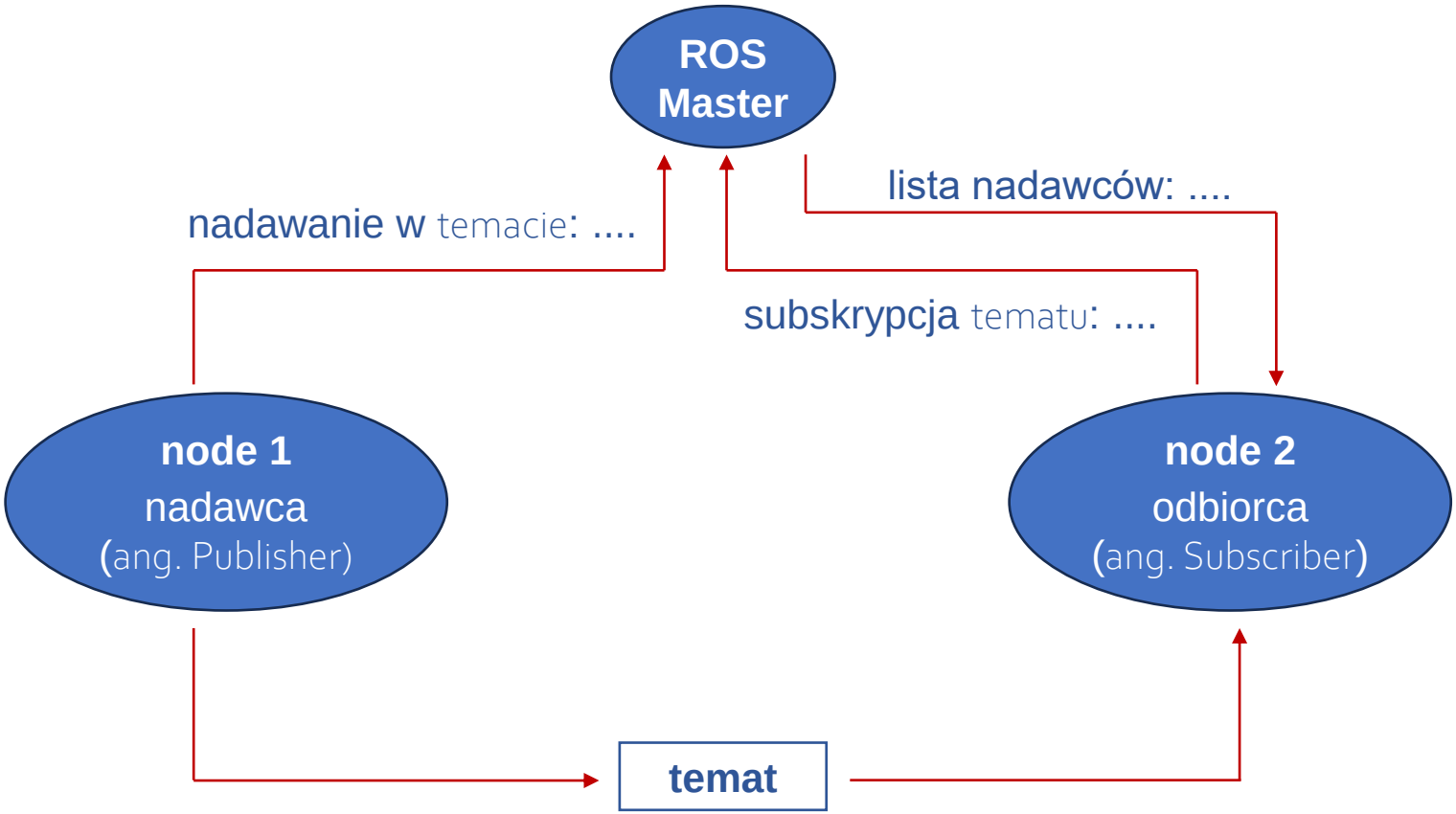
- centralny elementem systemu jest ROS
- zarządza komunikacją między węzłami
- prowadzi listy nadawców, odbiorców i usług
- działa jako serwer rejestrujący

  każdy węzeł, który chce uczestniczyć w systemie musi zarejestrować się w ROS Masterze, podając swoją nazwę i adres

  zarządzanie połączeniami
- pośredniczy w nawiązywaniu połączeń

  gdy węzeł chce subskrybować temat, pyta ROS Mastera o dostępnych nadawców danego tematu, ROS Master odpowiada wskazując węzły odpowiedzialne za dany temat (sama komunikacja między węzłem subskrybującym a węzłem publikującym odbywa się bezpośrednio, ROS Master nie jest zaangażowany w przesyłanie wiadomości. umożliwia tylko węzłom odnalezienie się i nawiązanie komunikacji

ROS Master


ROS Master
lista nadawców: ....
nadawanie w temacie: ....
subskrypcja tematu: ....
node 1
nadawca
(ang. Publisher)
node 2
odbiorca
(ang. Subscriber)
temat

roscore

roscore

polecenie, które inicjuje i uruchamia kluczowe komponenty systemu ROS:

