Zadanie nawigacji robota mobinego PeopleBot z elementami systemu ROS*

Katarzyna Zadarnowska†
Laboratorium Robotyki
Wydział Elektroniki
Politechnika Wrocławska

1 Wstęp

Robot PeopleBot (rys.1) został wyprodukowany przez działającą w latach 1990-2018 firmę Adept MobileRobots głównie do celów badawczo - rozwojowych. Podstawowe parametry ro-

Rysunek 1: PeopleBot

bota:

- maksymalna prędkość – 0,8m/s, waga – 12kg, wysokość – 112cm, czas pracy – 8 godzin na w pełni naładowanych bateriach, czas ładowania baterii – 12 godzin;
- komputer PC104 z Linux Ubuntu;

*Ćwiczenie laboratoryjne przeznaczone do realizacji w ramach kursu Robotyka (3) – data ostatniej modyfikacji dokumentu 6 listopada 2023

†Katedra Cybernetyki i Robotyki

- dwa silniki napędzające koła robota mobilnego Pioneer 3-DX;
- 2 enkodery obliczające przebyty dystans;
- 10 zderzaków, informujących o tym, czy robot uderzył w przeszkodę;
- dalmierz laserowy z zakresem skanowania 180 stopni do tworzenia map pomieszczeń oraz sonary ultradźwiękowe do lokalizacji przeszkód;
- sensor wysyłający wiązkę podczerwieni służący do wykrywania płaskich powierzchni, takich jak blat stołu;
- mikrofon;
- joystick;
- kamery: PTZ (z funkcją obrotu, pozwala śledzić ruch) i stereowizyjna.

Z uwagi na duże wymiary robota, jego koła napędowe są wysunięte do przodu względem środka geometrycznego robota. Z tyłu jest zastosowane koło kastora, dzięki czemu zmaksymalizowany jest obszar stabilności statycznej robota. Para napędzanych kół tworzy układ wykonawczy. Układ sensoryczny jest znacznie bardziej zróżnicowany. Tworzą go enkodery inkrementalne przy kołach napędowych, zestaw 24-rech sonarów, skaner laserowy SICK LMS200 i 12 zderzaków (7 z przodu, 5 z tyłu). Enkodery mają rozdzielczość 1024 impulsy na obrót i mierzą obroty wału silników sprzęgniętych z kołami przekładnią o przełożeniu 22.3:1. Zasięg skanera wynosi około 80 [m], a rozdzielczość 0.5^o przy kącie skanowania 180^o. Takie parametry pozwalają na dobrą nawigację w dość dużych pomieszczeniach. Macierze sonarowe umieszczone wysoko (tuż pod komputerem pokładowym) i nisko (tuż pod płytą platofrmy mobilnej) wraz ze zderzakami, zlokalizowanymi nisko, tuż nad podłogą, są przeznaczone do detekcji przeszkód i wspomagają nawigację przy przeplanowywaniu ścieżek i omijaniu przeszkód. Rozlokowanie sonarów i zderzaków zapewnia, że robot praktycznie nie ma martwego pola widzenia na bliskie odległości.

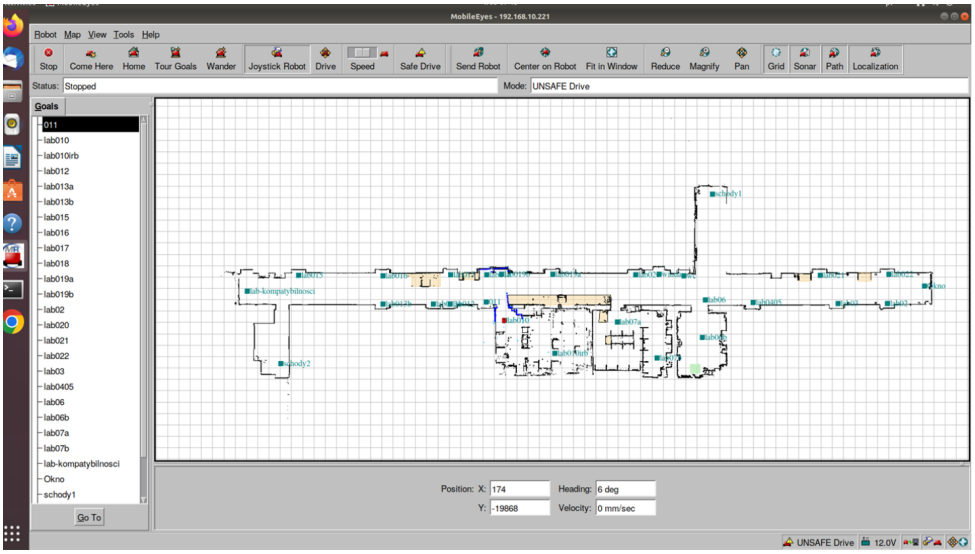
2 Oprogramowanie

PeopleBot był oferowany wraz z oprogramowaniem o charakterze API (ARIA, ARNL) oraz aplikacjami: GUI (MobileEyes), edytora map otoczenia (Mapper3) i symulatora (MobileSim). Udostępnione przez producenta specjalne oprogramowanie pozwala sterować robotem za pomocą zewnętrznego komputera oraz sieci bezprzewodowej.

1. MobileEyes - interfejs użytkownika, oryginalny GUI PeopleBota, umożliwiający m.in. tworzenie mapy otoczenia, lokalizację robota na mapie, sterowanie manualne, monitorowanie ruchu na mapie, zaznaczanie punktów docelowych na mapie na potrzeby jazdy autonomicznej. Dzięki MobileEyes można się połączyć z rzeczywistym robotem lub jego wersją wirtualną (w programie MobileSim), wystarczy podać adres IP stowarzyszonego z rzeczywistym bądź wirtualnym robotem (patrz rys. 2).

   Okno główne MobileEyes, patrząc od góry, zawiera: pasek rozwijanych menu (Robot, Map, View, Tools, Help), pasek komend, pasek pól tekstowych (Status, Mode), obszar z wgraną mapą otoczenia oraz obszar Goals (z predefiniowanymi pozycjami na mapie otoczenia, do których można skierować robota komendą **`Go To`**) oraz pasek komunikatów na samym dole. Pasek komend umożliwia sterowanie w kilku trybach: ręcznie przy

Rysunek 2: logowanie do robota



Rysunek 3: mapa

użyciu klawiatury (**`Drive`**) lub przez konsolę do sterowania (**`Joystick Robot`**). Jeśli jest wgrana mapa to można wskazać punkt docelowy na niej metodą Point & Click. Robot zacznie się tam przemieszczać w trybie autonomicznym. Można też zdefiniować miejsca docelowe i je następnie wybrać z listy w obszarze Goals, zatwierdzając przyciskiem **`Go To`**, żeby robot tam pojechał w trybie autonomicznym, z wykorzystaniem nawigacji i lokalizacji z ARNL. Można włączyć też tryb **`Wander`**. Wtedy robot porusza się losowo po dostępnym obszarze, zmieniając kierunek po wykryciu przeszkody przez sensory. Jest to tryb szczególnie przydatny przy tworzeniu mapy jakiegoś terenu. Zebrane dane trzeba później przetworzyć przy pomocy programu Mapper3. Wszystkie powyższe działania, związane z autonomicznym ruchem w oparciu o mapę, muszą być poprzedzone manualną lokalizacją na mapie w MobileEyes, po wydaniu instrukcji w pasku rozwijanych menu: **`Tools>Localization>Localize on map`**.
Mapa widoczna na rysunku 3 przedstawia cały korytarz na parterze budynku C3 na kampusie Politechniki Wrocławskiej wraz z pomieszczeniami laboratoryjnymi o numerach: 06, 07 i 010.

MobileEyes może być wykorzystywany do wstępnej lokalizacji PeopleBota na mapie,

monitorowania jego ruchu poza zasięgiem wzroku oraz jako narzędzie wspomagające przy wyznaczaniu współrzędnych pozycji robota wymaganych przez typy przesyłanych danych w ROSARIA.

2. MobileSim - środowisko symulacyjne, w którym można sterować wirtualnym robotem tak jak fizycznym, również przy użyciu interfejsu MobileEyes.

3. Mapper 3 - aplikacja do tworzenia/edytowania map otoczenia w oparciu o dane ze skanera laserowego. Do użytku wraz z oprogramowaniem ARIA, MobileSim i nawigacyjnym.

