Planowanie ruchu robota w przestrzeni 2D z kamerą Kinect

Adrian Glinkowski
Politechnika Poznańska
Poznań, Polska

Małgorzata Zdunek
Politechnika Poznańska
Poznań, Polska

Streszczenie—Celem niniejszej pracy było uruchomienie SLAM gmapping oraz algorytmu planowania trajektorii A* w symulatorze Gazebo oraz na rzeczywistym robocie LABbot. Środowiskiem projektowym było Ubunto 16.04 oraz ROS. Projekt stworzony na zajęcia z Nowoczesnych Sensorów w Robotyce oraz Robotów Autonomicznych.

Index Terms—SLAM, gmapping, ROS, Ubuntu, Kinect, A*

I. Wstęp

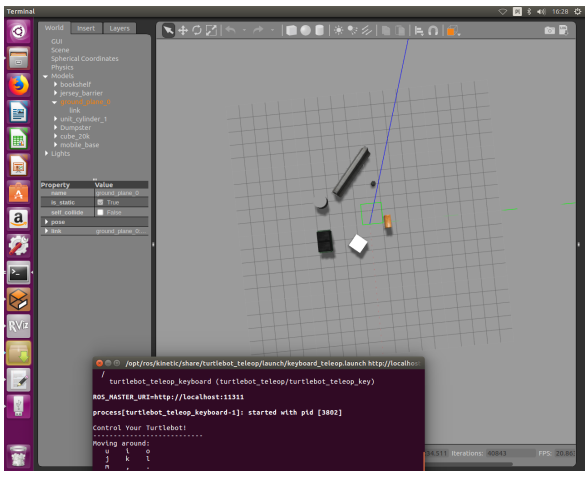
W części symulacyjnej projektu wykorzystano w dużej mierze istniejące już rozwiązania służące do symulowania otoczenia, mapowania oraz nawigacji. Całość oparto na pakietach „turtlebot".

W kolejnym etapie podjęto próby wdrożenia rozwiązania w rzeczywistym robocie. Przeprowadzono testy w laboratorium, aby sprawdzić skuteczność stworzonego systemu.

II. Symulacja Gazebo

A. Uruchomienie symulatora Gazebo

Pierwszym zadaniem było uruchomienie środowiska Gazebo z przygotowanym światem i modelem robota. Wykorzystano gotowy moduł „turtlebot", który zawierał model wraz z odometrią i sterowaniem. Aby aplikacja

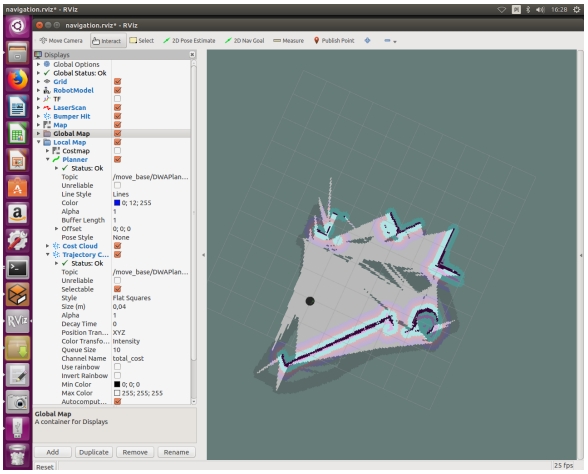


Rysunek 1. Robot w zasymulowanym środowisku

działała poprawnie, należy zainstalować środowisko ROS w pełnej wersji. Przy tworzeniu projektu wykorzystywano wersję Kinetic oraz Ubuntu 16.04. Zainstalowano dodatkowo pakiety „turtlebot", „turtlebot_gazebo", „turtlebot_rviz_launcher".

B. Wykorzystanie gmapping'u

Do budowy mapy otoczenia wykorzystano metodę gmapping. Wdrożono istniejący już pakiet, który jest wbudowany w moduł „turtlebot". Aby rozwiązanie działało poprawnie, dołączono model kamery Kinect, oraz funkcję zmieniającą obraz pobrany z kamery w wiązkę laserową. Nie był potrzebny pełny obraz, gdyż projekt dotyczy planowania w przestrzeni 2D, a dodatkowo sam robot nie daje możliwości przemieszczenia w trzeciej osi. Do stero-



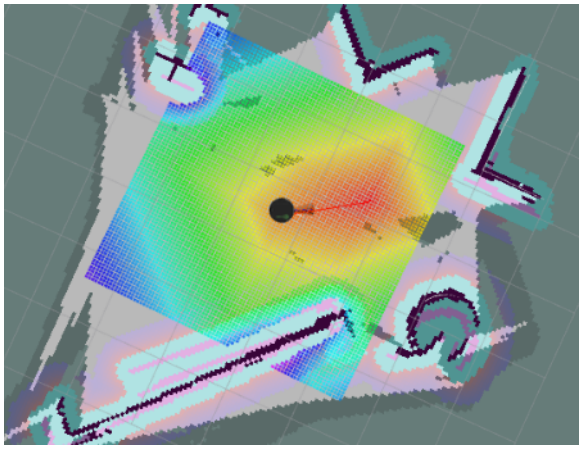
Rysunek 2. Zmapowane pomieszczenie

wania robotem, w celu zbudowania mapy, wykorzystano plik „keyboard_teleop.launch" znajdującego się w pakiecie „turtlebot_teleop"