- ROS Master
- rosout
- Parameter Server

```
ros@ubuntu:~$ roscore
... logging to /home/ros/.ros/log/.../roslaunch-ubuntu-2055.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu:42215/
ros_comm version 1.14.13

SUMMARY
========
PARAMETERS
 * /rosdistro: melodic
 * /rosversion: 1.14.13
NODES
auto-starting new master
process[master]: started with pid [2069]
ROS_MASTER_URI=http://ubuntu:11311/

setting /run_id to 1a8d5764-cf81-11ef-a291-000c291f04d8
process[rosout-1]: started with pid [2080]
started core service [/rosout]
```

rosout

usługa logowania, zbiera wszystkie komunikaty logujące z węzłów (umożliwia śledzić statusy systemu i błędów)

Parameter Server

przechowuje parametry konfiguracyjne, które mogą być używane przez węzły

rosrun

rosrun

- narzędzie w systemie ROS, które służy do uruchamiania wielu węzłów
- pozwala na uruchomienie pliku wykonywalnego z dowolnego pakietu z dowolnego miejsca, bez konieczności podawania najpierw pełnej ścieżki dostępu lub polecenia cd/roscd

Składnia

```
rosrun <pakiet> <program wykonywalny>
```

Przykłady

```
rosrun turtlesim turtlesim_node
rosrun turtlesim turtle_teleop_key
```

roslaunch

roslaunch

- narzędzie w systemie ROS, które służy do uruchamiania wielu węzłów (nodes) w sposób zorganizowany i zautomatyzowany
- kolejność i parametry uruchamianych węzłów zapisywane są w plikach w formacie XML z rozszerzeniem .launch
- roslaunch automatycznie uruchamia roscore, jeśli nie jest on już uruchomiony

Składnia

```
roslaunch <pakiet> <plik.launch>
```

Przykład

```
roslaunch turtlesim turtlesim_with_color.launch
```

turtlesim_with_color.launch

```
<launch> <!-- Uruchomienie węzła turtlesim -->
<node name="turtlesim_node" pkg="turtlesim" type="turtlesim_node"
   output="screen">
  <!-- Ustawienie koloru tła (czerwony) -->
  <param name="background_r" value="255"/>
  <param name="background_g" value="0"/>
  <param name="background_b" value="0"/></node>
</launch>
```

ROS

Ważniejsze polecenia

Polecenia

ROS

- opiera się na poleceniach tekstowych wpisywanych ręcznie
- polecenia wpisywane są w oknie terminalu Ubuntu (odpowiednik wiersza polecenia w Windows)
- niektóre polecenia pozostają aktywne i „zajmują” terminal do momentu kiedy nie zostaną zakończone, uruchomienie kolejnego polecenia może wymagać otwarcia nowego okno terminala
- w Ubuntu nowe okno terminala można otworzyć kombinacją klawiszy Ctrl + Alt + T
- terminal Ubuntu obsługuje **autouzupełnianie** klawiszem Tab, po wpisaniu początkowego fragmentu i naciśnięciu Tab terminal:
  - uzupełnia pozostałą część polecenia (jeśli jest jednoznaczna)
  - wyświetla możliwe opcje (jeśli jest kilka)

Polecenia

Polecenia uruchamiające

polecenie	wyjaśnienie	opis
roscore	ros+core	uruchamianie kluczowych elementów ROS
rosrun	ros+run	uruchomienie węzła
roslaunch	ros+launch	uruchomienie wielu węzłów i ustawianie opcji

Polecenia informacyjne

polecenie	wyjaśnienie	funkcje	Opis
rostopic	ros+topic	bw, echo, find, hz, info, list pub, type	lista tematów (list), informacje o temacie (bw, hz, info, type), nasłuch (echo) lub publikacja tematu (pub)
rosservice	ros+service	args, call, find, info, list, type, uri	lista serwisów (list), informacje o serwisie (args, info, type, uri), wywołanie serwisu (call)
rosnode	ros+node	cleanup, info, kill, list, machine, ping	lista węzłów (list), usunięcie węzła (kill, cleanup), informacja (info, machine, ping)
rosparam	ros+param	delete, dump, get, list, load set	lista parametrów (list), informacje o parametrze (get, dump), ustawienie parametru (set, load), kasowanie (delete)
rosmsg	ros+msg	list, md5, package, packages, show	listy komunikatów: wszystkich, z pakietu (list, package), informacje o komunikacie (show, md5),
rossrv	ros+srv	list, md5, package, packages, show	listy i informacje o serwisach (jak dla komunikatów)

Polecenia

Terminal 1

```
$ roscore
```

Terminal 2

```
$ rosnode list
/rosout
```

→ lista węzłów
jest tylko węzeł: **rosout**

Terminal 2

```
$rosnode info /rosout
--------------------------------------------------------------------------------
Node [/rosout]

Publications:
 * /rosout_agg [rosgraph_msgs/Log]

Subscriptions:
 * /rosout [unknown type]

Services:
 * /rosout/get_loggers
 * /rosout/set_logger_level
```