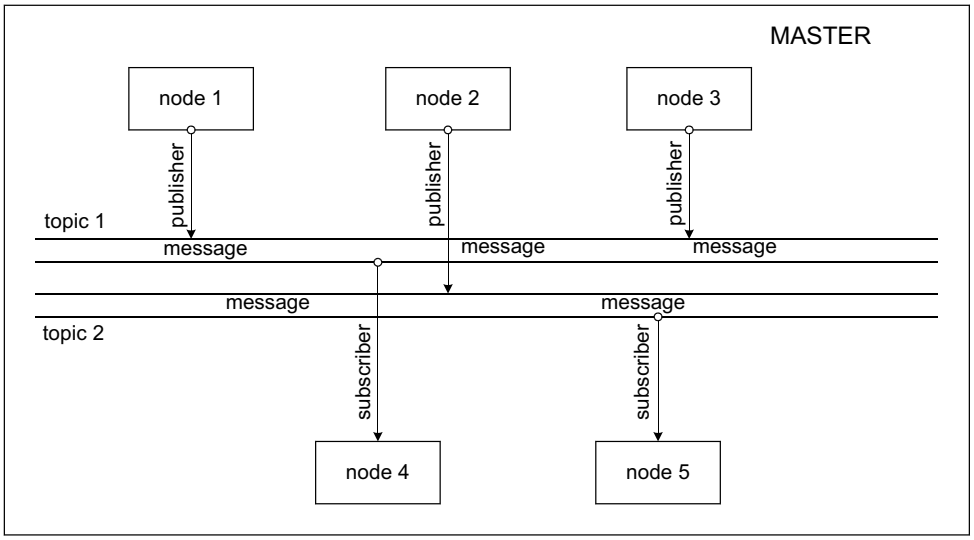
4. ARIA (Advanced Robot Interface for Applications) - API, biblioteka C++ będąca interfejsem programowym do sensorów i układów wykonawczych PeopleBota.

5. ARNL - biblioteka C++ będąca interfejsem przeznaczona do nawigacji i lokalizacji PeopleBota.

6. ACTS (ActivMedia Robotics' Color Tracking System) - aplikacja GUI współpracująca z kolorową kamerą i hardwarem framegrabbera, umożliwiająca identyfikację i śledzenie kolorowych obiektów zarejestrowanych w obrazie kamery (obecnie nieużywana).

3 Oprogramowanie ROS

ROS (Robot Operating System) to platforma programistyczna do tworzenia oprogramowania oraz sterowania robotów. Stanowi ona zbiór bibliotek pozwalających na sterowanie i symulację pracy robota. W szczególności, ROS zawiera podstawowe procesy systemowe obsługujące urządzenia sprzętowe robota, sterowanie niskopoziomowe, implementacje wykonywania typowych funkcji, komunikację międzywątkową oraz zarządzanie pakietami. Ideę środowiska ROS prezentuje rysunek 4.



Rysunek 4: Schemat projektu na platformie ROS

Oprogramowanie robotów tworzy się w postaci zbioru małych, działających współbieżnie i w dużej mierze niezależnych od siebie programów, nazywanych węzłami (ang. `nodes`). W celu ułatwienia procesu wymiany danych pomiędzy węzłami w systemie ROS jest wykorzystywany węzeł pełniący funkcję węzła nadrzędnego (ROS Master), który dba o poprawność komunikacji zachodzącej między pozostałymi węzłami systemu. Węzeł ten uruchamia się wydając

polecenie `roscore`. Węzły komunikują się między sobą poprzez wysyłanie wiadomości (ang. `messages`). Wiadomości zorganizowane są w tematach (ang. `topic`). Węzły, które chcą dzielić informacje (ang. `publishers`) publikują wiadomości w obrębie odpowiedniego tematu, natomiast węzły odbierające wiadomości, tzw. subskrybenci (ang. `subscribers`) subskrybują wiadomości. Węzeł nadrzędny powinien działać przez cały czas trwania pracy w systemie ROS, należy więc uruchomić go w oddzielnym terminalu (polecenie `roscore`) i pozostawić uruchomionym na czas pracy. Zatrzymanie węzła nadrzędnego następuje po wysyłaniu sygnał SIGINT (Ctrl-C).
Węzły* to wykonywalne instancje programów systemu ROS. Wyróżniamy następujący zestaw poleceń dotyczących węzłów:

- tworzenie węzła (uruchamianie programu systemu ROS)
  `rosrun` *`package-name executable-name`*
- tworzenie węzła wraz z nadaniem mu nazwy
  `rosrun` *`package-name executable-name`* `__name:=`*`node-name`*
- wyświetlenie listy uruchomionych węzłów
  `rosnode list`
- uzyskiwanie informacji na temat węzła
  `rosnode info` *`node-name`*
- zatrzymanie pracy (zabicie) węzła
  `rosnode kill` *`node-name`*
- usunięcie z listy zatrzymanego (zabitego) węzła
  `rosnode cleanup`

System ROS udostępnia następujący zbiór poleceń dotyczących tematów† i wiadomości przesyłanych w obrębie tematów:

- lista aktywnych tematów
  `rostopic list`
- wyświetlenie wiadomości przesyłanych aktualnie w obrębie danego tematu
  `rostopic echo` *`topic-name`*
- mierzenie częstotliwości publikowania wiadomości (wiadomość/sec)
  `rostopic hz` *`topic-name`*
- mierzenie przepustowości wiadomości (bajt/sec)
  `rostopic bw` *`topic-name`*
- uzyskiwanie informacji na temat danego tematu
  `rostopic info` *`topic-name`*
- uzyskiwanie szczegółowych informacji na temat przesyłanych wiadomości
  `rosmsg show` *`message-type-name`*
- publikowanie wiadomości
  `rostopic pub -r` *`rate-in-hz topic-name message-type message-content`*

*http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes
†http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics

W związku z zawieszeniem działalności firmy, software robota nie jest rozwijany. Niemniej, można wykorzystać biblioteki ROSARIA i ROSARNL, dzięki którym można kontrolować parametry ruchu robota, odczytywać dane z sensorów, zarządzać urządzeniami takimi jak kamery.

ROSARIA

ROSARIA to ROS-owe API dla robotów wytworzonych przez Adept MobileRobots, wytworzone na bazie biblioteki ARIA. Umożliwia programowy dostęp od wszystkich układów wykonawczych i sensorycznych robota.

ROSARNL

Środowisko ROSARNL zostało rozwinięte przez firmę Adept MobileRobots na bazie biblioteki ARNL. Pozwala ono nawigować robotem w oparciu o mapę otoczenia za pomocą interfejsu tematu (ang. `topic`) `move base`. Jest to standardowy interfejs ROS-owy do nawigacji i automatycznego lokalizowania robota.

ROSARNL dostarcza węzeł `rosarnl_node`. Lista tematów dla tego węzła jest następująca:

```
robotyka@urs: rostopic list
/peoplebot/rosarnl_node/amcl_pose
/peoplebot/rosarnl_node/arnl_path_state
/peoplebot/rosarnl_node/arnl_server_mode
/peoplebot/rosarnl_node/arnl_server_status
/peoplebot/rosarnl_node/battery_status
/peoplebot/rosarnl_node/bumper_state
/peoplebot/rosarnl_node/current_goal
/peoplebot/rosarnl_node/goalname
/peoplebot/rosarnl_node/initialpose
/peoplebot/rosarnl_node/jog_position/cancel
/peoplebot/rosarnl_node/jog_position/feedback
/peoplebot/rosarnl_node/jog_position/goal
/peoplebot/rosarnl_node/jog_position/result
/peoplebot/rosarnl_node/jog_position/status
/peoplebot/rosarnl_node/jog_position_simple/goal
/peoplebot/rosarnl_node/lms2xx_1_laserscan
/peoplebot/rosarnl_node/lms2xx_1_pointcloud
/peoplebot/rosarnl_node/motors_state
/peoplebot/rosarnl_node/move_base/cancel
/peoplebot/rosarnl_node/move_base/feedback
/peoplebot/rosarnl_node/move_base/goal
/peoplebot/rosarnl_node/move_base/result
/peoplebot/rosarnl_node/move_base/status
/peoplebot/rosarnl_node/move_base_simple/goal
```

