Mapowanie i lokalizacja robota mobilnego w środowisku ROS symulacja w Gazebo

Łukasz Chojnacki

Koło Naukowe Robotyków KoNaR
WWW.KONAR.PWR.EDU.PL

9 stycznia 2017

Spis treści

1 Wstęp **2**

2 Implementacja **2**

2.1 Robot 4

2.2 Planowanie ruchu 5

2.3 Lokalizacja 6

2.4 Tworzenie mapy 7

3 Podsumowanie **7**

Literatura **8**

Streszczenie

Referat przedstawia sposób tworzenia robota mobilnego potrafiącego rozwiązywać problematykę jednoczesnej lokalizacji i mapowania (SLAM). Robot został zbudowany jako wirtualny obieky w Gazebo, wykorzystujący środowisko robotyczne ROS. Pakiet Navigation Stack, który został użyty, pozwala na autonomiczną nawigację robota mobilnego.

1 Wstęp

Robot, jako urządzenie autonomiczne i inteligentne, powinno poruszać się w otoczeniu pełnym przeszkód, ludzi i nieprzewidywalnych zdarzeń w taki sposób, aby nie powodować zniszczeń oraz większych zmian otoczenia, tzn. podczas poruszania się z jednego pomieszczenia do drugiego, robot nie może zniszczyć ściany, aby zrealizować zadanie. W takim wypadku powinien potrafić stworzyć mapę otoczenia, zlokalizować się oraz wykonać w nim ruch. W tym celu badacze oraz inżynierowie opracowali metody pozwalające na rozwiązanie problematyki jednoczesnej lokalizacji robota i mapowanie jego otoczenia, w języku angielskim znanym pod pojęciem *SLAM* (*simultaneous localization and mapping*). Niektóre z tych rozwiązań zostały zastosowane podczas budowy autonomicznych samochodów na zawody *DARPA Urban Challenge*, które odbyły się 3 października 2007 roku w mieście Victorville, California na zachodnim wybrzeżu USA. Pojazdy miały za zadanie w pełni autonomiczną jazdę oraz parkowanie w środowisku miejskim. Szczegółowy opis samochodów oraz rozwiązań, które zostały w nich zastosowane można znaleźć w [Buehler et al., 2009].

Referat przedstawia praktyczną realizację lokalizacji i mapowania otoczenia na przykładzie prostego dwukołowego robota mobilnego tzw. platformę mobilną (2,0) (*ang. differential wheeled robot*). W pracy zostaną przedstawione poszczególne węzły systemu ROS [Quigley et al., 2009] pozwalające na przeprowadzenie symulacji. Praca nie przedstawia dokładnych algorytmów oraz sposobów postępowania. Można je jednak znaleźć m.in. w następujących publikacjach [Siciliano and Khati, 2016], [Siegwart and Nourbakhsh, 2004], [Thrun et al., 2005].

2 Implementacja

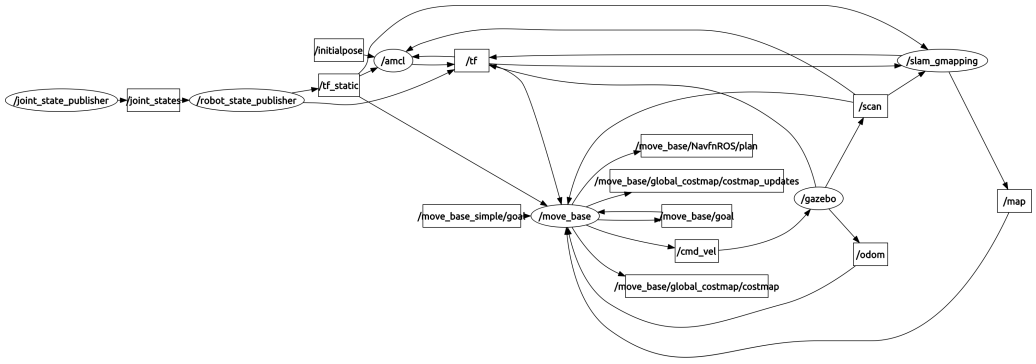
Budowę robota od strony programowej można zdefiniować jako projektowanie jego architektury, dzieląc główne zadanie (np. przejazd robota z jednego

