

Instalacja Linuxa Ubuntu 20.04 LTS

Wskazówki

Jeśli masz zamiar używać Linuxa tylko do ROS, to chyba lepiej skorzystać z maszyny wirtualnej. Do wirtualizacji polecam VMware Player – nie miałem z nim żadnych problemów, w przeciwieństwie do Oracle VirtualBoxa. Inna opcja to dual boot z Windowsem i Linuxem.

Przy instalacji Ubuntu zwróć uwagę na wystarczającą pojemność dysku twardego (myślę, że co najmniej z 25GB – ja maksymalny rozmiar wirtualnego dysku twardego maszyny ustawiłem na 40GB).

W Linuxie jest tzw. folder domowy – ma ścieżkę „/home” i jej często używany alias „~/”.

Polecam też zainstalować sobie jakiś lepszy klient terminala, np. Terminator.

```
sudo apt-get install terminator
```

UWAGA: Poniżej są gotowe komendy do wklejania, jeśli:

a) nie jest zaindentowana (tzn. wcięta/przesunięta) – możesz ją po prostu wkleić i powinna działać

b) jest zaindentowana, to znaczy że omawiam alternatywne komendy i musisz się wczytać o co chodzi.

Instalacja i konfiguracja ROS Noetic Ninjemys na Ubuntu 20.04

0. Konfiguracja repozytoriów Ubuntu (raczej niepotrzebna)

Sam nie miałem żadnych problemów, ale tutorial instalacji ROS wspomina, że mogą się takie zdarzyć. Jeśli je masz, to chyba musisz skonfigurować ustawienia repozytoriów Ubuntu, tak aby pozwalać korzystać z repozytoriów "restricted," "universe," i "multiverse." Pod [tym linkiem](#) znajdziesz instrukcje jak to zrobić.

1. Dodanie repozytorium z ROS

Poniższa komenda dodaje możliwość pobierania paczek z packages.ros.org.

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

2. Instalacja *curl* i pobranie

```
sudo apt install curl
```

```
curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
```

3. Instalacja ROS Noetic

Najlepiej zaktualizować sobie indeksy paczek linuxowych (Linux package index) poniższą komendą:

```
sudo apt update
```

Teraz wybierz sobie edycję ROS do instalacji. Dla wygody najlepiej zainstalować sobie pełną (Desktop Full).

- **Edycja pełna:** Wszystko, co poniżej plus symulatory 2D/3D i paczki percepcji 2D/3D

```
o sudo apt install ros-noetic-desktop-full
```

- **Edycja Desktop:** Wszystko co w edycji bazowej poniżej, plus narzędzia [rqt](#) and [rviz](#), i chyba kilka innych.

```
o sudo apt install ros-noetic-desktop
```

- **ROS-Base: (edycja podstawowa)** Podstawowe paczki ROS, narzędzia do budowania, bez GUI. Nie warto jej teraz wybierać.

```
o sudo apt install ros-noetic-ros-base
```

Zawsze można sobie doinstalować dowolną paczkę poniższą komendą.

```
sudo apt install ros-noetic-NAZWAPACZKI
```

na przykład:

```
sudo apt install ros-noetic-slam-gmapping
```

Setup środowiska ROS

Poniższy skrypt trzeba source'ować w **każdym terminalu, w którym używamy ROS:**

```
source /opt/ros/noetic/setup.bash
```

Najwygodniej jest automatycznie source'ować ten skrypt za każdym odpaleniem konsoli. Poniższa komenda właśnie do tego służy.

```
echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Zależności do budowania paczek

Jeśli tu jesteś, to znaczy, że masz zainstalowane podstawowe paczki ROS. Teraz pora zrobić ROS workspace, czyli folder gdzie są przechowywane twoje paczki – ich kody źródłowe, buildy, itd.

Jest narzędzie [rosinstall](#), służące do pobierania brakujących zależności (tj. paczek potrzebnych do skorzystania z określonej paczki) za pomocą jednej komendy.

Poniższa komenda instaluje to i inne narzędzia do zarządzania ROSowymi paczkami:

```
sudo apt install python3-rosdep python3-rosinstall python3-rosinstall-generator python3-wstool build-essential
```

Inicjalizacja rosdep'a

Rosdep jest potrzebny do uruchamiania niektórych krytycznych komponentów ROS, i pozwala na łatwe zarządzanie i instalowanie zależności. Poniższa komenda go instaluje:

```
sudo apt install python3-rosdep
```

Po zainstalowaniu rosdep'a, musisz go jeszcze zainicjować i zaktualizować

```
sudo rosdep init  
rosdep update
```

Konfiguracja workspace'a do robota Dzik

1. Utworzenie workspace'a o nazwie *dzik_ws*

Tworzymy nowy ROSowy workspace, o nazwie np. *dzik_ws* (bo zawierać będzie wszystkie paczki potrzebne do robota Dzik) za pomocą poniższych komend:

```
mkdir -p ~/dzik_ws/src  
cd ~/dzik_ws  
catkin_make
```

Następnie trzeba będzie zsource'ować plik bash workspace'u, aby móc w nim pracować, ale to tym w następnym punkcie.

Sourcing polega na wywołaniu skryptu i zachowaniu zmiennych i funkcji w instancji terminala (powłoki – ang. shell), np., dzięki zsource'owaniu `/opt/ros/noetic/setup.bash` po instalacji ROS, możemy w terminalu korzystać z poleceń `roscd` i `rosls`.

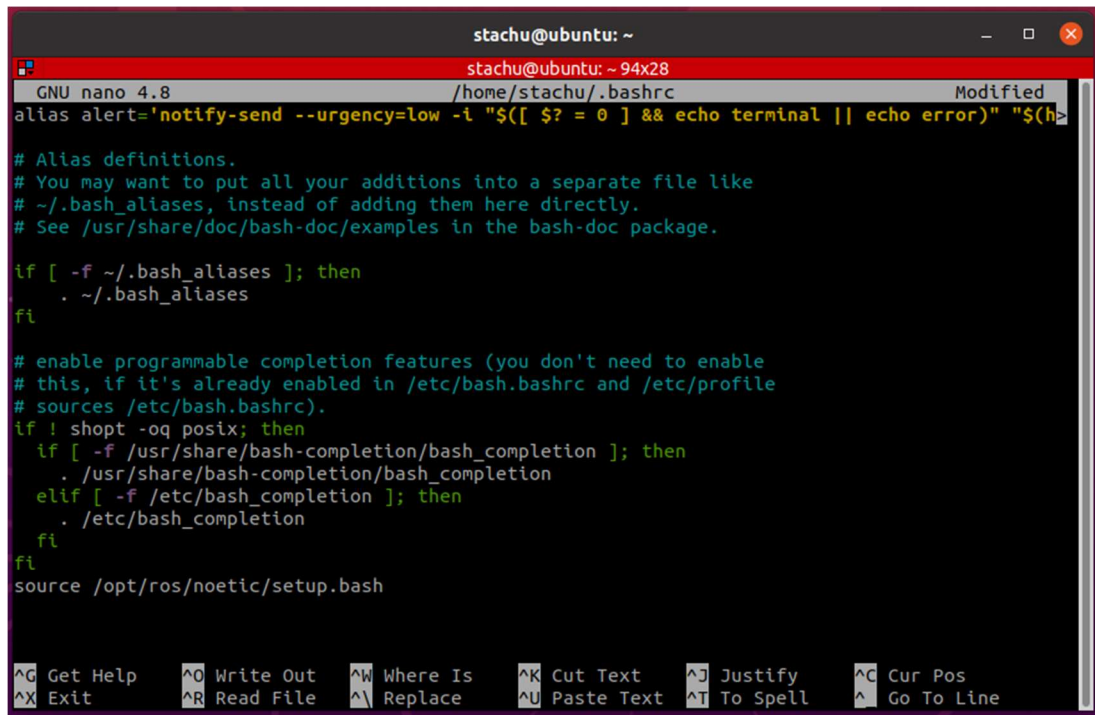
2. Nowa instalacja ROS vs istniejąca instalacja ROS

Jeśli właśnie pierwszy raz zainstalowałeś ROS na swoim komputerze, to najprawdopodobniej nie masz stworzonego żadnego ROSowego workspace'a (domyślnie nazywa się on `catkin_ws` i jest w katalogu domowym użytkownika (tj. `~/catkin_ws` lub innymi słowy `/home/NazwaUżytkownika/catkin_ws`). W takim przypadku nie musisz nic robić.

Jeśli masz już jakiś ROSowy workspace, to zapewne jest on automatycznie source'owany podczas uruchomienia nowej instancji terminala (w pliku `~/.bashrc`). Możesz to sprawdzić otwierając go poniższym poleceniem:

```
nano ~/.bashrc
```

Poniżej skrin z moim plikiem `~/.bashrc`, jak widać nie mam source'owanych żadnych workspace'ów.