Lista serwisów stowarzyszonych z `rosarnl_node` jest następująca:

```
robotyka@urs: rosservice list
/peoplebot/rosarnl_node/disable_motors
/peoplebot/rosarnl_node/dock
/peoplebot/rosarnl_node/enable_motors
/peoplebot/rosarnl_node/get_loggers
/peoplebot/rosarnl_node/global_localization
/peoplebot/rosarnl_node/load_map_file
/peoplebot/rosarnl_node/make_plan
```

```
/peoplebot/rosarnl_node/set_logger_level
/peoplebot/rosarnl_node/stop
/peoplebot/rosarnl_node/wander
```

Lista tematów dla węzła `RosAria` jest następująca:

```
robotyka@urs: rostopic list
/RosAria/battery_recharge_state
/RosAria/battery_state_of_charge
/RosAria/battery_voltage
/RosAria/bumper_state
/RosAria/cmd_vel
/RosAria/motors_state
/RosAria/parameter_descriptions
/RosAria/parameter_updates
/RosAria/pose
/RosAria/sonar
/RosAria/sonar_pointcloud2
/rosout
/rosout_agg
/tf
```

Najbardziej popularne tematy węzła `rosarnl_node`:

- `/peoplebot/rosarnl_node/amcl_pose` - topic typu publisher, który zwraca aktualną pozycję na mapie.
- `/peoplebot/rosarnl_node/move_base_simple/goal` - topic typu subscriber, do którego można przekazać współrzędne docelowego położenia robota. Typ przesyłanych wiadomości to geometry `msgs/PoseStamped`.
- `/peoplebot/rosarnl_node/arnl_path_state` - topic typu publisher, który zwraca aktualny stan wyznaczonej ścieżki, na podstawię czego można określić czy robot osiągnął wyznaczony cel czy też nie,
- `/peoplebot/rosarnl_node/bumper_state` - topic typu publisher, który zwraca aktualną wartość stanu czujników zamontowanych na zderzakach (włączony/wyłączony). Za każdym razem, gdy któryś czujnik zmieni swój stan, wysyłana jest informacja na ten topic.
- `/peoplebot/rosarnl_node/stop` - topic serwisowy, jego wywołanie spowoduje zatrzymanie się robota.

Przez w/w tematy `rosanl_node` przesyłane są wiadomości, których typy danych mają złożone struktury. W celu podejrzenia tych struktur i informacji w nich zadanych wystarczy użyć polecenia `rostopic echo nazwa_tematu` np.
`rostopic echo /peoplebot/rosarnl_node/amcl_pose` – informujący o aktualnej pozycji na mapie;
`rostopic echo /peoplebot/rosarnl_node/bumper_state` - informujący o stanie zderzaków.

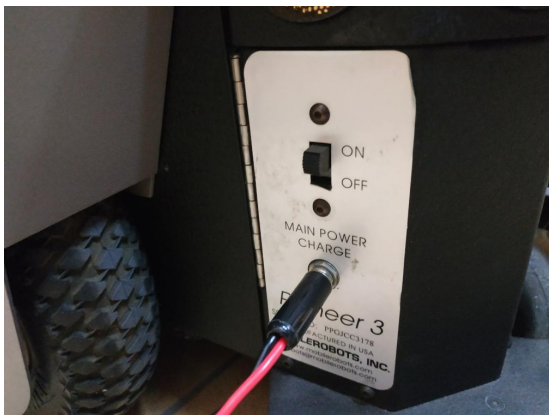
4 Nawigowanie robotem PeopleBot: sterowanie z poziomu aplikacji Mobile Eyes

Program Mapper odtwarza przejazd robota wraz z odczytami z sensorów - dzięki temu tworzona jest mapa pomieszczenia. Po wykonaniu mapy możliwe jest sterowanie robotem (np. przemieszczenie robota w określone miejsce z jednoczesnym omijaniem przeszkód). Dzięki odczytom z sonarów, utworzonej mapie oraz odpowiedniemu oprogramowaniu możemy obserwować ruch robota na ekranie komputera. W oprogramowaniu zamplementowano odpowiednie algorytmy planowania ruchu robota z omijaniem przeszkód. Robotem można sterować za pomocą klawiszy kierunkowych (**`Drive`**) lub joysticka (**`Joystick Robot`**) za pomocą dołączonego oprogramowania.
Aby móc korzystać z oprogramowania MobileEyes należy wykonać następujące czynności:

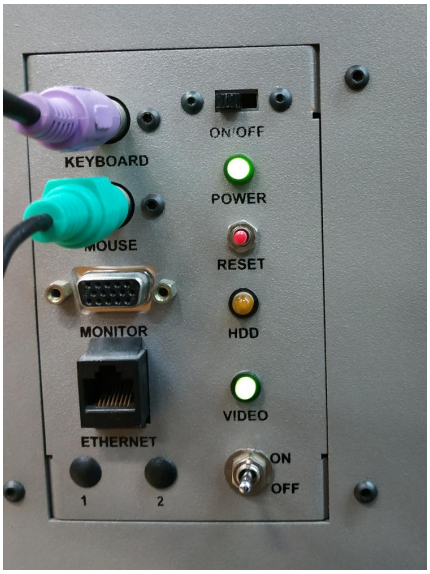
1. W terminalu **`LXTerminal`** komputera pokładowego robota (rys. 7a)) uruchom umieszczony w katalogu `/run_scripts/peoplebot_scripts` skrypt **`./rosarnl_node_start_script.sh`** (rys. 7b))

   Alternatywnie, można się zalogować z komputera stacjonarnego `urs` na komputer pokładowy robota:
   `robotyka@urs: ssh -l lirec -X 192.168.10.221`
   i dalej z katalogu `/run_scripts/peoplebot_scripts` uruchomić skrypt
   `./rosarnl_node_start_script.sh`

   Uwaga: W obu przypadkach na serwerze `rab` musi być uruchomioy `roscore`.



a)

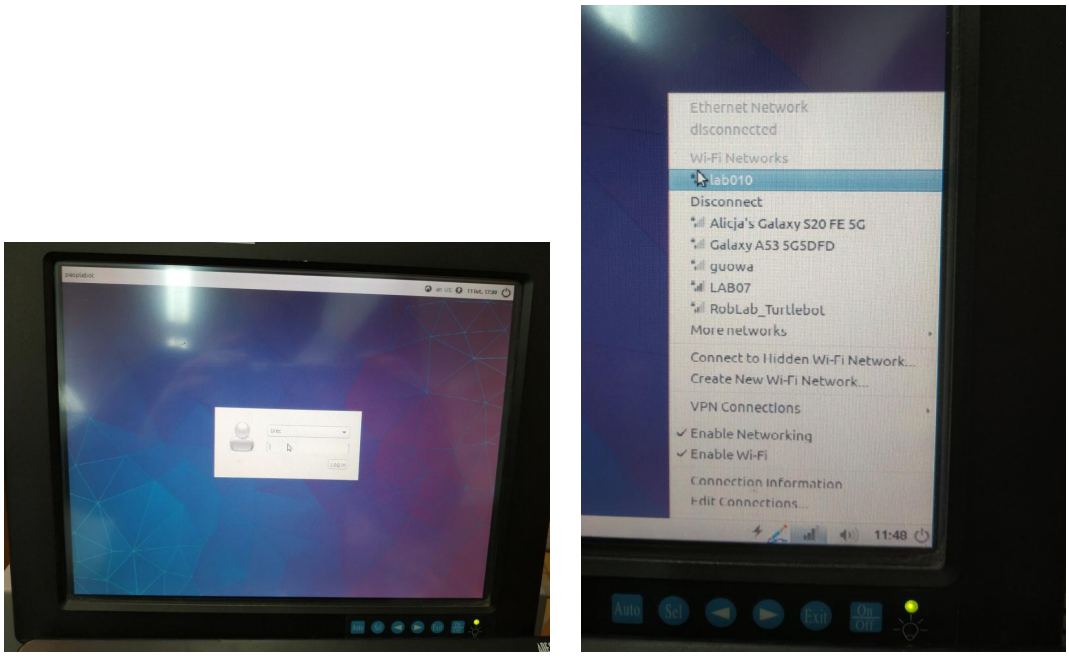


b)

Rysunek 5: Włączanie zasilania w kolejności: a) platforma mobilna, b) komputer pokładowy – góra oraz monitor – dół

