



POLITECHNIKA POZNAŃSKA

LABORATORIUM PROBLEMOWE 1



RAPORT - AUTONOMICZNE ROBOTY MOBILNE

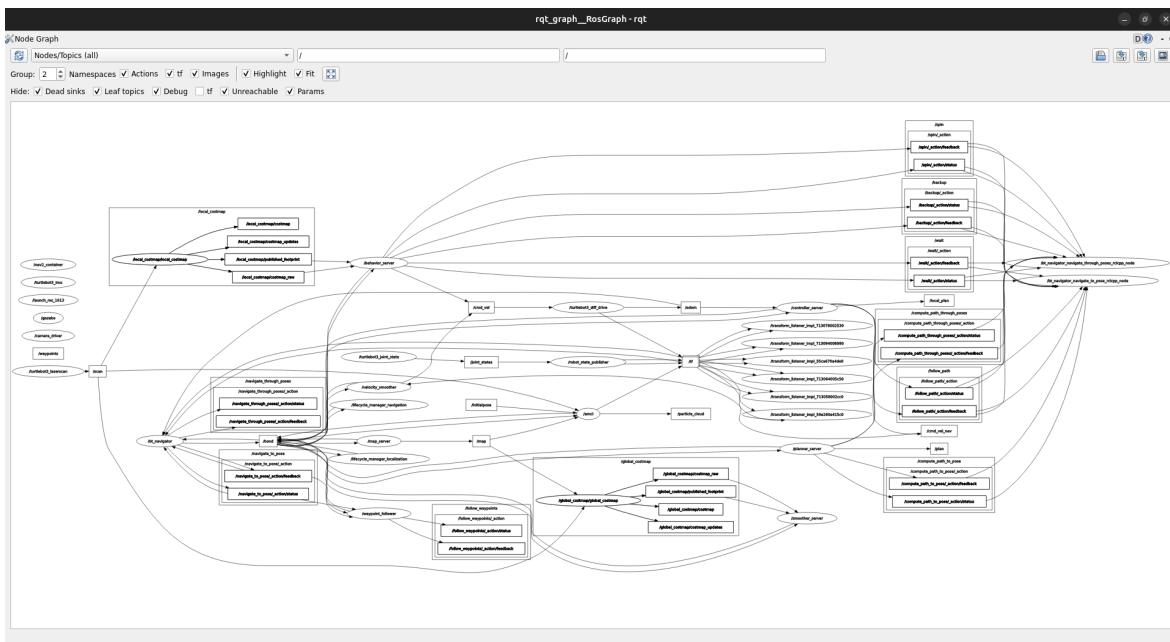
KACPER KRASIŃSKI, 151234

KACPER.KRASINSKI@STUDENT.PUT.POZNAN.PL

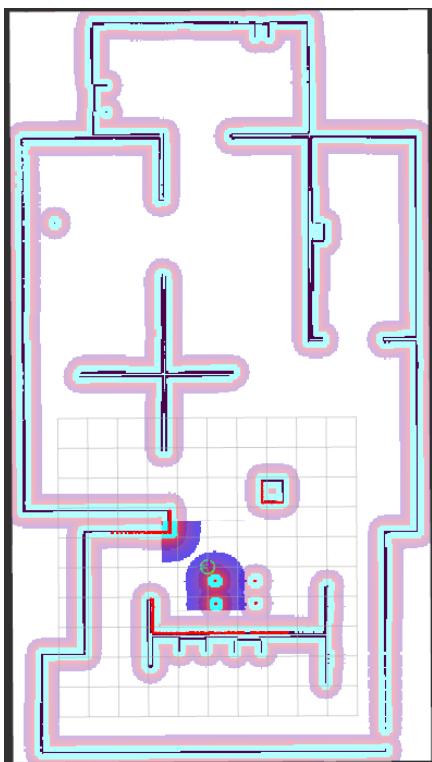
1 Cel i założenia

krótki opis problemu oraz przyjętych ograniczeń.

Problem zadania polega na znalezieniu pozycji pięciu znaczników ArUco i podaniu ich pozycji. Robot zna mapę po której się porusza, ma ograniczony skaner laserowy do wartości 3.5m. Jest to robot TurtleBot3 typu waffle - środowisko zapewnia od początku poniższe tematy i węzły oraz akcje.