do drugiego pomieszczenia) na mniejsze podzadania (lokalizacja, mapowanie, planowanie trajektorii, kinematyka robota). Dysponując teraz zbiorem podzadań, każde z nim implementuje się jako oddzielny program, tak, że podajemy na jego wejście jakieś dane np. odczyty z skanera laserowego i otrzymujemy dane wyjściowe np. wykrycie przeszkody. Następnie każdy program należy połączyć w jedną całość tworząc system robotyczny. ROS (*Robotic Operating System*) pomaga tworzyć tak zaprojektowaną architekturę dostarczając szeregu narzędzi do ich monitorowania oraz debugowania, gdzie każdy program, który realizujące dane podzadanie nazywa się węzłem (*ang. node*) i komunikuje się przez tzw. *topic*. Więcej na temat samego ROSa można znaleźć pod adresem `http://wiki.ros.org/ROS/Tutorials` oraz w książce [O'Kane, 2013]. W dalszej części publikacji autor zakłada, że czytelnik posiada podstawową znajomość środowiska ROS.

W podrozdziałach zostały opisane poszczególne węzły systemu realizujące następujące zadania:

- reprezentacja robota jako zbiór ogniw (*ang. link*) i przegubów (*ang. joint*) – plik URDF
- planowanie ruchu robota – węzeł move_base
- lokalizacja robota – węzeł amcl
- tworzenie mamy otoczenia – węzeł slam_gmapping

Wzajemną relację wyżej wymienionych węzłów przedstawia rysunek 1. Węzły



Rysunek 1: Wzajemna relacja węzłów i przepływ informacji

odpowiedzialne za lokalizację i planowanie ruchu robota należą do pakietu *Navigation Stack*[1], a natomiast węzeł tworzący mapę otoczenia znajduje się

[1] `http://wiki.ros.org/navigation`

w pakiecie *gmapping*[2]. Więcej informacji na ich temat można znaleźć w rozdziale 4 książki [Joseph, 2015].

2.1 Robot

Definicja robota (jego kinematyki i dynamiki) przebiega w pliku URDF (Unified Robot Description Format), który jest formatem XML dla reprezentowania modelu robota. Rozpoczyna się ona i kończy znacznikami `<robot>`, następnie podaje poszczególne człony podając parametry inercyjne, masowe, wizualizacyjne i kolizyjne.

```
<link name="example_link">
  <inertial>Podaj parametry inercyjne</inertial>
  <visual>Podaj parametry wizualizacyjne </visual>
  <collision>Podaj parametry kolizyjne</collision>
</link>
```

Należy także zdefiniować przeguby podając pomiędzy jakimi członami chcemy go zdefiniować oraz gdzie ma się znajdować.

```
<joint name="example_joint" type="fixed">
  <parent link="example_link"/>
  <child link="example_link_1"/>
  <origin xyz=... rpy=.../>
</joint>
```

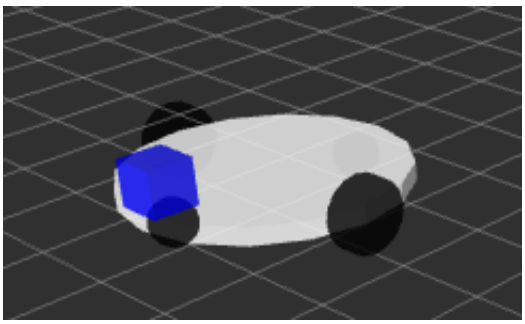
W omawianym przykładzie symulowany był robot dwukołowy (rys. 2), który posiadał skaner laserowy. Jeżeli chce się, aby robot posiadał daną funkcjonalność podczas symulacji np. działający skaner laserowy, sterownik robota (2,0) można dołączyć ją, dodając odpowiedni plugin na przykład w następujący sposób

```
<gazebo reference="hokuyo_link">
  <sensor name="head_hokuyo_sensor" type="ray">
                ...
    <plugin filename="libgazebo_ros_laser.so"
          name="gazebo_ros_head_hokuyo_controller">
      <topicName>/scan</topicName>
      <frameName>hokuyo_link</frameName>
    </plugin>
  </sensor>
</gazebo>
```

[2] `http://wiki.ros.org/gmapping`

```
<!-- Differential drive controller -->
<gazebo>
  <plugin filename="libgazebo_ros_diff_drive.so"
        name="differential_drive_controller_front">
                ...
  </plugin>
</gazebo>
```

Napisany model można zweryfikować korzystając z węzłów joint_state_publisher, robot_state_publisher oraz w programie rviz[3] lub na stonie internetowej `http://mymodelrobot.appspot.com/`.



