# Inteligentne Maszyny

## Skrypt do laboratorium ROS

Semestr: 2022 L

Autor: Wojciech Dudek

# Spis treści

SEKCJA 1

# Przygotowanie środowiska

## 1. Środowisko prekonfigurowane – Docker

### Pobranie i uruchomienie środowiska

Środowisko jest udostępnione jako obraz systemu w repozytorium Docker. Podstawową pomoc i samouczki dotyczące obsługi narzędzia Docker można znaleźć pod linkiem:
https://docs.docker.com/get-started/
Adres repozytorium, w którym przechowywany jest obraz środowiska wykorzystywanego na przedmiocie IMA znajduje się poniżej:
https://hub.docker.com/repository/docker/dudekw/siu-base
Aby pobrać obraz i uruchomić kontener z systemem należy pobrać i uruchomić system poprzez wywołanie polecenia:

```
> docker run --name ima -p 6080:80 dudekw/siu-base
```

Uruchomiony w ten sposób system zawiera serwer zdalnego pulpitu. Jest on dostępny na maszynie `host` pod adresem: `localhost:6080`. Wystarczy wpisać ten adres w przeglądarkę internetową, a otworzy nam się zdalny pulpit systemu uruchomionemu w dockerze.

### Korzystanie z narzędzia Docker

1. przesyłanie plików z/do kontenera:
   (a) Sprawdź nazwę uruchomionego kontenera:
   ```
   > docker container list
   ```
   (b) Wykonaj polecenie kopiowania pliku:
   ```
   > docker cp <ścieżka-do-nazwa-pliku> <nazwa kontenera>:<ścieżka-docelowa-pliku>
   ```
2. Zapisywanie zmienionego kontenera do obrazu:
   (a) Sprawdź nazwę uruchomionego kontenera:
   ```
   > docker container list
   ```
   (b) Ustal nazwę/tag obrazu, pod którym chcesz zapisać kontener ze zmianami:
   ```
   > docker images
   ```
   (c) Zapisz aktualny stan kontenera do obrazu:
   ```
   > docker commit <nazwa kontenera> <nazwa-obrazu>:<tag>
   ```

Dodatkowe funkcje, które daje narzędzie docker można poznać zaglądając do dokumentacji:
https://docs.docker.com/get-started/
Środowisko to system Ubuntu w wersji 18.04, więc działają wszystkie polecenia/programy-/biblioteki dostępne dla tego systemu. System ma zainstalowane dodatkowe pakiety systemu ROS:

- ros-melodic-turtlebot3-*
- ros-melodic-telop-twist-keyboard

## 2. Przygotowanie środowiska wirtualnej maszyny, lub natywnego systemu Ubuntu

Można korzystać z innego narzędzia wirtualizacji/konteneryzacji. Można też korzystać z systemu Ubuntu 18.04/20.04 zainstalowanego natywnie. Ważne jest, aby system (instalowany na wirtualnej maszynie/natywnie) miał zainstalowany ROS w wersji melodic (Ubuntu 18.04)/noetic (Ubuntu 20.04) oraz powyższe pakiety dla odpowiedniej wersji ROS (melodic/noetic)

SEKCJA 2

# Sposób użytkowania środowiska

**PRZYDATNE!** **W trakcie pracy na laboratorium potrzebna będzie praca na wielu terminalach, które muszą być odpowiednio skonfigurowane przez polecenie:**
**'`source /root/siu_ws/devel/setup.bash`'**
**W przygotowanym systemie dodano powyższą linię do pliku konfiguracyjnego '/root/.bashrc', więc**
**przy każdym otwarciu terminala na zdalnym pulpicie, niniejsze polecenie będzie już wykonane.**