Rys. 1. *rqt_graph*



Rys. 2. Znana mapa wyświetlona w aplikacji RViz

```
root@krasa-35-OMEN-Laptop-15-en0xxx:/arm_ws# ros2 action list
/backup
/compute_path_through_poses
/compute_path_to_pose
/drive_on_heading
/follow_path
/follow_waypoints
/navigate_through_poses
/navigate_to_pose
/smooth_path
/spin
/wait
```

Rys. 3. wynik komendy `ros2 action list`

2 Strategia działania

najważniejsze kroki algorytmu oraz podział odpowiedzialności między node'ami.

Rozłożymy działanie algorytmu na kilka węzłów.

2.1 Preprocessing mapy

Wzorując się na mapie na rysunku 2 mamy obszary białe, które są niedostępne, aby efektywnie przeszukać każdą taką mapę przydałby się algorytm przeszukiwania - BFS / DFS. Aby jednak uproszczyć sobie życie powinniśmy zacząć od pogrubienia ścian mapy. Tak pogrubioną mapę można będzie rozważyć pod kątem zajętości. Do tych celów przydałyby się dwie funkcjonalności - można wykonywać je w tym samym wątku.

2.1.1 Pogrubienie mapy

Funkcjonalność odpowiedzialna za pogrubienie czarnych elementów na mapie - subskrybent topica `/map` - publikuje topic `/map_bold` lub wykorzystanie wyłącznie na potrzeby drugiej funkcjonalności wewnętrznie.

2.1.2 Zdefiniowanie dostępnych przez robota pozycji

Funkcjonalność wykorzystująca np. algorytm BFS wyłącznie do utworzenia struktury dostępnych pozycji na mapie. Service albo Action z informacją `/map_bold`, szerokości robota, zasięgu skanera - uwzględnienie tych informacji w algorytmie tak, żeby każde miejsce było pokryte z jakiejś pozycji. Funkcjonalność zwraca strukturę z dostępnymi pozycjami wraz z możliwymi dojściami do tej pozycji - takie, które można przekazać robotowi do intrukcji ruchowej. Rasteryzacja mapy co do jakiejś dokładności.

2.2 Globalne sterowanie robota

Dzięki poznanej technice drzew behawioralnych rozwiążemy problem poruszania robota. Tutaj będziemy decydowali kiedy robot ma być nawigowany do kolejnego celu, oraz akcja dojazdu do celu może być cancelowana jeżeli lokalny system (odczyt ze sknera Lidar zgłosi zagrożenie). Wtedy kontrolę przejmowałby lokalny system sterowania, żeby wyprowadzić robota z pozycji niebezpiecznej - zbyt blisko przeszkodej.

2.2.1 Sposoby wysyłania pozycji docelowych

Teraz za pomocą jednej z poniższych opcji możemy poruszać robotem po uzyskanych punktach. Poniżej zapisano wywołania z CLI, jednak docelowo usługę takiego interfejsu należy zaimplementować w wątku.

- Wysyłanie celu za pomocą akcji

```
01.      ros2 action send_goal /navigate_to_pose nav2_msgs/action/NavigateToPose "{  
02.          pose: {  
03.              header: {  
04.                  frame_id: 'map'  
05.              },  
06.              pose: {  
07.                  position: {x: 2.0, y: 1.0, z: 0.0},  
08.                  orientation: {x: 0.0, y: 0.0, z: 0.0, w: 1.0}  
09.              }  
10.          }  
11.      }" --feedback
```

- Wysyłanie jednorazowego celu na topic

```
01.      ros2 topic pub --once /goal_pose geometry_msgs/msg/PoseStamped "{  
02.          header: {frame_id: 'map'},  
03.          pose: {  
04.              position: {x: 3.0, y: 2.0, z: 0.0},  
05.              orientation: {x: 0.0, y: 0.0, z: 0.0, w: 1.0}  
06.          }  
07.      }"
```

- Wysyłanie większej ilości punktów poprzez akcję

```
01.      ros2 action send_goal /follow_waypoints nav2_msgs/action/FollowWaypoints
02.          "{"
03.          poses: [
04.              {
05.                  header: {frame_id: 'map'},
06.                  pose: {
07.                      position: {x: 1.0, y: 0.0, z: 0.0},
08.                      orientation: {w: 1.0}
09.                  }
10.              },
11.              {
12.                  header: {frame_id: 'map'},
13.                  pose: {
14.                      position: {x: 2.0, y: 1.0, z: 0.0},
15.                      orientation: {w: 1.0}
16.                  }
17.              }
18.          ]" --feedback
```