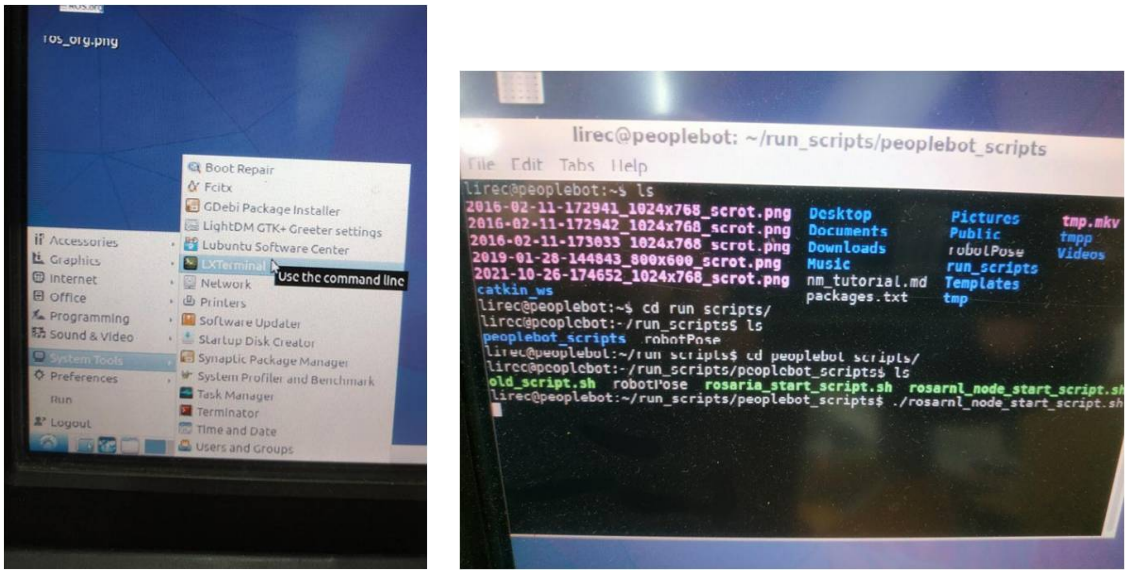
2. Uruchom program MobileEyes na komputerze stacjonarnym (8; login: **lirec**, hasło: (zapytaj prowadzącego), Robot Servers: 192.168.10.221).

3. W ten sposób otrzymasz okno główne programu MobileEyes wraz z mapą otoczenia (9)



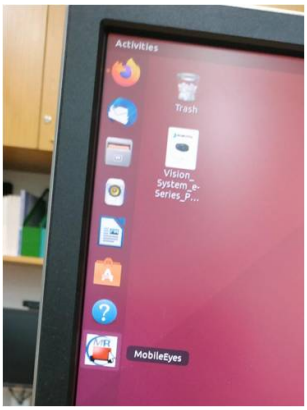
a) b)

Rysunek 6: Logowanie na komputerze pokładowym a) okno logowania, b) sprawdzenie połączenia z siecią



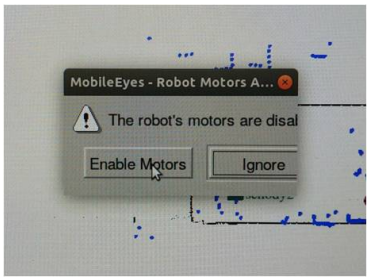
a) LXterminal b) Uruchomienie skryptu `./rosarnl_node_start_script.sh`

Rysunek 7: Uruchamianie ROSARNL na komputerze pokładowym robota



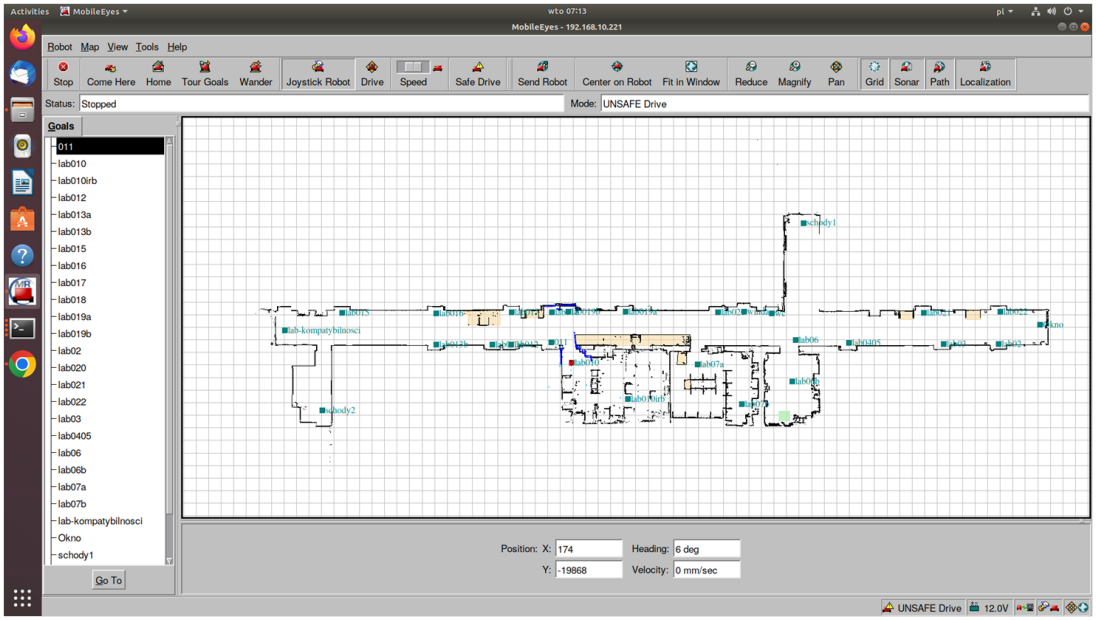
a) Wywołanie MobileEyes

b) Logowanie do Mobile Eyes



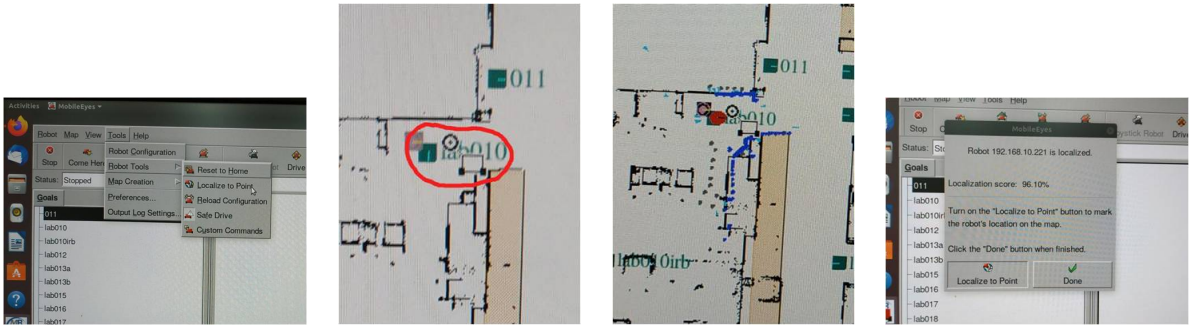
c) Aktywowanie sterowników

Rysunek 8: Uruchamianie program MobileEyes



Rysunek 9: Okno główne programu MobileEyes wraz z mapą otoczenia

4. Dokonaj lokalizacji robota na mapie. W tym celu korzystając z joysticka umieść robota naprzeciw otwartych drzwi sali 010 (jak na rys. 10b)), wybierz `Tools` → `Robot Tools` → `Localize to point` (10a)), zaznacz prawym przyciskiem myszy faktyczne położenie i orientację (przeciągając myszką z wciśniętym prawym przyciskiem myszy w kierunku zadanej orientacji) fizycznego robota na mapie (10b)), poczekaj aż robot zlokalizuje się na mapie (10c)) i kliknij `Done` (10d)).



a) b) c) d)