Rysunek 2: Wizualizacja robota w programie *rviz*

2.2 Planowanie ruchu

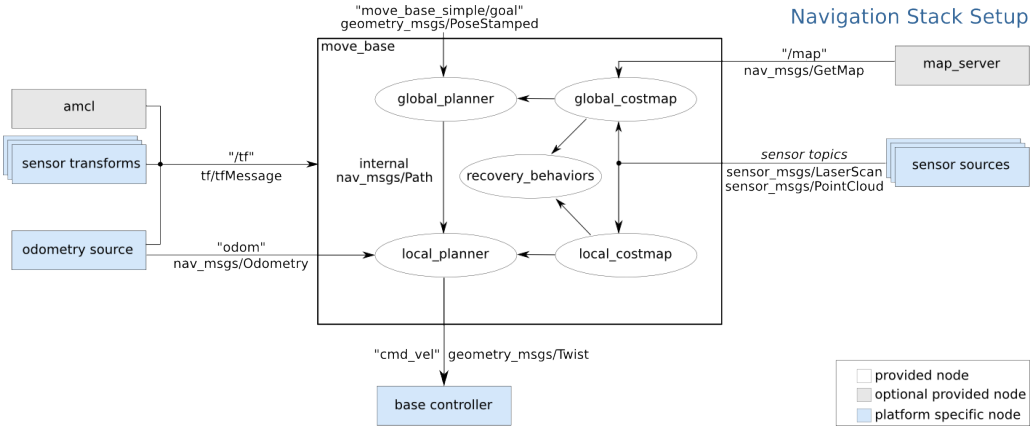
Węzeł move_base z pakietu, który umożliwia ruch robota z punktu startowego do zadanego punktu, bez powodowania kolizji z otoczeniem. Niestety posiada on kilka ograniczeń:

- pracuje lepiej z platformami mobilnymi (2,0) niż np. z samochodem kinematycznym.
- robot musi mieć możliwość sterowania nim przez zadanie prędkości liniowej i kątowej platformy.
- robot powinien mieć zamontowany skaner laserowy.
- działa lepiej z platformami o kształcie okręgu lub kwadratu.

[3] `http://wiki.ros.org/rviz`

Analizując schemat blokowy (rys. 3) programu, aby móc skorzystać z jego możliwości należy podać na wejście:

- źródło danych odometrycznych – na podstawie tych danych możliwe jest określenie pozycji robota od stanu początkowego. Głównym źródłem danych jest informacja z enkodera, IMU oraz kamery 2D/3D (odometria wizualna). Wiadomość jaką otrzyma węzeł powinna być typu `nav_msgs/Odometry`.
- źródło danych sensorycznych – na podstawie tych danych budowana jest globalna i lokalna mapa kosztów. Głównym źródłem danych jest skaner laserowy i/lub czujnik RGB-D (np. Kinect). Wiadomość jaką otrzyma węzeł powinna być typu `sensor_msgs/LaserScan` lub `sensor_msgs/PointCloud`.
- transform_frame – platforma powinna publikować zależności pomiędzy układami współrzędnymi członów robota po przez wiadomości `tf`.
- mapę otoczenia
- punkt docelowy do którego ma dojechać robot



Rysunek 3: Schemat blokowy węzła move_base z dokumentacji

2.3 Lokalizacja

Za lokalizację odpowiedzialny jest węzeł amcl (*adaptive monte carlo localization*). Metoda ta korzysta z filtru cząsteczkowego do śledzenia trajektorii pozycji robota w znanym otoczeniu. Większość zaimplementowanych algorytmów i ich parametry są dobrze opisane w książce [Thrun et al., 2005]. Na