2.2.2 Przydatne interfejsy sterowania

- topic /amcl_pose - aktualna pozycja robota
- topic /plan - planowana ścieżka
- topic /goal_pose - temat z którego robot subskrybuje funkcjonalność dojazdu robota
- akcja /navigate_to_pose - wysłanie celu do osiągnięcia przez robota
- akcja /follow waypoints - wysłanie ścieżki do wykonania przez robota

2.3 Lokalne sterowanie robota

Jeżeli skaner wykryje, że znajdujemy się zbyt blisko ściany lub zbliżamy się dość szybko do ściany na wprost kontrolę przejmuje lokalne sterowanie, którego zadaniem będzie wyprowadzenie robota w stronę wysłanego celu, ale nie zbliżając się do przeszkody. Jeżeli jest taka możliwość to próbujemy dojść do zadanej pozycji zwiększąc threshold i 'odhaczyć' zadany punkt. Jeżeli to niemożliwe to należy zaktualizować strukturę, że tą stroną nie dojdziemy do zadanego punktu.

2.3.1 Przydatne interfejsy sterowania

- wyjście z node'a globalnego sterowania
 - pogrubiona mapa
 - Struktura z dostępnymi punktami i ścieżkami dojścia
- topic /amcl_pose - aktualna pozycja robota
- topic /local_plan - status lokalnego planera
- topic /scan - aktualne odczyty sensora lidar

2.4 Odczytywanie kodu ArUco

Węzeł odpowiedzialny za monitorowanie topica zwracającego obraz z kamery i odczytywanie kodu ArUco i jego pozycji. Po odczytaniu takiego kodu jego pozycja zapisywana jest w tablicy z wynikami. na podstawie aktualnej pozycji robota, odczytu ze skanera i pozycji kodu ArUco na obrazie.

2.4.1 Czytanie kodu i pozycji

Wykonywane na podstawie obrazu z kamery za pomocą zewnętrznej biblioteki

2.4.2 Liczenie położenia kodu na mapie

Liczone na podstawie pozycji kodu na obrazie, pozycji robota i czujnika ze skanera Lidar. Można wprowadzić uśrednioną pozycję na podstawie odczytów z różnych pozycji robota.

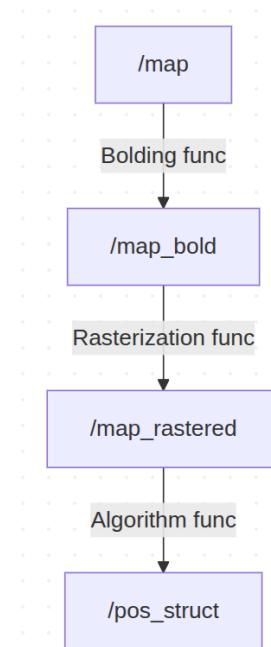
2.4.3 Przydatne interfejsy sterowania

- topic /camera/image_raw - obraz z kamery zamontowanej na robocie
- topic /amcl_pose - aktualna pozycja robota
- topic /scan - aktualne odczyty sensora lidar
- topic /cmd_vel - bezpośrednie sterowanie robotem

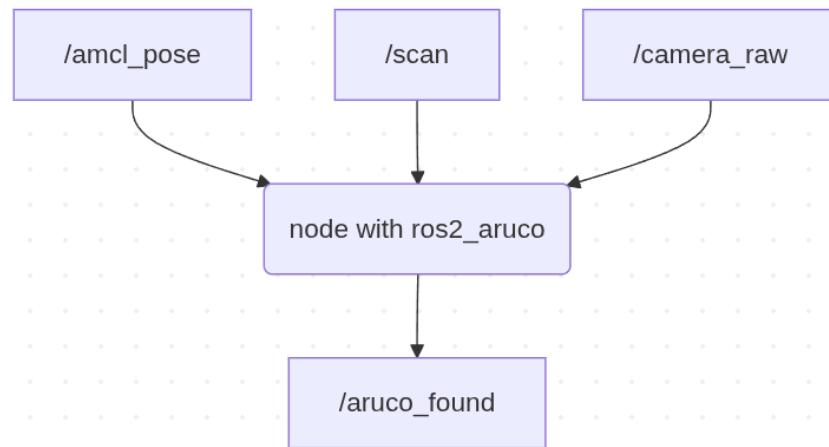
3 Schematy i diagramy

diagramy blokowe, diagramy przepływu danych między node'ami itp.

