Wstęp do Automatyki, Elektroniki i Telekomunikacji

Skrypt do laboratorium robotyka

Semestr: 2020 L

Autor: Wojciech Dudek

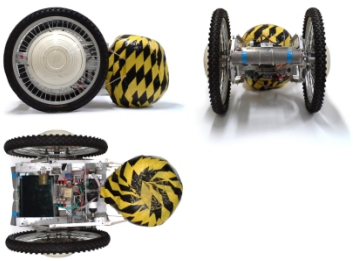
Spis treści

SEKCJA 1

Wstęp

Robot o napędzie różnicowym

Robot o napędzie różnicowym ma dwa niezależnie napędzane koła w jednej osi. Dla zachowania równowagi dodane jest jeszcze trzecie koło bierne (swobodnie obracająca się kula, lub koło wleczone), lub dwa bierne koła. Na rys. 1 przedstawiono przykłady robotów o napędzie różnicowym.



(a) Robot Ryś: https://robotyka.ia.pw.edu.pl

(b) Robot Pioneer 3DX: https://ztmir.meil.pw.edu.pl/

(c) Robot TIAGo: https://robotyka.ia.pw.edu.pl

Rysunek 1: Przykłady robotów o napędzie różnicowym

Podstawowe sterowanie ruchem robota różnicowego

Sterować ruchem kołowego robota mobilnego można na różne sposoby (np. prędkość obrotowa poszczególnych kół, lub prędkości liniowa i obrotowa bazy mobilnej). Na laboratorium będziemy wykorzystywać ten drugi sposób zadawania prędkości. Dyskretna ścieżka ruchu robota mobilnego składa się z listy pozycji (położenie i orientacja) robota, gdzie pierwszym elementem tej listy jest aktualna pozycja robota, a ostatnim pozycja docelowa. Algorytm sterowania, który będziemy wykorzystywać w ramach laboratorium jest jedną z najprostszych metod generowania trajektorii robota mobilnego na podstawie ścieżki, gdzie trajektoria to pozycje robota w czasie (ścieżka sparametryzowana czasem).
Działanie algorytmu składa się z powtarzających się dwóch etapów:

- ruch obrotowy – robot różnicowy jest orientowany w kierunku prowadzącym do kolejnego punktu ścieżki,

- ruch liniowy – robot porusza się do przodu (w kierunku osi X układu związanego z robotem), aż odległość między aktualnym położeniem robota i kolejnym punktem ścieżki zacznie rosnąć.

Powyższe etapy są powtarzane do wyczerpania punktów w ścieżce.

Programowa struktura ramowa ROS

Struktura ramowa ROS (*Robot Operating System*) jest najbardziej popularnym narzędziem do tworzenia systemów oprogramowania i sterowania robotów na świecie. Zarówno do celów przemysłowych, jak i do badań naukowych. Jest to nie tylko zbiór bibliotek zawierających implementacje algorytmów sterowania robotów i analizy danych sensorycznych, ale też zbiór programów służących do wizualizacji danych, administrowania systemem robota oraz debugowania.
Podstawowym elementem architektury systemów zbudowanych z wykorzystaniem ROS jest węzeł. Jest to program implementujący fragment funkcji całego systemu robota posiadający interfejsy wejściowe i wyjściowe. W ROS istnieją dwa typy interfejsów:

- **temat** (*topic*) – strumień ustrukturyzowanych danych przesyłany od jednego nadawcy (*publisher*) do jednego odbiorcy (*subscriber*) lub wielu odbiorców (*subscribers*).
- **usługa** (*service*) – implementuje model komunikacji serwer-klient, gdzie węzeł udostępniający usługę (*server*) oczekuje na żądanie od węzła klienta (*client*). Ten model komunikacji ma zdefiniowaną strukturę żądania oraz strukturę odpowiedzi. Klient wysyłając żądanie do serwera jest blokowany do czasu otrzymania odpowiedzi.

Narzędzia ROS

Aby skorzystać z narzędzi ROS, należy dowiązać ścieżki tychże narzędzi do ścieżek systemowych. Wykonuje się to poprzez komendę konsoli:
`source /opt/ros/melodic/setup.bash`
Każda nowo otwarta konsola (lub zakładka), w której będą wykorzystywane narzędzia ROS musi mieć dowiązane ścieżki do narzędzi ROS.