→ informacje o węźle **rosout**

- nadaje w temacie
  /rosout_agg
  (typ tematu: rosgraph_msgs/Log)
- odbiera w temacie
  /rosout
  (typ tematu nieznany – nikt jeszcze nie publikuje)
- obsługuje serwisy
  /rosout/get_loggers
  /rosout/set_logger_level

Polecenia

Terminal 2

```
$ rostopic list
/rosout
/rosout_agg
```

→ lista tematów,
są 2 tematy:
- /rosout
- /rosout_agg

Terminal 2

```
$ rostopic info /rosout_agg
Type: rosgraph_msgs/Log

Publishers:
 * /rosout (http://ubuntu:38269/)

Subscribers: None
```

→ informacje o temacie **rosout_agg**
- typ tematu: rosgraph_msgs/Log
- nadawcy: /rosout
- odbiorcy: brak

Polecenia

Terminal 2

```
$ rosservice list
/rosout/get_loggers
/rosout/set_logger_level
```

lista serwisów,
są 2 serwisy:
- /rosout/get_loggers
- /rosout/set_logger_level

Terminal 2

```
$ rosservice info /rosout/get_loggers
Node: /rosout
URI: rosrpc://ubuntu:36753
Type: roscpp/GetLoggers
Args:
```

informacje o serwisie **rosout/get_loggers**
- typ serwisu: roscpp/GetLoggers
- wykonawca: /**rosout**
- argumenty: brak

ROS

Narzędzia graficzne

rqt

rqt (ROS Qt-based GUI Framework)

narzędzie, które zapewnia graficzny interfejs użytkownika dla aplikacji ROS (Qt – zestaw bibliotek i narzędzi, których podstawowym elementem są klasy do budowy graficznych interfejsów programów komputerowych), składa się z trzech części/metapakietów

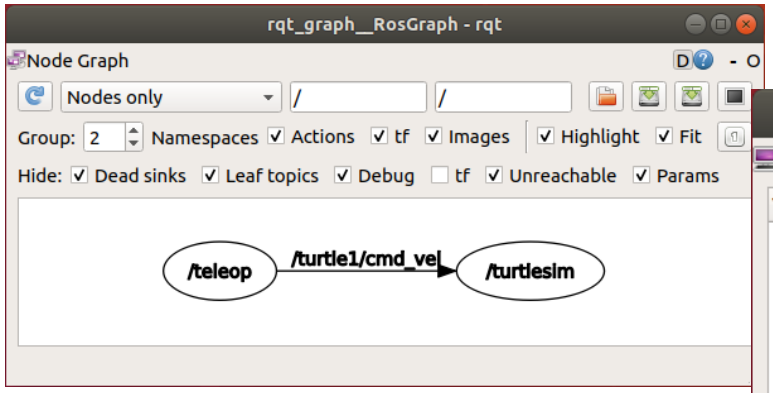
Podstawowe elementy

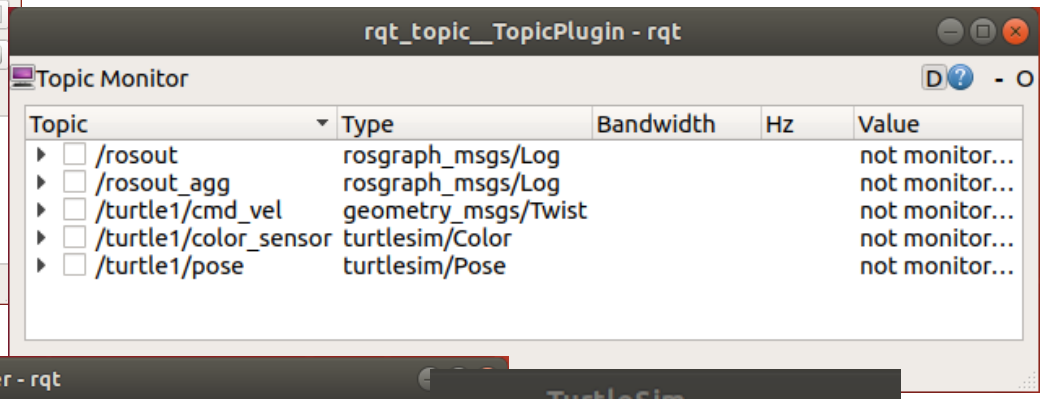
metapakiet	opis
rqt	okno kontenera pozwalające na dokowanie narzędzi rqt
rqt_common_plugins	narzędzia wykorzystywane w i poza środowiskiem uruchomieniowym robota
rqt_robot_plugins	narzędzia wykorzystywane podczas działania robota