```
stachu@ubuntu: ~
GNU nano 4.8 /home/stachu/.bashrc Modified
alias alert='notify-send --urgency=low -i "[ $? = 0 ] && echo terminal || echo error" "$(h>

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
    if [ -f /usr/share/bash-completion/bash_completion ]; then
        . /usr/share/bash-completion/bash_completion
    elif [ -f /etc/bash_completion ]; then
        . /etc/bash_completion
    fi
fi
source /opt/ros/noetic/setup.bash

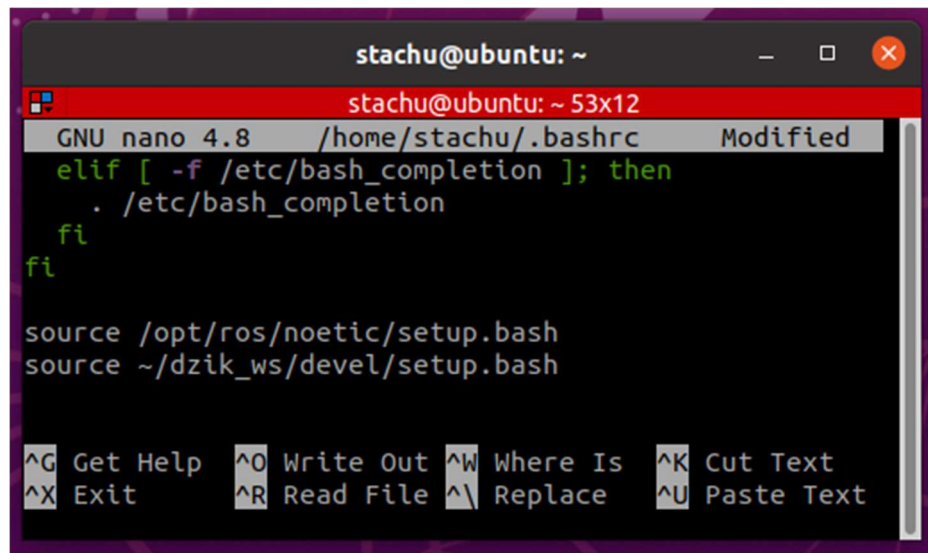
^G Get Help  ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos
^X Exit      ^R Read File  ^\ Replace    ^U Paste Text ^T To Spell   ^_ Go To Line
```

Jak widać powyżej, nie source'uję automatycznie żadnego workspace'a, a jedynie sam skrypt z poleceniami ROS.

- a) Jeśli w twoim pliku `~/.bashrc` nie jest source'owany żaden ROSowy workspace, to poniższymi komendami możesz dodać go do tego pliku, aby sam się source'ował podczas uruchamiania każdej nowej instancji terminala.

```
echo "source ~/dzik_ws/devel/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Po wywołaniu powyższych komend, odnośny fragment twojego pliku `~/.bashrc` powinien wyglądać tak:



```
stachu@ubuntu: ~
GNU nano 4.8 /home/stachu/.bashrc Modified
elif [ -f /etc/bash_completion ]; then
  . /etc/bash_completion
fi

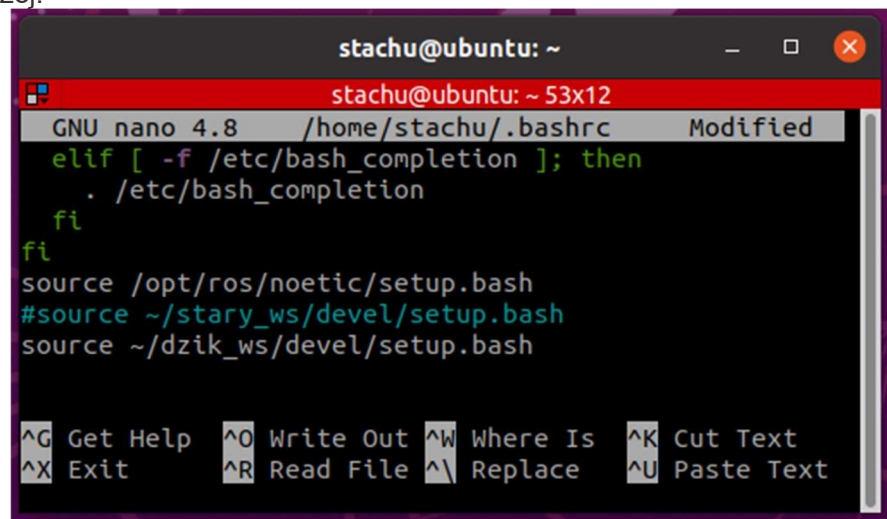
source /opt/ros/noetic/setup.bash
source ~/dzik_ws/devel/setup.bash

^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text
^X Exit      ^R Read File  ^\ Replace   ^U Paste Text
```