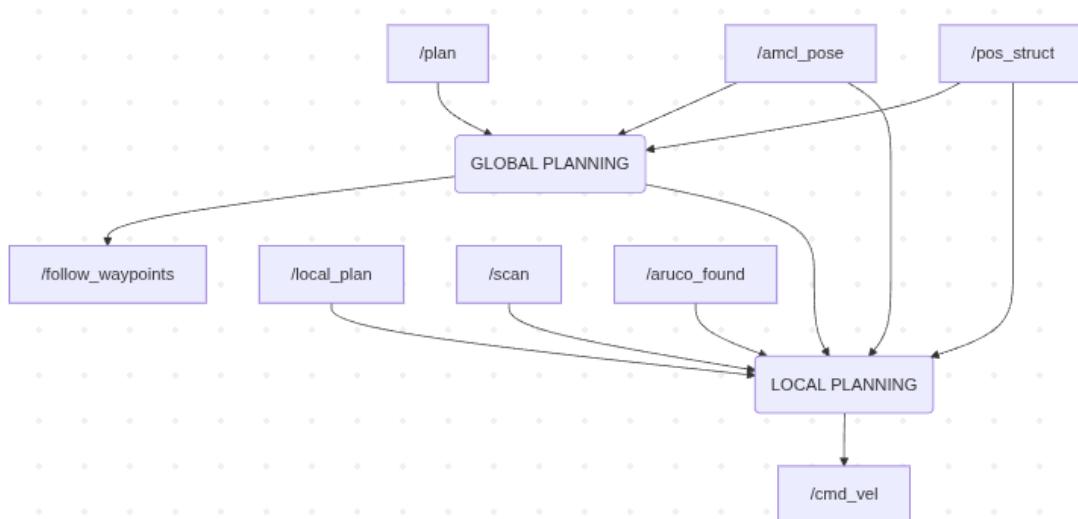
Poniżej diagram preprocessingu mapy oraz planowania



Rys. 4. Preprocessing mapy



Rys. 5. Znajdowanie i obliczanie pozycji kodu ArUco



Rys. 6. Planowanie ruchu globalne i lokalne

4 Wykorzystane narzędzia

biblioteki, paczki ROS2 i inne zasoby zewnętrzne.

- Do logiki drzew decyzyjnych skorzystamy z poznanej na zajęciach biblioteki py_trees_ros
- Do sterowania globalnego użyjemy biblioteki nav2
- Do odczytywania kodów ArUco można użyć paczki ros2_aruco, do tego potrzebne jest podłączenie pythona z pobranymi modułami opencv pod ROS2. Aby to było możliwe należy także:
 - Skonfigurować opencv oraz transforms3d pod środowiskiem wirtualnym venv, oraz przekazać odpowiednią ścieżkę PYTHON do parsowania skryptów python - [link](#). Dzieje się tak ponieważ niektóre paczki nie są bezpośrednio dostępne za pomocą plików package.xml i CMakeLists i nie mają skonfigurowane swoich symboli w rosdep - w ten sposób unikamy konieczności oczekiwania na pullrequest do githuba z kluczami rosdep - [link](#)

5 Potencjalne ryzyka

spodziewane trudności, sposoby ich weryfikacji lub obejścia.

- Problemy z wyjazdem robota z ciasnych punktów. Potencjalne rozwiązanie - system wykrywania korytarzy i ciasnych przejść
- Pominięcia kostek z kodami ArUco, które znajdują się po boku robota. Potencjalne rozwiązanie - obrót robota o 180 stopni w niektórych, zdefiniowanych na podstawie odległości od ścian miejscach
- Błędna rasteryzacja mapy. Potencjalne rozwiązanie - rasteryzacja zależna od komórek obok - Potencjalny problem - łączenie zależności i połączeń pomiędzy rastrami

6 Wnioski

podsumowanie, możliwe kierunki dalszego rozwoju.

Zadanie wymagające wykorzystania nowych bibliotek znalezionych w internecie - należałyby najpierw sprawdzić poprawność działania tych bibliotek, albo potwierdzić na podstawie popularności bibliotek. Integracja sterowania robota mobilnego z sprzężeniem z czujników i nowy dodatkowy element - obrazem z kamery. Scena prosta do modyfikacji i utrudnień w celu sprawdzenia działania algorytmu. Dobry przykład na wykorzystanie sterowania behawioralnego. Ciekawa scena z możliwością zaimplementowania większej ilości zadań - na przykład lokalizowanie się robota na podstawie odczytanych kodów ArUco zajmując położenie kostek - można wykorzystać zaimplementowane funkcje do takiego celu lokalizowania robota w znany otoczeniu np. na fabryce.