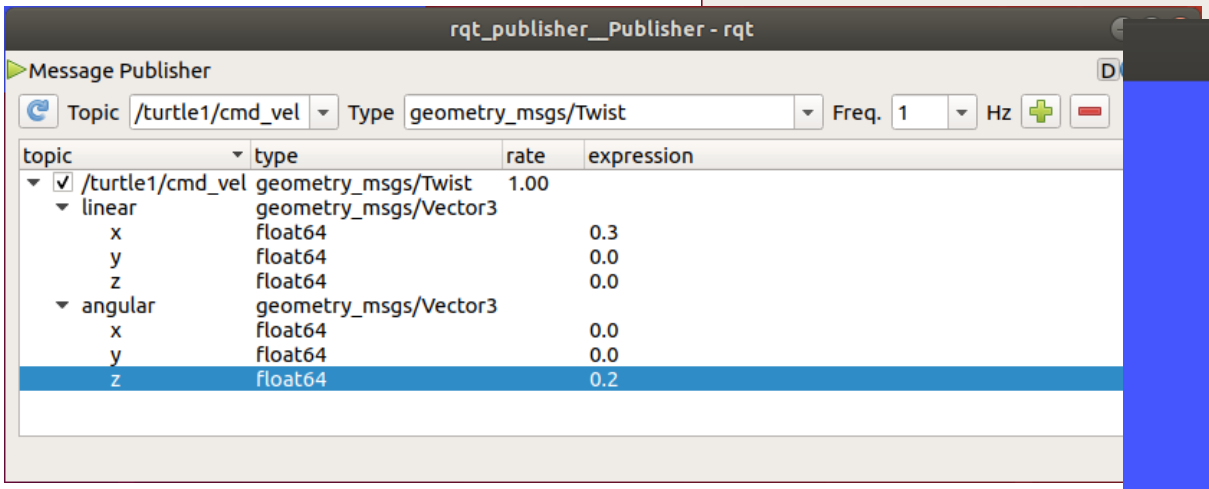
```
ros@ubuntu:~$ rosrun rqt_
rqt_action          rqt_gui_cpp         rqt_nav_view          rqt_robot_dashboard   rqt_srv
rqt_bag             rqt_gui_py          rqt_plot              rqt_robot_monitor     rqt_tf_tree
rqt_bag_plugins     rqt_image_view      rqt_pose_view         rqt_robot_steering    rqt_top
rqt_console         rqt_launch          rqt_publisher         rqt_runtime_monitor   rqt_topic
rqt_dep             rqt_logger_level    rqt_py_common         rqt_rviz              rqt_web
rqt_graph           rqt_moveit          rqt_py_console        rqt_service_caller
rqt_gui             rqt_msg             rqt_reconfigure       rqt_shell
```

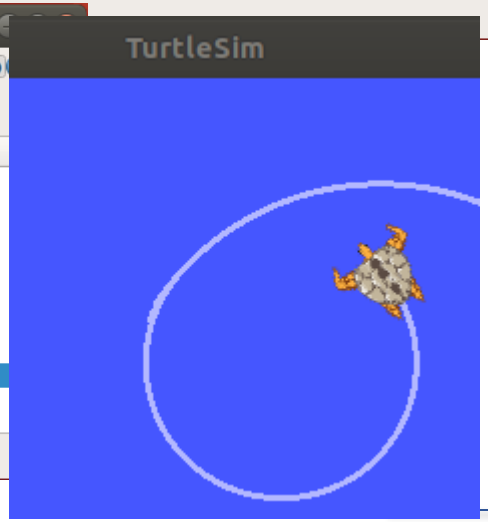
rqt_common_plugins

rqt_action rqt_bag rqt_bag_plugins rqt_console rqt_dep rqt_graph
rqt_image_view rqt_launch rqt_logger_level rqt_msg rqt_plot rqt_publisher
rqt_py_common rqt_py_console rqt_reconfigure rqt_service_caller rqt_shell
rqt_srv rqt_top rqt_topic rqt_web