- b) Jeśli w twoim pliku `~/.bashrc` jest już `source`'owany jakiś workspace, to zgrubsza masz dwie opcje:
- a) nic nie zmieniać, i ręcznie `source`'ować `dzik_ws` przy każdym uruchomieniu nowej instancji terminala (lub odpalać terminal skryptem, który robi to za nas), poniższą komendą:

```
source ~/dzik_ws/devel/setup.bash
```

- b) jeśli nie korzystasz ze „starego” workspace’a, możesz zakomentować/usunąć komendę, która odpowiada za jego sourcing albo po prostu `source`'ować nowy workspace w następnej linijce pliku, ponieważ `source`'owanie innego workspace’u nadpisuje zmienne i funkcje w terminalu po starym workspace’ie. Wtedy odnośny fragment twojego pliku `~/.bashrc` mógłby wyglądać tak jak poniżej:



```
stachu@ubuntu: ~
GNU nano 4.8 /home/stachu/.bashrc Modified
elif [ -f /etc/bash_completion ]; then
  . /etc/bash_completion
fi

source /opt/ros/noetic/setup.bash
#source ~/stary_ws/devel/setup.bash
source ~/dzik_ws/devel/setup.bash

^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text
^X Exit      ^R Read File  ^\ Replace   ^U Paste Text
```

Wskazówka: Jeśli planujesz pracować równolegle nad oprogramowaniem do innego robota, zalecane jest stworzenie dla niego oddzielnego workspace’a.

3. Pobranie paczek do robota Dzik z GitHuba

Nadszedł czas na pobranie z GitHuba paczek do Dzika do naszego workspace'a. Najpierw upewnij się, że masz dostęp do repozytorium pod linkiem <https://github.com/WalecznaLama/Dzik>, tj. jest ono publiczne, bądź (jeśli jest prywatne) jesteś zalogowany w programie Git na swoje konto GitHub, i jesteś dodany do repozytorium jako contributor. Póki co, to repozytorium jest publiczne, więc do rzeczy. Robimy to przy pomocy poniższych komend:

```
cd ~/dzik_ws/  
git clone https://github.com/WalecznaLama/Dzik  
mv -f Dzik/{.,}* src/
```

4. Instalacja symulatora Gazebo

Sprawdź czy posiadasz zainstalowany symulator Gazebo:

```
gazebo --version
```

Jeśli nie, to instalujemy go poniższą komendą:

```
sudo apt-get install ros-noetic-gazebo-ros-pkgs ros-noetic-gazebo-ros-control
```

5. Pobieranie zależności do projektu, instalacja catkin build i paczki joint_state_controller

Wszystkie potrzebne zależności możemy pobrać poniższą komendą:

```
cd ~/dzik_ws  
rosdep install --from-paths src --ignore-src -r -y
```

Następnie musimy zainstalować bibliotekę *joint_state_controller*:

```
sudo apt-get install ros-noetic-joint-state-controller
```

Oraz zainstalować *catkin build*, poniższą komendą:

```
sudo apt install python3-catkin-tools python3-osrf-pycommon
```

6. Budowanie projektu

Usuwamy wszystkie nie-ukryte foldery z workspace'u ~/dzik_ws z wyjątkiem folderu *src*:

```
cd ~/dzik_ws  
rm -v -r !("src"|"catkin_workspace")
```

Następnie budujemy zawartość workspace'u z użyciem *catkin build*, może to zająć kilka minut:

```
cd ~/dzik_ws  
catkin build
```

7. Uruchamianie symulacji

Otwórz nowe okno terminala (albo zsource'uj `~/dzik_ws/devel/setup.bash`) i wywołaj poniższą komendę:

```
roslaunch argo_mini startup.launch simulation:=true
```

Symulacja powinna się uruchomić, mogą się pojawiać jakieś błędy w konsoli i Rviz.

WERSJA 1.01

poprawione niektóre komendy, dodana wskazówka co do lepszego terminala

WERSJA 1.0