Rysunek 10: Lokalizacja robota na mapie

5. Przykładowe polecenia oprogramowania `MobileEyes`:

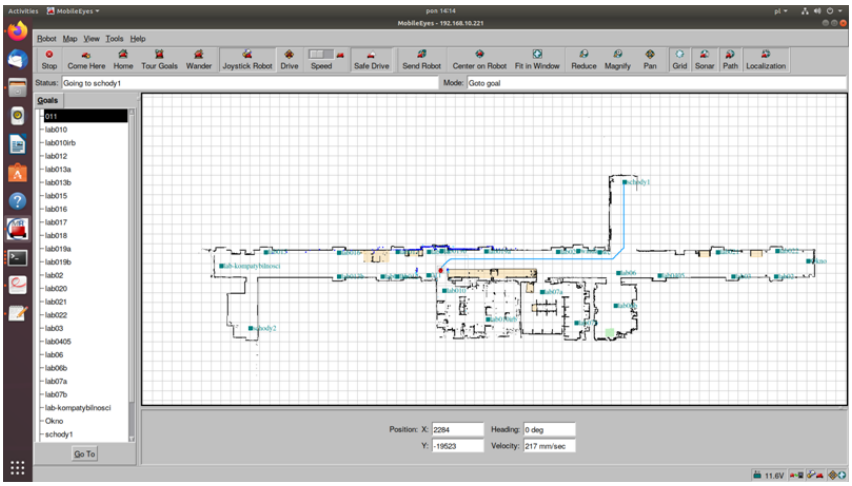
   - **`Safe Drive`** – włączenie/wyłączenie omijania przeszkód;
   - **`Joystick Robot`** – sterowanie robotem za pomocą joystick'a;
   - **`Drive`** – sterowanie robotem za pomocą klawiszy strzałek;
   - **`Send Robot`** – zadawanie pozycji i orientacji robota; oeirntację zadajemy po zadaniu położenia, poprzez przesunięcie myszką w zadanym kierunku;
   - **`Wander`** – wałęsanie się robota;
   - **`Tour goals`** – przejazd przez wszystkie punkty zadane na mapie.

6. Nawigowanie robotem do pozycji docelowej odbywa się poprzez kliknięcie na predefiniowany na mapie punkt (np. schody) co powoduje, że zostaje uruchomiony proces planowania ruchu od pozycji lokalizacji robota (czerwony prostokąt) do pozycji docelowej (schody) i następnie ruchu robota do pozycji docelowej (11).

7. W trakcie realizacji ruchu robota, na mapie kolorem jasnoniebieskim będą oznaczone obiekty rejestrowane przez sonary, natomiast kolorem ciemnoniebieskim - obiekty rejestrowane przez lidar.

8. W dolnym prawym, rogu aplikacji MobileEyes znajduje się informacja dotycząca poziomu naładowania baterii robota pokładowego. Gdy poziom naładowania spadnie poniżej 12V, pojawi się odpowiedni komunikat.

9. Przycisk **`Quit`** umożliwia zakończenie pracy z `MobileEyes`



Rysunek 11:

5 Nawigowanie rzeczywistym robotem PeopleBot

5.1 Sterowanie z linii komend terminala – publikowanie położeń

Zadanie polega na przemieszczeniu pomiędzy pozycją początkową i końcową. Zadanie można zrealizować wydając odpowiednie komendy w terminalu komputera `urs` (lub komputera pokładowego PeopleBota). Wcześniej należy wykonać dwie operacje. Najpierw nalezy zlokalizować PeopleBota na mapie przy użyciu programu MobleEyes. Następnie należy wyznaczyć współrzędne docelowych pozycji robota (początowej i końcowej). W tym celu, za pomocą MobileEyes należy skierować robota do pozycji początkowej i za pomocą polecenia **`rostopic echo /peoplebot/rosarnl_node/amcl_pose`** odczytać współrzędne pozycji. Następnie to samo należy wykonać dla pozycji docelowej. Uzyskamy współrzędne w formacie:
Pozycja początkowa
położenie x: 3.35, y: -25, z: 0.00 ,
orientacja x: 0.0, y: 0.0, z: 0.77, w: 0.45
Pozycja docelowa
położenie x:2.00, y:s-37.58, z: 0.00 ,
orientacja x: 0.0, y: 0.0, z: 0.99, w: 0.45
Teraz można publikować wiadomość na topic /peoplebot/rosarnl_node/move_base_simple/goal wydając polecenie z terminala postaci:

```
rostopic pub -1 /peoplebot/rosarnl_node/move_base_simple/goal
geometry_msg/PoseStamped '{header: {stamp:now, frame_id:"map"},
pose: {position: {x:2.00, y:s-37.58, z:0.00},
orientation: {x:0.0, y:0.0, z:0.99, w:0.45}}}'
```

Uwaga! Należy pamiętać aby czas na komputerze i na komputerze pokładowym robota był zsynchronizowany. W przeciwnym razie powyższa komenda spotka się z brakiem reakcji robota.

5.2 Sterowanie z poziomu skryptu w Pythonie – publikowanie położeń

Zadanie definiujemy jak w poprzednim podrozdziale. Można je zrealizować przy użyciu interfejsu topików `move_base` (http://wiki.ros.org/move_base). Do tego celu służy oferowana przez środowisko `ROSARNL` klasa `SimpleActionClient`.

Efektem realizacji zadania z tego rozdziału jest skrypt `move_peoplebot2.py`, (rysunek 12) będący częścią pakietu `ur3pbcooper` na komputerach `rab` i `urs`.Pakiet `ur3pbcooper` znajduje się na koncie `robotyka`, w katalogu `/home/robotyka/catkin_ws/src`.
Plik `move_powerbot2.py` jest demonstratorem ilustrującym nawigację pomiędzy predefiniowanymi pozycjami A i B w formie ruchu wahadłowego. Demonstrator może być użyty do tworzenia innych skryptów definiujących trasę robota. Tworzenie nowych tras oznacza wprowadzenie sekwencji docelowych pozycji robota. Jednym ze sposobów jest przejazd nowej trasy przy użyciu joysticka i odczytanie z topiku: `/peoplebot/rosarnl_node/amcl_pose` wartości pozycji robota w miejscach docelowych.

Pierwsza linia skryptu (rysunek 12) umożliwia jego uruchomienie z poziomu terminala, jeśli wcześniej nadamy mu odpowiednie uprawnienia. Dalej do skryptu importowane są trzy biblioteki:
`rospy` – do korzystania ze środowiska ROS w języku Python,
`actionlib` – w której znajduję się klasa `SimpleActionClient`, i
`tf.transformations` – do wyrażenia obrotu robota przy użyciu kwaternionu.

Z biblioteki `geometry_msgs.msg` pobieramy klasy `Pose`, `Quaternion`, potrzbne do wskazania pozycji docelowej dla PeopleBota. Z kolei, z biblioteki `move_base_msgs.msg` importujemy klasy `MoveBaseAction`, `MoveBaseGoal`, potrzebne do obsługi klasy `SimpleActionClient`.

W kolejnych liniach skryptu inicjujemy węzeł `ROS`-owy, w którym będzie wykonywany cały skrypt, a następnie inicjujemy klasę `SimpleActionClient` połączoną z PeopleBotem.

```
rospy.init_node('move_peoplebot',anonymous=True)
commander=actionlib.SimpleActionClient('rosarnl_node/move_base',
MoveBaseAction)
```

Następnie definiujemy dwa położenia docelowe dla PeopleBota

```
pose=Pose()
pose.position.x = 2.53
pose.position.y = -18.837
pose.position.z = 0.0

pose_first = Pose ()
pose_first . position . x = 2.000
pose_first . position . y = -37.58437
pose_first . position . z = 0.0.
```

Dodatkowo wyznaczamy orientację w jakiej ma się zatrzymać robot.

```
q = tf.transformations.quaternion_from_euler(0 , 0 , 1.5707)
pose.orientation = Quaternion (*q)
```

W pętli for wykonujemy sześć ruchów, zależnie od numeru iteracji pętli odpowiednio przypisujemy miejsce docelowe. Następnie wprawiamy robota w ruch i czekamy, aż program się zakończy.

```
#! /usr/bin/env python

import rospy
import actionlib
import tf.transformations

from geometry_msgs.msg import Pose, Quaternion
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal


rospy.init_node('move_powerbot', anonymous=True)
commander = actionlib.SimpleActionClient('rosarnl_node/move_base',
MoveBaseAction)

print 'Wait for server'
commander.wait_for_server()

pose = Pose()
pose.position.x = 2.53
pose.position.y = -18.837
pose.position.z = 0.0

pose_first = Pose()
pose_first.position.x = 2.000
pose_first.position.y = -37.58437
pose_first.position.z = 0.0

for x in range(6):
    q = tf.transformations.quaternion_from_euler(0, 0, 1.5707)
    pose.orientation = Quaternion(*q)

    goal = MoveBaseGoal()
    if x%2 == 0:
        goal.target_pose.pose = pose_first
        goal.target_pose.header.frame_id = 'map'
        goal.target_pose.header.stamp = rospy.Time.now()

    else:
        goal.target_pose.pose = pose
        goal.target_pose.header.frame_id = 'map'
        goal.target_pose.header.stamp = rospy.Time.now()
    print 'PowerBot go to the goal: %s' % goal
    commander.send_goal(goal)
    print 'Wait for result'
    commander.wait_for_result()

    print 'Commander state %s' % commander.get_state()
    print 'Goal status %s' % commander.get_goal_status_text()
```

Rysunek 12: Skrypt publikujący położenia dla robota. Autor: Radosław Pawłowski

Ostatnim elementem skryptu jest wyświetlenie komunikatu o rezultacie ruchu robota.