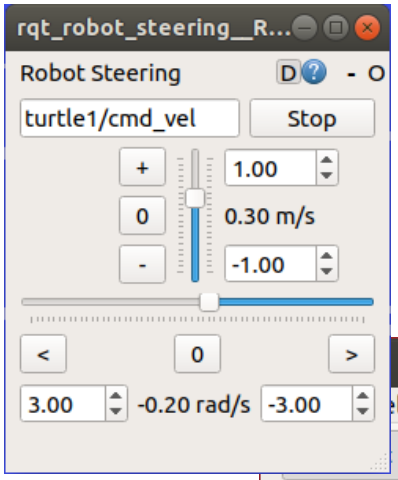


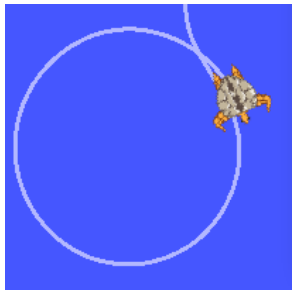


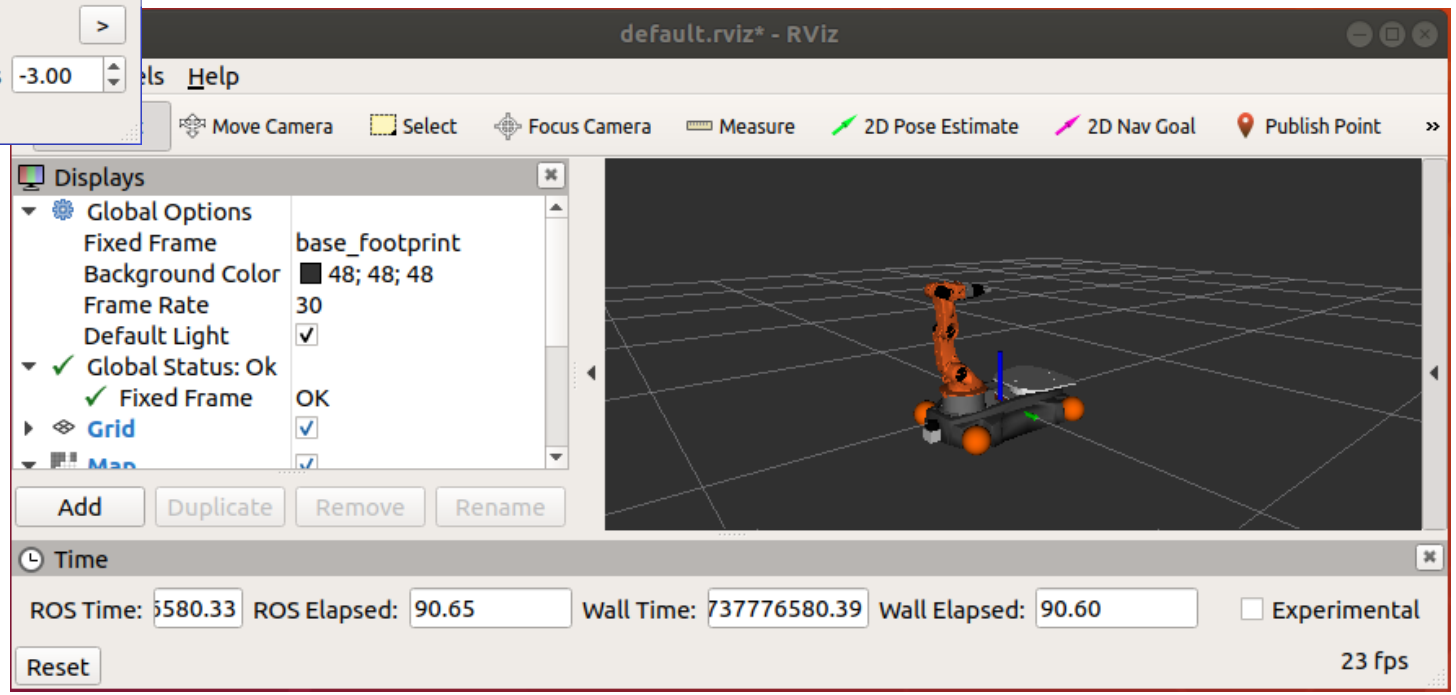


rqt_robot_plugins

rqt_moveit rqt_nav_view rqt_pose_view rqt_robot_dashboard rqt_robot_monitor
rqt_robot_steering rqt_runtime_monitor rqt_rviz rqt_tf_tree







rqt_robot_steering – turtlesim

Terminal 1

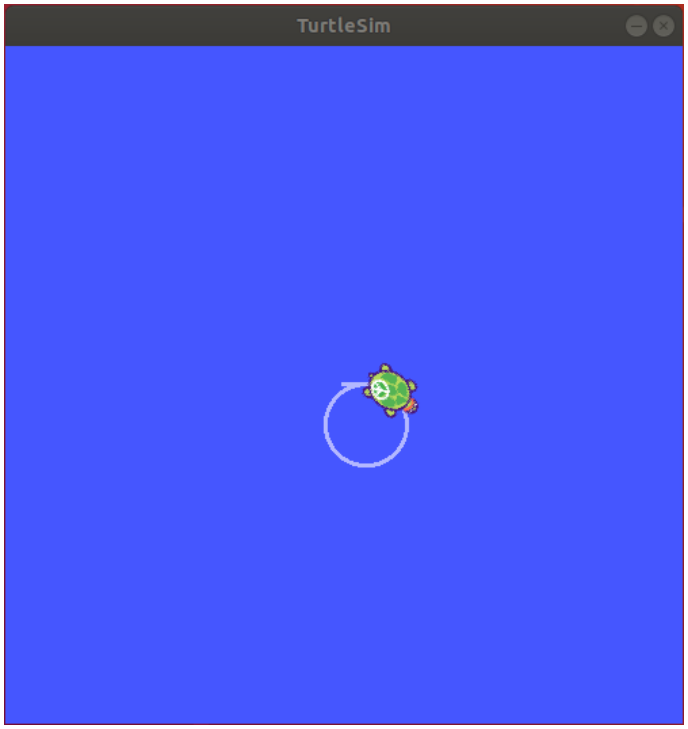
```
$ roscore
```

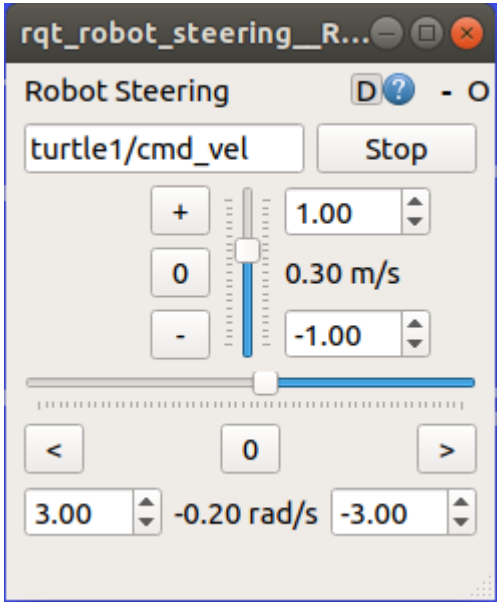
Terminal 2

```
$ rosrun turtlesim turtlesim_node
```