W trakcie laboratorium będą wykorzystane podstawowe narzędzia zaimplementowane w ROS:

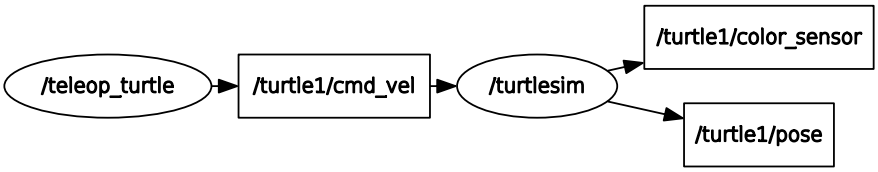
- **rostopic** – narzędzie konsoli, które umożliwia:
  - publikowanie wiadomości na zadany temat ROS (`rostopic pub <nazwa-tematu> <wiadomość>`),
  - ciągłe wypisywanie wiadomości publikowanych na dany temat (`rostopic echo <nazwa-tematu>`),
  - wypisanie aktualnie dostępnych tematów (`rostopic list`)
  - wypisanie informacji o danym temacie (`rostopic info <nazwa-tematu>`)
- **rosservice** – narzędzie konsoli, które umożliwia:
  - wywołanie usługi o podanej nazwie z podanym żądaniem (`rosservice call <nazwa-usługi> <żądanie>`),

– wypisanie aktualnie dostępnych usług
(`rosservice list`)

– wypisanie informacji o danej usłudze
(`rosservice info <nazwa-usługi>`)

Na rysunku 2 przedstawiono przykładową strukturę systemu sterowania w środowisku ROS. System ten składa się z dwóch węzłów *teleop_turtle* i *turtlesim*. Pierwszy z nich odczytuje stan przycisków klawiatury i publikuje odpowiednio ustrukturyzowane wiadomości zmiany prędkości na temacie */turtle1/cmd_vel*. Węzeł *turtlesim* w przykładowym systemie wyświetla żółwia emulującego robota i przemieszcza go na podstawie wiadomości subskrybowanych z tematu */turtle1/cmd_vel*. Węzeł *turtlesim* publikuje na dwóch tematach:

- */turtle1/color_sensor* – odczyt koloru znajdującego się pod żółwiem,
- */turtle1/pose* – odczyt aktualnej pozycji żółwia w globalnym układzie współrzędnych.



Rysunek 2: Przykładowa struktura programu sterującego robotem

ROS Python API

W trakcie laboratorium będą wykorzystywane tylko niektóre metody API struktury ramowej ROS. Głównie będą to metody wykorzystywane do komunikacji między węzłami ROS:

- tematy – link: samouczek
- usługi – link: samouczek

Tematy będą wykorzystywane do wysyłania sterowania oraz pobierania informacji o pozycji. Aby napisać węzeł publikujący/odbierający informacje na temacie (w przypadku laboratorium jest to nowa prędkość zadana, lub pozycja) należy zaimportować typ tejże wiadomości na początku skryptu:

```
from geometry_msgs.msg import Twist # typ wiadomości zawierającej prędkości zadane
from turtlesim.msg import Pose # typ wiadomości zawierającej pozycję
```

Wysyłanie danych na temacie jest realizowane przez moduł `rospy`, więc należy go najpierw zaimportować (`import rospy`). Następnie należy zaimportować odpowiedni typ wiadomości. Kolejno, trzeba utworzyć obiekt nadawcy:

```
pub = rospy.Publisher('<nazwa-tematu>', <typ-wiadomosci>, queue_size=10)
```

, zainicjować węzeł ROS:

```
rospy.init_node('<wybrana-nazwa-wezla>', anonymous=True)
```

, stworzyć obiekt wiadomości, która ma być wysłana:

```
msg = Twist()
```

zmienić wartości pól tego obiektu:

```
msg.linear.x = 1
msg.linear.y = 0
msg.linear.z = 0
msg.angular.x = 0
msg.angular.y = 0
msg.angular.z = 1
```

Wiadomość wysyłana jest przez metodę `publish` obiektu nadawcy:

```
pub.publish(msg)
```

Odbieranie danych z tematu jest realizowane przez moduł `rospy`, więc należy go najpierw zaimportować (`import rospy`). Następnie należy zaimportować odpowiedni typ wiadomości. Kolejno, trzeba zainicjować węzeł ROS:

```
rospy.init_node('<wybrana-nazwa-wezla>', anonymous=True)
```

zainicjować węzeł jako odbiorcę wiadomości na danym temacie oraz określenie funkcji, która ma się wykonać po otrzymaniu każdej kolejnej wiadomości:

```
rospy.Subscriber("<nazwa-tematu>", <typ-wiadomosci>, <nazwa-funkcji>)
```

Po przyjściu każdej wiadomości na danym temacie wykonana zostanie podana funkcja, którą w przypadku odbierania wiadomości pozycji trzeba samemu zdefiniować jako:

```
def <nazwa-funkcji>(pose):
        rospy.loginfo("Pozycja x: %8.2f", pose.x)
        rospy.loginfo("Pozycja y: %8.2f", pose.y)
        rospy.loginfo("Pozycja theta: %8.2f", pose.theta)
        ...
        # przetwarzanie odebranych danych
        ...
```

Turtlesim

Pakiet „turtlesim" należy do pakietów ROS. Służy on do nauki podstaw tworzenia systemów wykorzystujących strukturę ramową ROS. Pakiet „turtlesim" zawiera dwa węzły ROS (opisane w podsekcji narzędzia), a także wiele struktur wiadomości i usług oraz samouczki.