**UWAGA!** **Korzystając z konsoli systemu host (systemu w którym uruchomiliśmy narzędzie docker) będziemy musieli wykonywać powyższe polecenie przed uruchomieniem skryptów/programów wykorzystujących środowisko żółwi. Aby to zrobić należy przykładowo wykonać poniższe polecenie ('siu' to nazwa kontenera):**

```
> docker exec siu bash -c "source /root/siu_ws/devel/setup.bash && python3 /root/my_script.py"
```

SEKCJA 3

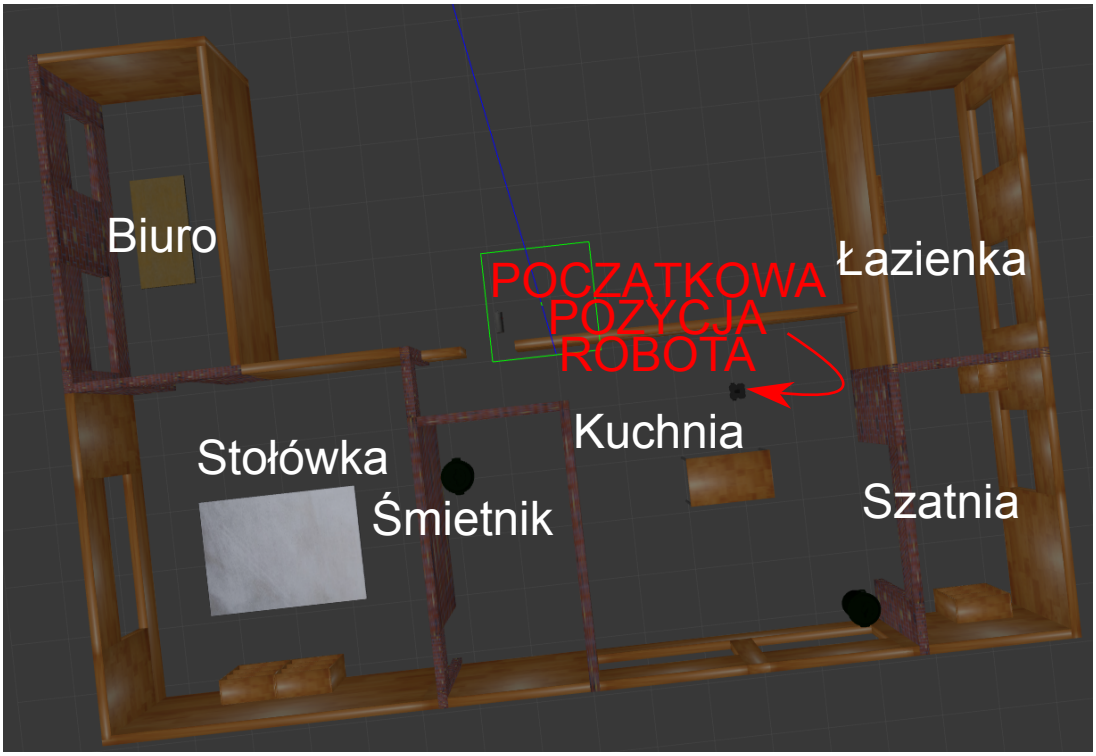
# Zadania

### Zadanie 1 – budowa mapy

Należy:

1. uruchomić symulator robota turtlebot3 w wersji waffle_pi w środowisku domowym:

   ```
   > roslaunch turtlebot3_gazebo turtlebot3_house.launch
   ```




Rysunek 1: Środowisko domowe robota turtlebot3

2. uruchomić węzły SLAM (równoczenej lokalizacji ibudowania mapy):