Terminal 3

```
$ rosrun rqt_robot_steering rqt_robot_steering
```





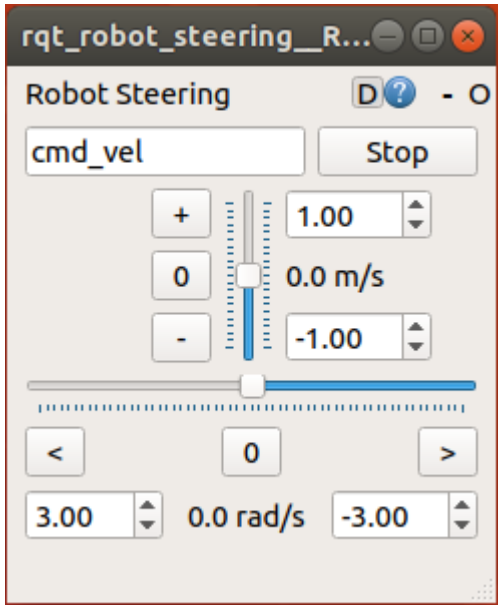
rqt_robot_steering – youbot

Terminal 1

$ **roslaunch** youbot_driver_ros_interface youbot_driver.launch

Terminal 2

$ **rosrun** rqt_robot_steering rqt_robot_steering



ROS
konfiguracja

Konfiguracja ROS

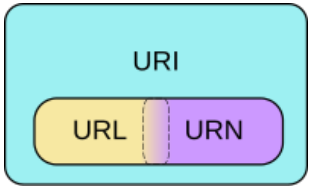
ROS_MASTER_URI

zmienna środowiskowa określająca adres URI ROS Master-a

ROS_IP/ROS_HOSTNAME

zmienne środowiskowe określające adres sieciowy węzła lub narzędzia ROS, zmienne wykluczają się wzajemnie, ROS_HOSTNAME ma wyższy priorytet

Ustawienie zmiennych jest opcjonalne, jeśli nie są ustawione wskazują na lokalny komputer na którym uruchomiony jest roscore.



WikipediA

URI – Ujednolicony Identyfikator Zasobów (ang. *Uniform Resource Identifier*) jest standardem internetowym umożliwiającym łatwą identyfikację zasobów w sieci. URI jest zazwyczaj łańcuchem znaków, zapisanym zgodnie ze składnią określoną w standardzie. Łańcuch ten określa nazwę (URN ang. *Uniform Resource Name*) lub adres (URL ang. *Uniform Resource Locator*) zasobu, identyfikowanego przez dany URI.

ROS – model single master

ROS

- węzły mogą być uruchamiane na różnych maszynach (pozwala to na dopasowanie systemu do dostępnych zasobów), istnieją wyjątki: węzeł sterownika, który komunikuje się z częścią sprzętu, musi działać na maszynie, do której sprzęt jest fizycznie podłączony
- wszystkie węzły muszą używać tego samego ROS Mastera (za pośrednictwem ROS_MASTER_URI)
- musi istnieć pełna, dwukierunkowa łączność między wszystkimi parami maszyn, na wszystkich portach