Sposób użycia
W celu wykonania skryptu należy:

1. Uruchomić robota.
2. Z katalogu `/run_scripts/peoplebot_scripts` robota pokładowego uruchomić skrypt **`./rosarnl_node_start_script.sh`** i połączyć się z robotem za pomocą programu MobileEyes.
3. Zlokalizować robota w programie MobileEyes.
4. Na komputerze urs przejść do kartoteki `/home/robotyka/catkin_ws/src/robotyka_3/scripts`.
5. Uruchomić skrypt na komputerze **urs** za pomocą polecenia

   `ROS_NAMESPACE=peoplebot rosrun ur3pbcooper move_powerbot2.py`

5.3 Sterowanie z poziomu klawiatury – publikowanie prędkości

Prędkości dla robota można publikować z klawiatury. Aby uruchomić sterowanie kinematyczne przy użyciu klawiatury z wykorzystaniem RosAria i `/cmd_vel` należy:

1. Na komputerze pokładowym robota uruchomić RosArię (w tym celu z katalogu `run_scripts/peoplebot_scripts/` wywołać polecenie **`./rosaria_start_script.sh`**)
2. Na komputerze **urs** przejść do kartoteki `/home/robotyka/catkin_ws/src/robotyka_3/scripts`.
3. Na komputerze **urs** uruchomić węzeł:
   `ROS_NAMESPACE=RosAria`
   `rosrun teleop_twist_keyboard teleop_twist_keyboard.py`

W efekcie powinien ukazać się prosty komunikat informujący o tym jak poruszać robotem przy użyciu klawiszy (patrz rys. 13). (http://wiki.ros.org/teleop_twist_keyboard)

5.4 Sterowanie z poziomu skryptu w Pythonie – publikowanie prędkości

Program `cmd_vel_peoplebot.py` (patrz rysunek 14) tworzy węzeł, który generuje oraz publikuje prędkości dla robota. Prędkości są typu `Twist`. Jako, że typ `Twist` zdefiniowano w pakiecie `geometry_msgs`, konieczne jest pojawienie się w programie dyrektywy
`from geometry_msgs.msg import Twist`
Dodatkowo, w celach bezpieczeństwa, program uwzględnia informacje z sonarów, a zatem z biblioteki `sensor_msgs.msg` importujemy klasę `PointCloud2` potrzebną do obróbki sygnałów z sonarów
`from sensor_msgs.msg import PointCloud2`
Do skryptu importujemy bibliotekę `ros_numpy`, która posłuży do macierzowego zapisu informacji z czujników.

```
Reading from the keyboard  and Publishing to Twist!
---------------------------
Moving around:
   u    i    o
   j    k    l
   m    ,    .

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
anything else : stop

CTRL-C to quit
```

Rysunek 13: Klawisze funkcyjne w `teleop_twist_keyboard`

Funkcja `callback` pobiera dane z sonarów zapisane w postaci chmury punktów i przekształca je do postaci trójwymiarowej tablicy zawierającej współrzędne punktów w przestrzeni 3D. `ros_numpy.point_cloud2` jest modułem biblioteki ROS zawierającym funkcje do pracy z chmurami punktów. `pointcloud2_to_xyz_array(data)` to funkcja modułu, która służy do przekształcania danych z chmury punktów na tablicę `NumPy` zawierającą współrzędne punktów. W pętli `for` sprawdzane są wartości tablicy mające sens odległości sonarów od przeszkody. Jeśli odległość jest mniejsza niż zadana wartość ustwiana jest zmienna globalna `STOP`, natomiast wartość punktu zapisywana jest do logów ROS, co może pomóc w monitorowaniu działania programu lub w celach diagnostycznych.

Funkcja `emergency_stop` jest odpowiedzialna za sprawdzenie, czy flaga `STOP` została ustawiona na `True`, co wskazuje na napotkanie przeszkody. funkcja `rospy.logerr` służy do zapisywania błędów (logów o wysokim priorytecie) do systemu logów ROS. Log ten zawiera komunikat informujący o napotkaniu przeszkody i instrukcję, jak przejść w tryb sterowania ręcznego.

Funkcja `move` ma na celu poruszanie robotem z zadaną prędkością liniową (`lin_vel`) i prędkością kątową (`ang_vel`) przez określony czas (`time`). Typ `rospy.Rate` pozwala stworzyć obiekt, który kontroluje częstotliwość pętli (ilość iteracji na sekundę). W tym przypadku, pętla będzie działać z częstotliwością $100Hz$, co oznacza, że każda iteracja trwa $1/100$ sekundy. Typ wiadomości `Twist` pakietu `geometry_msgs` składa się z dwóch pól opisujących prędkość liniową (m/sec) oraz prędkość kątową (rad/sec) robota.

```
geometry_msgs/Vector3 linear
float64 x
float64 y
float64 z
geometry_msgs/Vector3 angular
float64 x
float64 y
```

```
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Twist
from sensor_msgs.msg import PointCloud2
import ros_numpy

STOP = False

def callback(data):
    global STOP
    xyz_array = ros_numpy.point_cloud2.pointcloud2_to_xyz_array(data)
    for point in xyz_array:
        p = abs(point[1])
        if p < 0.09 and p != 0:
            STOP = True
            rospy.loginfo(p)


def emergency_stop():
    global STOP
    if STOP:
        rospy.logerr('Napodkano przeszkode.\n Aby wejsc w tryb steroawnia
           recznego wywolaj: \n ROS_NAMESPACE=RosAria rosrun
           teleop_twist_keyboard teleop_twist_keyboard.py')
        return True
    else:
        return False

def move(lin_vel,ang_vel, time):
    r = rospy.Rate(100)
    vel_msg = Twist()
    vel_msg.linear.x = lin_vel
    vel_msg.linear.y = 0
    vel_msg.linear.z = 0
    vel_msg.angular.x = 0
    vel_msg.angular.y = 0
    vel_msg.angular.z = ang_vel
    for i in range(time * 100):
        if emergency_stop():
            break
        vel_pub.publish(vel_msg)
        r.sleep()
    vel_msg.linear.x = 0
    vel_msg.angular.z = 0
    vel_pub.publish(vel_msg)

SIMULATION = True

rospy.init_node("poeplebot_cmd_vel", anonymous = True)

if SIMULATION:

    vel_pub = rospy.Publisher("/cmd_vel", Twist, queue_size = 10)
else:
    vel_pub = rospy.Publisher("RosAria/cmd_vel/", Twist, queue_size = 10)
    rospy.Subscriber("/RosAria/sonar_pointcloud2", PointCloud2, callback)


if not rospy.is_shutdown():
    move(lin_vel=0.1,ang_vel=0.1,time=10) # 0.1 m/s i jednoczesnie 0.1 rad/s
                przez 10s; predkosc dodatnia - jazda w przod, ujemna w tyl
```

Rysunek 14: Skrypt publikujący prędkości dla robota. Autor: Jakub Kusz

```
float64 z
```

Zatem obiekt vel_msg będzie przechowywać informacje o prędkości postępowej robota wzdłuż osi `x` (`linear.x`) i prędkości zmiany orientacji robota wokół osi `z` (`angular.z`). Pozostałe składowe przyjmują zerowe wartości. W każdej iteracji pętli `for` jest sprawdzana funkcja `emergency_stop()`. Jeśli ta funkcja zwróci wartość True, pętla zostanie przerwana, co oznacza zatrzymanie ruchu robota z powodu napotkania przeszkody. W przeciwnym razie, w każdej iteracji pętli przy użyciu metody textttpublish publikowana jest prędkość robota `vel_pub.publish(vel_msg)`. Metoda dodaje wiadomość do kolejki wiadomości, skąd wiadomość możliwie najszybciej zostaje wysyłana do wszystkich subskrybentów odpowiadających danemu tematowi. Opóźnienie `r.sleep()` zapewnia utrzymanie stałej częstotliwości pętli. Na końcu pętli prędkość robota jest ustawiana na zero i ponownie publikowana, aby zatrzymać robota.