   ```
   > roslaunch turtlebot3_slam turtlebot3_slam.launch
   ```

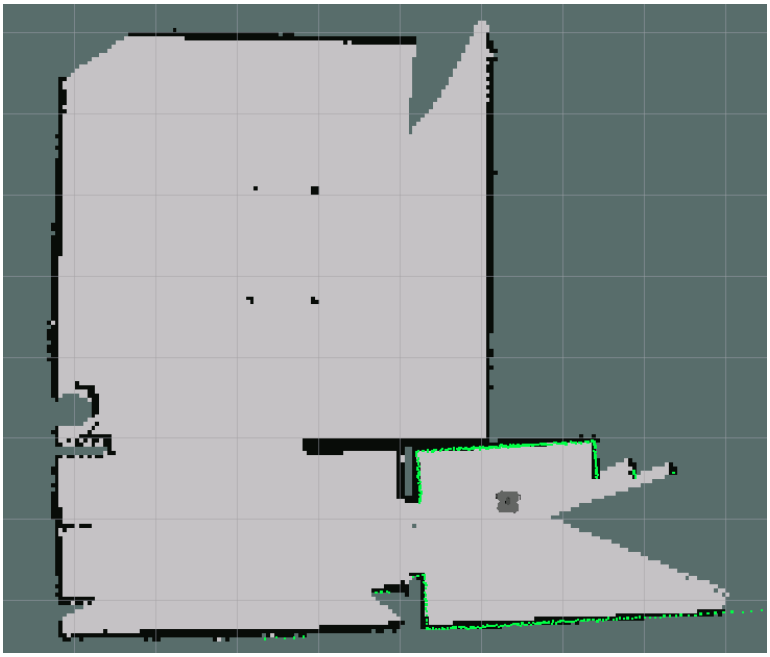
3. uruchomić węzeł zdalnego sterowania z klawiatury:

   ```
   > rosrun teleop_twist_keyboard teleop_twist_keyboard.py
   ```

4. zmniejszyć zadawane prędkości do wartości około (czterokrotnie wciskając przycisk 'z'):

   ```
   speed 0.328     turn:  0.656
   ```

5. sterować zdalnie robotem, aby została zbudowana cała mapa (widoczna w programie Rviz):



Rysunek 2: Przykładowy fragment zbudowanej mapy

6. zapisać mapę wywołując polecenie:

```
> rosrun map_server map_saver -f <BEZWZGLĘDNA-ścieżka-do-własnego-pakietu>/map/house
```

### Zadanie 2 – ruch do zadanego pokoju

Zadanie polega na napisaniu węzła (dowolny język programowania), który wyznaczy odpowiedni cel ruchu dla systemu nawigacji oraz zleci ruch do tego celu w środowisku domowym. Celem ruchu jest pokój, który zostanie podany węzłowi jako argument przy uruchomieniu węzła.
Aby uruchomić gotowy system nawigacji należy:

1. Uruchomić symulację robota w środowisku domowym:

   ```
   > roslaunch turtlebot3_gazebo turtlebot3_house.launch
   ```

2. Uruchomić system nawigacji w podaną ścieżką do zapisanej wcześniej mapy:

   ```
   > roslaunch turtlebot3_navigation turtlebot3_navigation.launch \
                        map_file:=<bezwzględna-ścieżka-do-mapy-z rozszerzeniem-yaml>
   ```

   np:

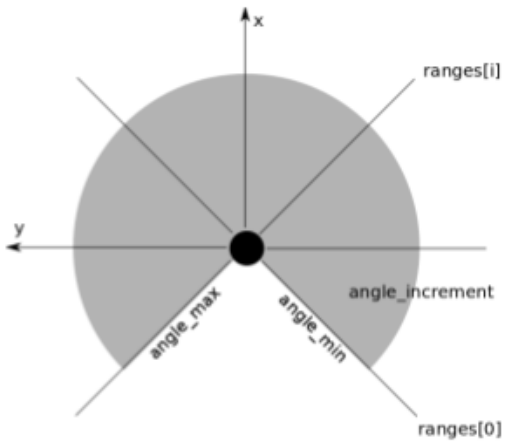
   ```
   > roslaunch turtlebot3_navigation turtlebot3_navigation.launch \
              map_file:=/home/uzytkownik/my_catkin_ws/src/package-123456/map/house.yaml
   ```