ROS_MASTER_URI:
http://10.0.0.100:11311
ROS_IP: 10.0.0.100

ROS_MASTER_URI:
http://10.0.0.100:11311
ROS_IP: 10.0.0.101

ROS_MASTER_URI:
http://10.0.0.100:11311
ROS_IP: 10.0.0.102

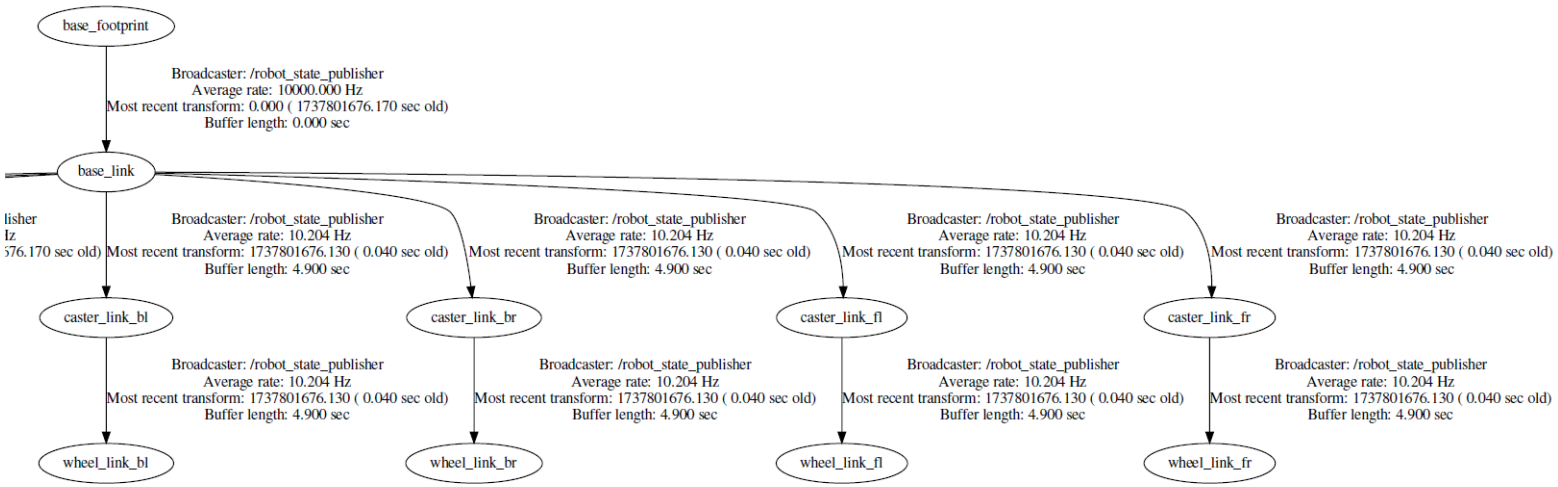
ROS
układy współrzędnych

tf, tf2

tf (ang. transformation frame)

- przechowuje relacje między układami współrzędnych
- umożliwia przekształcanie punktów, wektorów itp. między dwoma dowolnymi układami współrzędnych w dowolnym momencie

tf2 druga generacja pakietu **tf**



Publishery tf

robot_state_publisher

- pakiet, który umożliwia publikowanie drzewa **tf2** na podstawie opisu robota w formacie **URDF** (ang. Unified Robot Description Format)
- używany do wizualizacji i przetwarzania pozycji robota w czasie rzeczywistym
- jeżeli robot ma przeguby ruchome robot_state_publisher do generowania układów potrzebuje informacji o stanach przegubów robota, które w ROS publikuje joint_state_publisher

URDF

my_robot_model.urdf

```
<?xml version="1.0 " ?>
<robot name= "my_robot_platform">
  <material name="red">
    <color rgba="0.8 0.0 0.0 1.0"/>
  </material>
  <material name="green">
    <color rgba="0.0 0.8 0.0 1.0"/>
  </material>
  <material name="blue">
    <color rgba="0.0 0.0 1.0 1.0"/>
  </material>
  <material name="yellow">
    <color rgba="1.0 1.0 0.0 1.0"/>
  </material>
```

URDF

my_robot_model.urdf

```
<!-- Podstawowa platforma robota -->
 <link name="base_link"><visual>
    <geometry><cylinder radius="0.2" length="0.05"/></geometry>
    <material name="blue"/>
 </visual></link>
 <!-- Lewe koło -->
 <link name="left_wheel"><visual>
    <geometry><cylinder radius="0.1" length="0.05"/></geometry>
    <material name="yellow"/>
 </visual></link>
 <!-- Prawe koło -->
 <link name="right_wheel"><visual>
    <geometry><cylinder radius="0.1" length="0.05"/></geometry>
    <material name="yellow"/>
 </visual></link>
 <!-- Przednie kółko (caster) -->
 <link name="chasis_caster"><visual>
    <geometry><sphere radius="0.05"/></geometry>
    <material name="red"/>
 </visual></link>
 <!-- Czujnik -->
 <link name="chasis_sensor"><visual>
    <geometry><sphere radius="0.05"/></geometry>
    <material name="green"/>
 </visual></link>
```

URDF

my_robot_model.urdf

```
<!-- Połączenie lewego koła -->
  <joint name="left_wheel_joint" type="fixed">
    <parent link="base_link"/>
    <child link="left_wheel"/>
    <origin xyz="0 -0.2 0" rpy="1.5707 0 0"/>
  </joint>
  <!-- Połączenie prawego koła -->
  <joint name="right_wheel_joint" type="fixed">
    <parent link="base_link"/>
    <child link="right_wheel"/>
    <origin xyz="0 0.2 0" rpy="1.5707 0 0"/>
  </joint>
  <!-- Połączenie kółka caster -->
  <joint name="chasis_caster_joint" type="fixed">
    <parent link="base_link"/>
    <child link="chasis_caster"/>
    <origin xyz="0.15 0 -0.05" rpy="0 0 0"/>
  </joint>
  <!-- Połączenie sensora -->
  <joint name="chasis_sensor_joint" type="fixed">
    <parent link="base_link"/>
    <child link="chasis_sensor"/>
    <origin xyz="0.15 0 0.05" rpy="0 0 0"/>
  </joint>

</robot>
```

ROS

pierwszy pakiet

catkin

catkin

- framework do budowania pakietów w ROS (oparty na CMake)
- obsługuje kody w C++ i python-ie
- uwzględnia zależności pomiędzy pakietami (dba o kolejność budowania pakietów)
- tworzy przestrzeń roboczą (workspace) pakietu