Dalej mamy główną część programu. Metoda `init_node` inicjalizuje węzeł ROS o nazwie `peoplebot_cmd_vel` i dzięki opcji `anonymous=True` zapewnia unikalność jego nazwy (na wypadek gdyby istniało wiele węzłów o podobnej nazwie). Zmienna `SIMULATION` określa czy będziemy działać w trybie symulacji (sterować wirtualnym robotem) czy na fizycznym sprzęcie. W bloku `if` tworzony jest obiekt o nazwie `vel_pub` służący do publikownia komunikatów typu `Twist`. W trybie symulacji obiekt publikuje do tematu `cmd_vel`, natomiast w trybie pracy na robocie obiekt publikuje do tematu `/RosAria/cmd_vel` z rozmiarem kolejki równym 10. W przypadku trybu pracy na robocie tworzony jest obiekt `rospy.Subscriber`, który subskrybuje temat `/RosAria/sonar_pointcloud2` i przypisuje funkcję callback jako funkcję obsługi zdarzeń dla odebranych danych. `if not rospy.is_shutdown()::` to polecenie, w którym sprawdzamy, czy węzeł nie został zamknięty. Jeśli węzeł jest aktywny, program przechodzi do następnej linii, w której następuje wywołanie funkcji `move` z parametrami prędkości odpowiedzialnymi za sterowanie ruchem robota przez określony czas.

Sposób użycia
W celu wykonania skryptu należy:

1. Uruchomić robota.

2. Na komputerze pokładowym robota uruchomić RosArię (w tym celu z katalogu `run_scripts/peoplebot_scripts/` wywołać polecenie **`./rosaria_start_script.sh`**.

3. Na komputerze **urs** przejść do kartoteki `/home/robotyka/catkin_ws/src/robotyka_3/scripts`.

4. Na komputerze **urs** uruchomić węzeł **`ROS_NAMESPACE=RosAria rosrun teleop_twist_keyboard teleop_twist_keyboard.py`**

5. Umieścić robota w pozycji startowej przy użyciu `teleop_twist_keyboard`.

6. Otworzyć w edytorze skrypt `cmd_vel_peoplebot.py`. Upewnić się, że flaga SIMULATION ustawiona jest na False.

7. Uruchomić skrypt `cmd_vel_peoplebot.py` wprawiający w ruch robota.

6 Symulator MobileSim

Wytwórca PeopleBota, Adept MobileRobots, udostępnił środowisko MobileSim, w którym wirtualny robot (np. PeopleBot) porusza się na mapie wytworzonej w programie Mapper3 z wykorzystaniem wirtualnych sensorów we współpracy z GUI MobileEyes. Uruchamianie w/w programów, ich konfiguracja i sposób użycia są przedstawione na filmie `https://www.youtube.com/watch?v=Tzz4K6VUSVk&t=625s`.

W celu zaobserwowania możliwości symulowania działania wirtualnego robota w symulatorze MobileSim należy:

1. W nowym terminalu uruchmoć symulator MobileSim `/usr/local/MobileSim/MobileSim`.
2. Wybrać mapę (`/PeopleBot/mobile_robots_mapy/parter0607010-r2023.map`) oraz model robota wirtualnego `peoplebot-sh`.
3. W osobnym terminalu uruchomić znajdujący się w katalogu `/usr/local/Arnl/examples/` skrypt `arnlServer`.
4. Uruchomić aplikację MobileEyes wybierając w polu `Robot Servers` `localhost` (nie podajemy loginu, ani hasła).
5. Zlokalizować robota (`Tools` $\rightarrow$ `Robot Tools` $\rightarrow$ `Localize to Point`)
6. Wreszcie, zadając kolejne pozycje i orientacje robota w MobileEyes (korzystając np. z polecenia `Send Robot`) obserwować jego ruch w symulatorze MobileSim.

7 Publikowanie prędkości dla robota wirtualnego zaimplementowanego w środowisku Gazebo

Na komputerze `urs` dostępne jest środowisko `Gazebo`, a wraz z nim wirtualny robot Pioneer 3-DX. Z perspektywy ROS-owych topików jest on równoważny rzeczywistemu Pioneerowi 3-DX lub PowerBotowi (zbudowanemu na bazie Pioneera).
W celu zaobserwowania możliwości symulowania działania wirtualnego robota w symulatorze Gazebo należy wykonać następujące kroki:

1. Przejść do katalogu `home/robotyka/catkin_ws/src/pioneer_p3dx_model/p3dx_gazebo/launch`).
2. za pomocą polecenia **`roslaunch pioneer3dx.launch`** wywołać wirtualnego Pioneera w środowisku Gazebo. W efekcie pojawi się okno programu Gazebo z wirtualnym Pioneerem 3-DX.
3. Wywołując polecenie **`rostopic list`**, można uzyskać listę dostępnych tematów stowarzyszonych z wirtualnym Pioneerem 3-DC, w tym `/cmd_vel`.
4. W nowym oknie terminala, na komputerze **urs** przejść do kartoteki `/home/robotyka/catkin_ws/src/robotyka_3/scripts`.
5. Otworzyć w edytorze skrypt `cmd_vel_peoplebot.py`. Upewnić się, że flaga SIMULATION ustawiona jest na `True`.

6. Uruchomić skrypt `cmd_vel_peoplebot.py` wprawiający w ruch robota.

7. Warto również wywołać (z katalogu `/home/robotyka/catkin_ws/src/DifferentialDriveRobot/src/myrobot_gazebo/scripts`) i sprawdzić działanie skryptów `square_mode.py` lub `circle_mode.py`.

8. W tym celu uprzednio należy otworzyć w edytorze wybrany plik (ścieżka do plików: `/home/robotyka/catkin_ws/src/DifferentialDriveRobot/src/myrobot_gazebo/scripts`) i odkomentować linię `self.cmd_vel = rospy.Publisher('/cmd_vel', Twist, queue_size=10)`, natomiast zakomentować linię `self.cmd_vel = rospy.Publisher('/RosAria/cmd_vel', Twist, queue_size=10)` i dalej uruchomić skrypt.

9. Uruchomić wybrany skrypt jednym z poleceń

   - rosrun myrobot_gazebo square_mode.py
   - rosrun myrobot_gazebo circle_mode.py

   Wirtualny robot zostanie wprawiony w ruch.

8 Zadania do wykonania

W celu uruchomienia robota oraz realizacji zadania nawigowania robotem wykonaj następujące polecenia:

1. Odłącz kabel zasilacza. Włącz odpowiednio zasilanie platformy mobilnej, komputera pokładowego, oraz monitora (rys. 5).

2. Zaloguj się na serwerze `rab` do konta `ros` (prowadzący wpisze hasło)[‡]:

   ```
   robotyka@urs: ssh -l ros -X rab.kcir.pwr.edu.pl
   ros@rab.kcir.pwr.edu.pl's password:
   ```

3. Uruchom węzeł główny na serwerze `rab`:

   ```
   ros@rab: roscore
   ```

4. Na komputerze pokładowym robota zaloguj się na konto `lirec` (o hasło zapytaj prowadzącego) i upewnij się, że masz połączenie z siecią `LAB010` (rys. 6)

Czynności wstępne:

1. Komputer **urs**: przejdź do kartoteki /home/robotyka/catkin_ws/src/robotyka_3/scripts.

2. Komputer **urs**: wywołaj skrypt `skpoiuj_pliki_bazowe.sh` w celu aktualizacji skryptów.

Do wykonania:

1. Do rozdziału 4:

[‡]Tutaj jako emulator terminala warto wykorzystać `Terminator`, który daje możliwość dzielenia okien

- przećwiczyć przemieszczanie robota za pomocą joysticka,
- zaznajomić się z możliwościami oprogramowania MobileEyes,
- zrealizować ruch robota do zazaczonego na mapie położenia.

2. Do rozdziału 6:

   - zadawać kolejne położenia robota w programie MobileEyes i obserować jego ruch w symulatorze MobileSim.