Turtlesim – narzędzia

turtlesim_node:
Ten węzeł tworzy okno wizualizacji obecnego stanu żółwi emulujących robota, którym można poruszać wykorzystując API struktury ROS. Sposób sterowania oraz API jest analogiczne do tego wykorzystywanego w sterowaniu robotami. Tematy ROS oraz usługi zostały opisane w sekcji *turtlesim_node – ROS API.*
turtle_teleop_key:
Ten węzeł służy do sterowania żółwiem wyświetlanym w oknie węzła turtlesim_node. Odczytywane są naciśnięcia klawiszy strzałek na klawiaturze i na tej podstawie są wysyłane zmiany prędkości żółwia. Węzeł „turtle_teleop_key" publikuje prędkości na temacie „/turtle1/cmd_vel", więc za jego pomocą można jedynie sterować pierwszym żółwiem stworzonym

w węźle „turtlesim_node”. Istnieje możliwość skonfigurowania tematu, na który są wysyłane prędkości poprzez dodanie odpowiedniego argumentu przy uruchamianiu tego węzła:
`rosrun turtlesim turtle_teleop_key turtle1/cmd_vel:=turtleX/cmd_vel`
, gdzie „turtleX” to nazwa żółwia, którym ma sterować węzeł „turtle_teleop_key”.

turtlesim_node – ROS API

Tematy:

- /turtleX/cmd_vel – zadawanie prędkości żółwia ”turtleX”,
- /turtleX/color_sensor – odczyt koloru pod żółwiem ”turtleX”,
- /turtleX/pose – odczyt pozycji żółwia ”turtleX”.

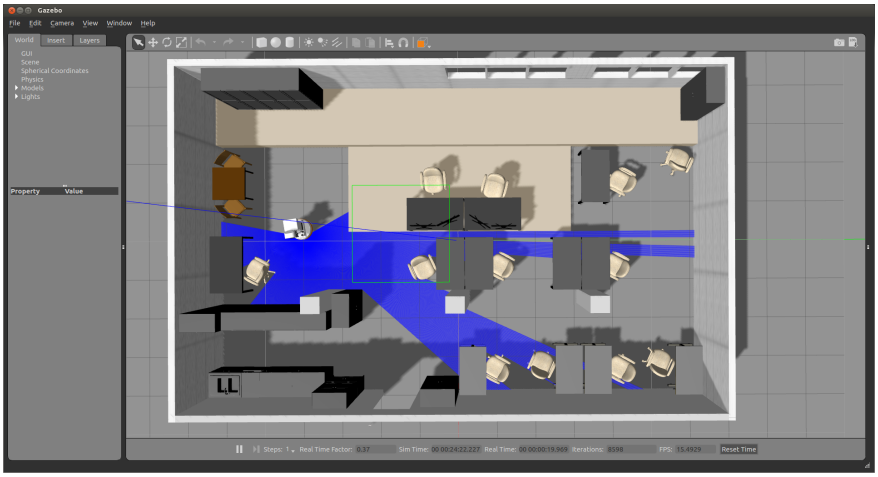
Usługi:

- /clear – wyczyszczenie nakreślonych linii, przykładowe żądanie: ”{}”
- /kill – wyłączenie zadanego żółwia, przykładowe żądanie: ”name: ’turtle1’ ”
- /reset – przywrócenie narzędzia `turtlesimn_node` do początkowej konfiguracji, przykładowe żądanie: ”{}”
- /spawn – stworzenie nowego żółwia o zadanej nazwie w zadanej pozycji, przykładowe żądanie:
  ”x: 5.544445
  y: 5.544445
  theta: 0.0
  name: ’turtle2’ ”
- /turtleX/set_pen – zmiana ustawień pisaka trzymanego przez żółwia ”turtleX”, przykładowe żądanie: ”{r: 0, g: 0, b: 0, width: 0, ’off’: 0}”

- /turtleX/teleport_absolute – teleportacja żółwia ”turtleX” do pozycji zadanej bezwzględnie, przykładowe żądanie:
  ”x: 0.0
  y: 0.0
  theta: 0.0”
- /turtleX/teleport_relative – teleportacja żółwia ”turtleX” do pozycji zadanej względnie, przykładowe żądanie: ”linear: 0.0 angular: 0.0”