Terminal

```
$ mkdir -p ~/myrobot_ws/src
$ cd ~/myrobot_ws/src
$ catkin_create_pkg my_robot std_msgs rospy
$ cd ~/myrobot_ws
$ catkin_make
$ echo "source ~/myrobot_ws/devel/setup.bash" >> ~/.bashrc
```

myrobot_ws
  nazwa workspace-a, zwykle catkin_ws

my_robot
  nazwa pakietu

std_msgs rospy
  pakiety od których tworzony pakiet jest zależny

pakiet my_robot

```
ros@ubuntu:~/myrobot_ws$ tree -d
├── build
│   ├── atomic_configure
│   ├── catkin
│   │   └── catkin_generated
│   │       └── version
│   ├── catkin_generated
│   │   ├── installspace
│   │   └── stamps
│   │       └── Project
│   ├── CMakeFiles
│   │   ├── 3.10.2
│   │   │   ├── CompilerIdC
│   │   │   │   └── tmp
│   │   │   └── CompilerIdCXX
│   │   │       └── tmp
│   │   ├── clean_test_results.dir
│   │   ├── CMakeTmp
│   │   ├── download_extra_data.dir
│   │   ├── doxygen.dir
│   │   ├── run_tests.dir
│   │   └── tests.dir
│   ├── gtest
│   │   ├── CMakeFiles
│   │   └── googlemock
│   │       ├── CMakeFiles
│   │       │   ├── gmock.dir
│   │       │   │   ├── _
│   │       │   │   │   └── googletest
│   │       │   │   │       └── src
│   │       │   │   └── src
│   │       │   └── gmock_main.dir
│   │       │       ├── _
│   │       │       │   └── googletest
│   │       │       │       └── src
│   │       │       └── src
│   │       │           └── gtest
│   │       │               └── CMakeFiles
│   │       │                   ├── gtest.dir
│   │       │                   │   └── src
│   │       │                   └── gtest_main.dir
│   │       │                       └── src
│   │       │
│   ├── my_robot
│   │   ├── catkin_generated
│   │   │   ├── installspace
│   │   │   └── stamps
│   │   │       └── my_robot
│   │   └── CMakeFiles
│   │       ├── std_msgs_generate_messages_cpp.dir
│   │       ├── std_msgs_generate_messages_eus.dir
│   │       ├── std_msgs_generate_messages_lisp.dir
│   │       ├── std_msgs_generate_messages_nodejs.dir
│   │       └── std_msgs_generate_messages_py.dir
│   └── test_results
├── devel
│   ├── lib
│   │   └── pkgconfig
│   └── share
│       └── my_robot
│           └── cmake
└── src
    └── my_robot
        └── src

62 directories
```

my_robot: turtlesim.launch

Terminal

```
$ cd ~/myrobot_ws/src/my_robot
$ mkdir launch
$ gedit launch/trutlesim.launch
```

turtlesim.launch

```
<?xml version="1.0"?>

<launch>
   <!-- Uruchomienie turtlesim_node -->
   <node pkg="turtlesim" type="turtlesim_node" name="turtlesim" output="screen" />

   <!-- Uruchomienie teleop do sterowania za pomocą klawiatury -->
   <node pkg="turtlesim" type="turtle_teleop_key" name="teleop" output="screen" />
</launch>
```

Terminal

```
$ roslaunch my_robot trutlesim.launch
```

my_robot: turtlesim_robot.launch

turtlesim.launch

```
<?xml version="1.0"?>

<launch>
  <!-- Załadowanie parametrów do ROS -->
  <rosparam file="$(find my_robot)/urdf/my_robot_model.urdf" command="load"
            param="robot_description"/>
  <!-- Uruchomienie robot_state_publisher -->
  <node name="robot_state_publisher" pkg="robot_state_publisher"
        type="robot_state_publisher" output="screen">
    <param name="use_sim_time" value="false"/>
  </node>
  <!-- Uruchomienie RViz-->
  <node name="rviz" pkg="rviz" type="rviz"/>
</launch>
```

Terminal

```
$ roslaunch my_robot trutlesim_robot.launch
```

my_robot: turtle_tf_broadcaster.py

Terminal

```
$ cd ~/myrobot_ws/src/my_robot
$ gedit scripts/turtle_tf_broadcaster.py
```

turtle_tf_broadcaster.py

```
#!/usr/bin/env python
import roslib
import rospy
import tf
from turtlesim.msg import Pose

def pose_callback(msg):
    br = tf.TransformBroadcaster()
    br.sendTransform((msg.x, msg.y, 0),
                     tf.transformations.quaternion_from_euler(0, 0, msg.theta),
                     rospy.Time.now(), "base_link", "world")

if __name__ == "__main__" :
    rospy.init_node("turtle_tf_broadcaster")
    rospy.Subscriber("/turtle1/pose", Pose, pose_callback)
    rospy.spin()
```

my_robot: turtle_tf_broadcaster.py

Terminal

```
$ cd ~/myrobot_ws/src/my_robot
$ chmod +x scripts/turtle_tf_broadcaster.py
```

Terminal

```
$ chmod +x scripts/turtle_tf_broadcaster.py
```