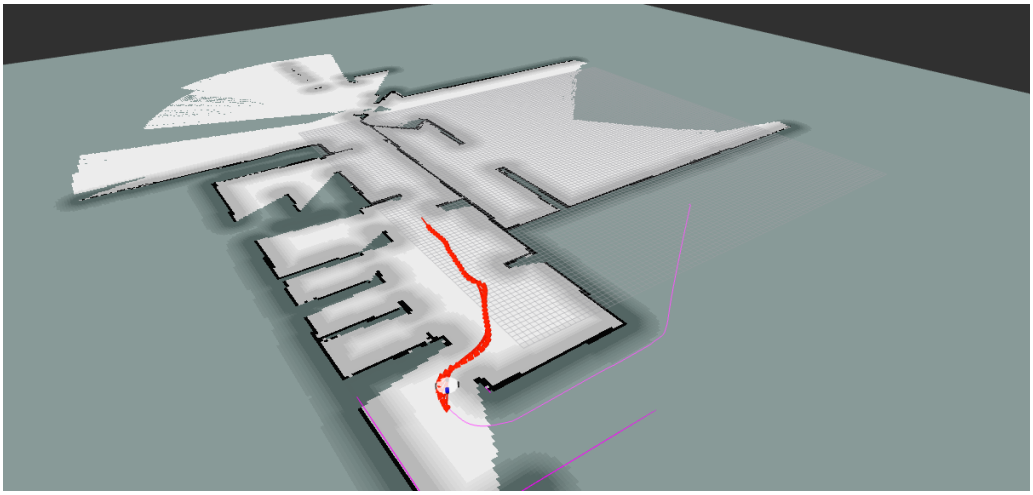
wejście węzła należy podać mapę, odczyty ze skanera laserowego oraz tzw. transform frame, czyli układy współrzędnych członów robota. Wspomniane transform frame można otrzymać z węzła robot_state_publisher, który został wspomniany wcześniej. Na wyjściu otrzymuje się estymowaną pozycję robota względem otrzymanej mapy.

2.4 Tworzenie mapy

Tworzona mapa jest dwuwymiarową tzw. *occupacy grid map*, czyli siatkową mapą zajętości. Oznacza to, że mapa jest podzielona na mniejsze kwadraty tworząc siatkę, na którą nanoszona jest informacja czy na danym polu znajduje się przeszkoda lub nie. Można to porównać do obrazu, który jest podzielony na piksele, gdzie każdemu pikselowi przypisana jest liczba oznaczająca kolor. Mapa tworzona jest na podstawie transform frame oraz odczytów ze skanera laserowego.

3 Podsumowanie

W pracy została w skrócie przybliżona problematyka lokalizacji robot i mapowania otoczenia oraz został zaproponowany sposób w jaki można zbudować robota mobilnego potrafiącego autonomicznie poruszać się w otoczeniu z różnymi przeszkodami. Oprogramowanie robota zostało oparte o środowisko robotyczne ROS, które dostarcza szeregu rozwiązań upraszczające tworzenie systemów robotycznych. Robot został zamodelowany w pliku URDF. Zostały przeprowadzone symulacje w Gazebo (rys. 4) pozwalające zweryfikować poprawność zbudowanego systemu. Wideo demonstracyjne można znaleźć na portalu YouTube pod adresem `https://www.youtube.com/user/KoNaRobotics/`. Pakiet systemu ROS wraz z instrukcją umożliwiającą przetestować zaproponowane rozwiązanie można znaleźć w repozytorium git pod adresem `https://bitbucket.org/konar/`.



Rysunek 4: Robot realizujący zadanie SLAM

Literatura

[Buehler et al., 2009] Buehler, M., Iagnemma, K., and Singh, S. (2009). *The DARPA Urban Challenge, Autonomous Vehicles in City Traffic*. Springer-Verlag New York.

[Joseph, 2015] Joseph, L. (2015). *Mastering ROS for Robotics Programming*. Packt Publishing.

[O'Kane, 2013] O'Kane, J. M. (2013). *A Gentle Introduction to ROS*. Independently published. Available at `http://www.cse.sc.edu/~jokane/agitr/`.

[Quigley et al., 2009] Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*.

[Siciliano and Khati, 2016] Siciliano, B. and Khati, O. (2016). *Springer Handbook of Robotics*. Springer-Verlag New York.

[Siegwart and Nourbakhsh, 2004] Siegwart, R. and Nourbakhsh, I. R. (2004). *Introduction to Autonomous Mobile Robots*. MIT Press.

[Thrun et al., 2005] Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics*. MIT Press.