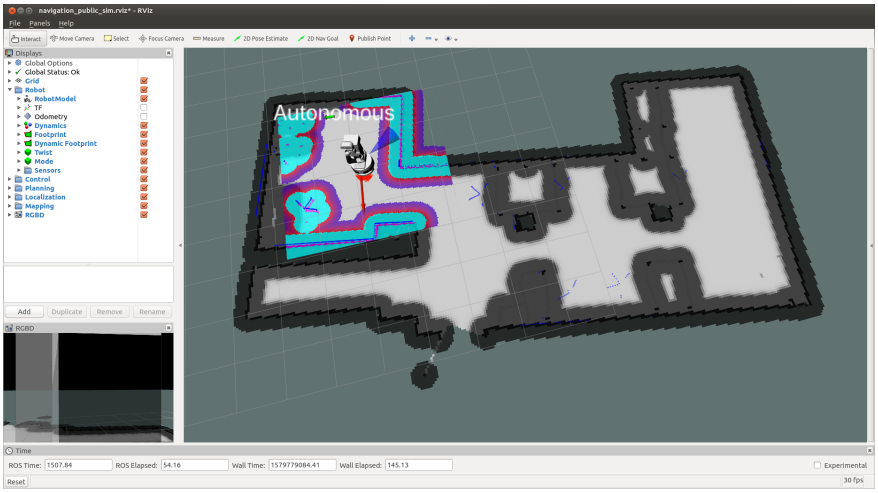
Symulator robota TIAGo

Robot TIAGo (fig. 1c) jest ma bazę mobilną z napędem różnicowym, korpus ruchomy w pionie oraz głowicę *Pan-Tilt* z kamerą RGB-D. Symulator tego robota zawiera dwa okna GUI. Pierwszym z nich jest okno programu Gazebo do symulacji robotów:



Rysunek 3: Widok okna programu Gazebo z symulacją robota TIAGo w sali 012

Drugim jest okno programu Rviz służącego do wizualizacji danych z systemu sterowania robotem:



Rysunek 4: Widok okna programu Rviz wizualizującego dane systemu sterowania

System sterowania robotem TIAGo ma wiele konfigurowalnych parametrów dotyczących planowania ruchu, wykonywania zadanego sterowania oraz percepcji środowiska. Użytkownik ma

możliwość sterowania bazą mobilną robota za pomocą prędkości liniowej i obrotowej, lub wskazując pozycję docelową w układzie mapy. W trakcie laboratorium będziemy sterować robota prędkościowo. Trzeba mieć jednak na uwadze, że system nawigacji robota omija przeszkody jedynie przy sterowaniu pozycyjnym.
Aby skorzystać z pakietów symulatora (w tym uruchomić go) należy dopisać ścieżki tych pakietów do ścieżek systemowych poprzez komendę terminala:
`source /opt/tiago/devel/setup.bash`
Po wykonaniu powyższej komendy można uruchomić symulator:
`roslaunch tiago_sim_integration tiago_navigation_public_012.launch`

SEKCJA 2

Opis zadań

Zadanie 1 – narzędzia Turtlesim

Za pomocą narzędzi ROS: *rostopic*, *rosservice*:

- zmiana koloru i szerokości pisaka,
- zmiana prędkości żółwia,
- nasłuchiwanie pozycji żółwia,
- przywrócenie początkowej konfiguracji narzędzia `turtlesim_node`.

Zadanie 2 – program sterujący żółwiem w Turtlesim

Napisać węzeł ROS do sterowania żółwiem Turtlesim. W programie jest zdefiniowana lista punktów (podanych we układzie współrzędnych globalnych), do których żółw ma się przemieścić. Węzeł subskrybuje temat z danymi o pozycji żółwia. W funkcji obsługującej otrzymaną wiadomość wyznacza nową prędkość zadaną (wyznaczoną według algorytmu z sekcji 1.2) i zapisuje ją w zmiennej globalnej. Program zawiera pętlę, która jest wykonywana z częstotliwością 10 Hz:

```
rate = rospy.Rate(10) # 10Hz
while not rospy.is_shutdown():
        pub.publish(new_vel) # wyslanie predkosci zadanej
        rate.sleep()
```

Zadanie 3 – sterowanie robotem TIAGo

Napisać skrypt w języku Python na podstawie zadania 2 i skryptu „zad3.py". Nowy skrypt ma oczekiwać na wiadomość określającą cel ruchu robota w układzie globalnym, na temacie „tiago_move" i wywołać funkcję „get_path" zdefiniowaną w skrypcie „zad3.py". Funkcja ta zapisuje zaplanowaną ścieżkę do zmiennej globalnej „new_path", która to zmienna jest dostępna w głównej funkcji skryptu.
Ponadto skrypt ma zawierać pętlę w której wyznaczane są kolejne prędkości zadane takie, aby robot poruszał się do **co dziesiątego** punktu ścieżki wyznaczonej przez funkcję „get_path" w analogiczny sposób, jak to miało miejsce w zadaniu 2.