Polecenie ruchu do celu wydaje się za pomocą interfejsu akcji ROS:

**Python**:
http://wiki.ros.org/actionlib_tutorials/Tutorials/Writing%20a%20Simple%20Action%20Client%20%28Python%29
**C++**:
http://wiki.ros.org/actionlib_tutorials/Tutorials/SimpleActionClient
Można się wspomagać samouczkiem dostępnym pod adresem: http://wiki.ros.org/navigation/Tutorials/SendingSimpleGoals
**UWAGA! Należy pamiętać, że każde przesłane żądanie ruchu musi mieć inny identyfikator celu (wartość pola `goal_id` w wiadomości)!!**

## Zadanie 3 – analiza danych skanera laserowego

Wynikiem tego zadania jest publikowanie przetworzonych danych ze skanera laserowego. Dane ze skanera laserowego są publikowane na temacie `/scan`. Za pomocą narzędzi terminala poznanych na zajęciach należy sprawdzić strukturę wiadomości przesyłanych na tym temacie. W tej strukturze dla laboratorium istotne są cztery pola. Jedno zawiera listę odległości zwróconych przez promienie skanera, a pozostałe określają kąty pod którymi te promienie zostały wysłane (minimalny i maksymalny kąt odchylenia od czoła robota oraz droga kątowa między promieniami). Wizualizacja skanu laserowego jest pokazana poniżej:




Rysunek 3: Wizualizacja skanu laserowego

Węzeł utworzony w tym zadaniu na subskrybować temat `/scan` i publikować `/vacuum_sensors`. Na tym temacie mają byc publikowane wiadomości składające się z 'n' wartości typu `float`. Każda z wartości powinna być równa najniższej odległości pewnego wycinka obszaru badanego przez skaner. W ten sposób ze skanera powinniśmy otrzymać 'n' czujników sektorowych.

## Zadanie 4 – Odkurzanie zadanego pokoju

Należy napisać węzeł ROS, który zada cel ruchu do pokoju wskazanego w argumencie węzła oraz będzie zadawał kolejne cele, które poprowadzą robota przez zadany pokój tak, aby odkurzyć możliwie dokładnie i szybko ten pokój. Węzeł może bazować na danych dostarczonych przez węzeł napisany w zadaniu 3 oraz z gotowego systemu nawigacji (zadając mu odpowiednie cele analogicznie jak w zadaniu 2). System powinien reagować na odpowiedzi systemu nawigacji:

- subskrybować wiadomości na temacie /move_base/status
- wysyłać kolejny cel w razie gdyby serwer zwrócił status PREEMPTED(2), ABORTED(4), REJECTED(5).

```
uint8 PENDING         = 0   # The goal has yet to be processed by the action server
uint8 ACTIVE          = 1   # The goal is currently being processed by the action server
uint8 PREEMPTED       = 2   # The goal received a cancel request after it started executing
                            #   and has since completed its execution (Terminal State)
uint8 SUCCEEDED       = 3   # The goal was achieved successfully by the action server (Terminal State)
uint8 ABORTED         = 4   # The goal was aborted during execution by the action server due
                            #    to some failure (Terminal State)
uint8 REJECTED        = 5   # The goal was rejected by the action server without being processed,
                            #    because the goal was unattainable or invalid (Terminal State)
uint8 PREEMPTING      = 6   # The goal received a cancel request after it started executing
                            #    and has not yet completed execution
uint8 RECALLING       = 7   # The goal received a cancel request before it started executing,
                            #    but the action server has not yet confirmed that the goal is canceled
uint8 RECALLED        = 8   # The goal received a cancel request before it started executing
                            #    and was successfully cancelled (Terminal State)
uint8 LOST            = 9   # An action client can determine that a goal is LOST. This should not be
                            #    sent over the wire by an action server
```

Rysunek 4: Opis identyfikatorów stanów otrzymywanych w wiadomościach na temacie `/move_base/status`