3. Do rozdziału 5.2

- zmodyfikować skrypt tak, by zrealizować ruch robota do wybranego przez siebie położenia (wykorzystaj odpowiednie polecenie służące do odczytu współrzędnych pozycji). Uruchomić skrypt na komputerze **urs** za pomocą polecenia **ROS_NAMESPACE=peoplebot rosrun robotyka**$_3$*move_powerbot*$2.py$

4. Do rozdziału 5.3

   - przećwiczyć publikowanie prędkości z poziomu klawiatury.

5. Do rozdziału 7:

   - przetestować publikowanie prędkości do robota wirtualnego zaimplementowanego w Gazebo. W szczególności sprawdzić działanie programów: `cmd_vel_peoplebot.py`, `square_mode.py` oraz `circle_mode.py`

6. Do rozdziału 5.4 (Uwaga: każde uruchomienie skryptu publikującego prędkości na robocie należy poprzedzić sprawdzeniem w symulatorze Gazebo poprawności jego działania – patrz rozdział 7):

   - uruchomić skrypt `cmd_vel_peoplebot.py` publikujący prędkości dla robota.
   - Wiedząc, że na ruch robota nałożone jest ograniczenie w postaci braku poślizgu bocznego, a zmienne zadaniowe są następujące $q = (x, y, \theta)^T \in R^3$, bezdryfowy układ sterowania może przybrać formę

$$\dot{q} = \begin{bmatrix} \cos(\theta) & \cos(\theta) \\ \sin(\theta) & \sin(\theta) \\ -\frac{1}{d} & \frac{1}{d} \end{bmatrix} \begin{pmatrix} \eta_1 \\ \eta_2 \end{pmatrix}$$

   lub

$$\dot{q} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix},$$

   gdzie d jest długością odcinka łączącego środek osi kół z punktem styku koła do podłoża. Jaki sens mają elementy η_1 i η_2 oraz u_1 i u_2? Który z modeli zaimplementowano dla fizycznego robota? Zweryfikuj swoje przypuszczenia na robocie.

   - Włączając jedną wybraną składową sterowań a wyłączając pozostałe, np. $u_1 = 1$ i $u_2 = 0$ powodujemy ruch robota wzdłuż trajektorii pola wektorowego – np. generatora $g1$. W wyniku czego możemy powiedzieć, że stosując odpowiednie stałe sterowania uzyskujemy możliwość poruszania się po przestrzeni rozpiętej przez generatory układu $P = span_R\{g_1, g_2\}$. Zbadaj eksperymentalnie jakie kierunki ruchu otrzymujemy bezpośrednio z generatorów. Jakie jeszcze inne kierunki możemy wygenerować przy pomocy sterowań dla obu zapisanych wyżej układów?

- przeprowadź robota w taki sposób, aby zakreślił on kwadrat o zadanej długości boku.
- Formuła Campbella–Bakera–Hausdorffa–Dynkina mówi o tym, że po odpowiednim złożeniu czterech segmentów sterowań uzyskujemy dla małych czasów t

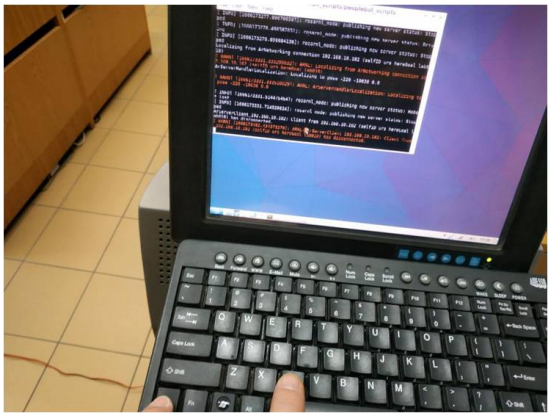
$$\exp(t^2[X,Y]+o(t^3)) = \exp(tX)\exp(tY)\exp(-tX)\exp(-tY) = \exp(tX)\exp(tY)\exp(tX(-1))\exp(tY(-1)).$$

Co oznacza, że jeśli włączymy sterowanie działające na generator X przez czas t, następnie przez taki sam czas włączymy sterowanie generatorem Y , później sterowanie generatorem $-X$ i na końcu $-Y$, to otrzymamy sterowanie generujące ruch wzdłuż pola wektorowego $[X,Y]$ w czasie t^2.
Wiedząc, że kierunek otrzymany poprzez nawias Liego możemy zrealizować za pomo- cą sterowań posługując się formułą Campbella–Bakera–Hausdorffa–Dynkina (for- mułą CBHD) przygotuj eksperyment mający na celu zweryfikowanie poprawności, obliczonego przez siebie w poprzednim zadaniu, nowego (trzeciego) kierunku ru- chu robota. Sprawdź, że kierunek $\exp(t^2[X,Y]+o(t^3))$ uzyskasz również poprzez cykliczną zmianę elementów składowych, i tak

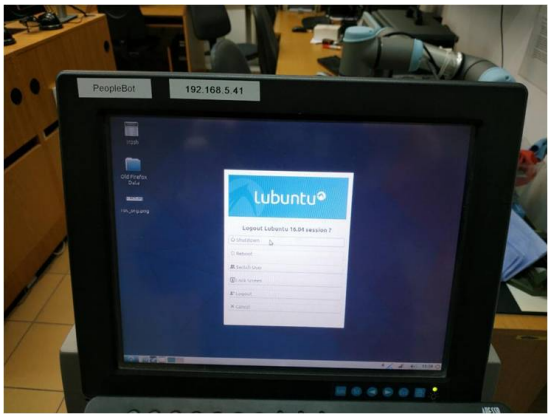
$$\begin{aligned}\exp(t^2[X,Y]+o(t^3)) &= \exp(tX)\exp(tY)\exp(-tX)\exp(-tY) = \\ &\exp(tY)\exp(-tX)\exp(-tY)\exp(tX) = \\ &\exp(-tX)\exp(-tY)\exp(tX)\exp(tY) = \\ &\exp(-tY)\exp(tX)\exp(tY)\exp(-tX)\end{aligned}$$

Co otrzymamy jeśli wygenerujemy ruch $\exp(tX)\exp(tY)\exp(-tY)\exp(-tX)$?
- Zaobserwuj wpływ czasu sterowania na otrzymany ruch w kierunku nawiasu Liego. Wykonaj na robocie manewr parkowania równoległego.

Zakończenie pracy na komputerze pokładowym objaśnia rysunek 15



a) ctrl-C



b) shutdown

Rysunek 15: Zakończenie pracy na komputerze pokładowym robota

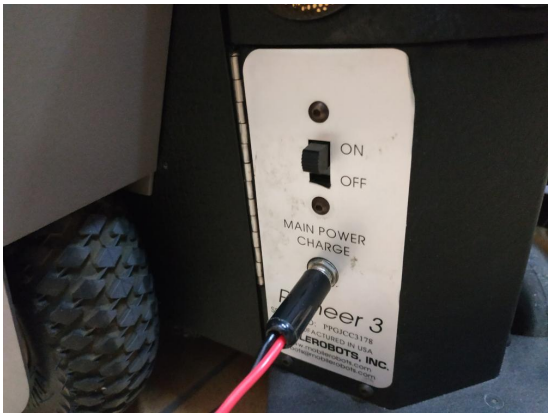
Jeśli logowano się do komputera pokładowego robota z poziomu komputera stasjonarnego, można się wylogować z komputera pokładowego wpisując w linii terminala, z którego uruchomiono komputer pokładowy sekwencję

`ctrl-c`

`robotyka@urs:` **sudo poweroff**

W celu wyłączenia zasilania robota po zakończeniu pracy wyłącz odpowiednio zasilanie monitora, komputera pokładowego i platformy mobilnej (rys. 16).

a)



b)

Rysunek 16: Wyłączanie zasilania w kolejności: a) komputer pokładowy – góra oraz monitor – dół, b) platforma mobilna,

9 Przydatne materiały

1. Materiały wideo
   `https://www.youtube.com/@powerbot_pz2323`,
   `https://www.youtube.com/channel/UCzlpUEEHpU1FjMcJ7h-B_Hw`,
   `https://www.youtube.com/watch?v=Tzz4K6VUSVk&t=625s`.
2. Materiały szkoleniowe
   `https://www.youtube.com/channel/UCzlpUEEHpU1FjMcJ7h-B_Hw`.
3. ROSARIA. `http://wiki.ros.org/ROSARIA`,
4. ROSARNL. `https://github.com/MobileRobots/ros-arnl`. Adept MobileRobots.