C. Nawigacja

Do nawigacji wykorzystano gotowy system znajdujący się w „turtlebot_navigation". Nawigacja w tym przypadku odnosi się głównie do planowania przez robota ścieżki ruchu w środowisku, które uprzednio zmapował lub w inny sposób posiadł mapę otoczenia. Planowanie opiera się o algorytm A*. Aby program użył odpowiedniego algorytmu, należało doko-

nać zmiany parametru use_dijkstra w pliku „turtlebot_navigation/param/global_planner_params.yaml” na FALSE. Po uruchomieniu projektu oraz Rviz, można zadawać pozycje i orientację robota za pomocą przycisku 2D Nav Goal. Ścieżka wyrysowuje się na mapie czerwoną linią, jak to jest przedstawione na rysunku 3. Całość



Rysunek 3. Zaplanowana ścieżka wraz z mapą kosztów

programu uruchamia się po włączeniu pliku glizdu.launch.

III. Eksperyment na rzeczywistym robocie

A. Instalacja i uruchomienie LABbota

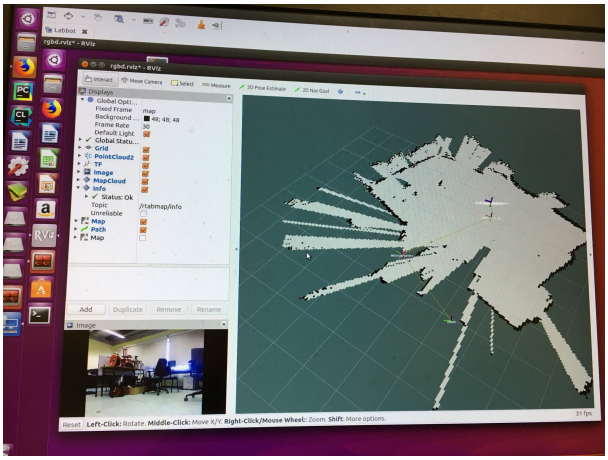
Aby przetestować stworzone rozwiązanie na rzeczywistym robocie, należało zainstalować z repozytorium pakiety umożliwiające sterowanie. Następnie koniecznym było doinstalowanie modułu tmux. Jeżeli to nie byłoby wystarczające i nie dało się uruchomić LABbota, należy dołączyć spacenavd oraz libspnav-dev

B. Uruchomienie przygotowanego rozwiązania

Pierwsza próba uruchomienia robota zakończyła się niepowodzeniem, gdyż próbowano przenieść w całości program z symulacji. Nie było to możliwe, gdyż LABbot posiada wiele funkcji, które pokrywały się lub wzajemnie blokowały z napisanym programem symulacyjnym.

Trzeba było zrezygnować z modelu „turtlebot” i wykorzystać gotowy model LABbot. Należało powiązać układ kamery z układem robota, aby przemieszczała się wraz z nim. W tym celu stworzono static_transform_publisher, który tworzył połączenie miedzy układem współrzędnych robota a kamery. Następnie wykonano kilka testów, które wykazały, że oś X robota, w stosunku do kamery, ma zmieniony znak. Po skorygowaniu znaków w plikach „labbot_odometry.cpp” oraz „labbot_teleoperation_twist.cpp” pozbyto się problemu.

Przeprowadzone eksperymenty wykazały, że udało się dobrze odwzorować pomieszczenie, co widać na rysunku 4. Udało się to wyświetlić dzięki wcześniejszej konfiguracji zdalnego pulpitu. Bez tej opcji późniejsze zadawanie punktów byłoby niemożliwe.



Rysunek 4. Zmapowane rzeczywiste pomieszczenie

Przeprowadzono testy nawigacji, które wyszły pomyślnie. Robot wybierał najbardziej optymalną trajektorię oraz dojeżdżał do punktu z ominięciem przeszkód i zadowalającą dokładnością. Dodatkowo przemieszczając się do określonego miejsca mapował pomieszczenie na bieżąco i uzupełniał istniejącą mapę. Dzięki kamerze można było obserwować otoczenie w czasie rzeczywistym. Cały projekt uruchamiany jest za pomocą pliku „map_service_glizdu.launch”.

Literatura

[1] Repozytorium z całym projektem
https://github.com/glizdu/Projekt

[2] Repozytorium LABbot
https://github.com/PUTvision/ROS-labbot

[3] Dokumentacja ROS
http://www.ros.org/