Sprawozdanie blok mobilny

Marcin Hanas, Rafał Szczepanik

Spis treści

2. Laboratorium 1.

2.1. Zadanie 1.

Aby zrealizować to zadanie napisano węzeł - „Lab1”, który subskrybuje topic – „Lab1_topic” i publikuje na topic – „mux_vel_nav/cmd_vel” W „Lab1_topic” zostaje przekazana pozycja, którą osiągnąć ma robot. Po otrzymaniu danych wywoływana jest funkcja „move_elektron”, która wylicza odpowiednie parametry ruchu i publikuje je na odpowiedni topic.

Opis algorytmu interpolacji liniowej punktów na podstawie zadawanych prędkości:

1. Otrzymanie pozycji do osiągnięcia.
2. Wyliczenie kąta o jaki należy obrócić robota przy pomocy funkcji *atan2*
3. Z twierdzenia pitagorasa wyznaczenie odległości jaka dzieli obecną pozycję od pozycji zadanej
4. Na podstawie znanych prędkości wyliczenie czasu przez jaki zadawana ma być prędkość kątowa
5. Na podstawie znanych prędkości wyliczenie czasu przez jaki zadawana ma być prędkość liniowa
6. Zadanie prędkości kątowej przez czas z 4.
7. Zadanie prędkości liniowej przez czas z 5.

2.2. Zadanie 2.

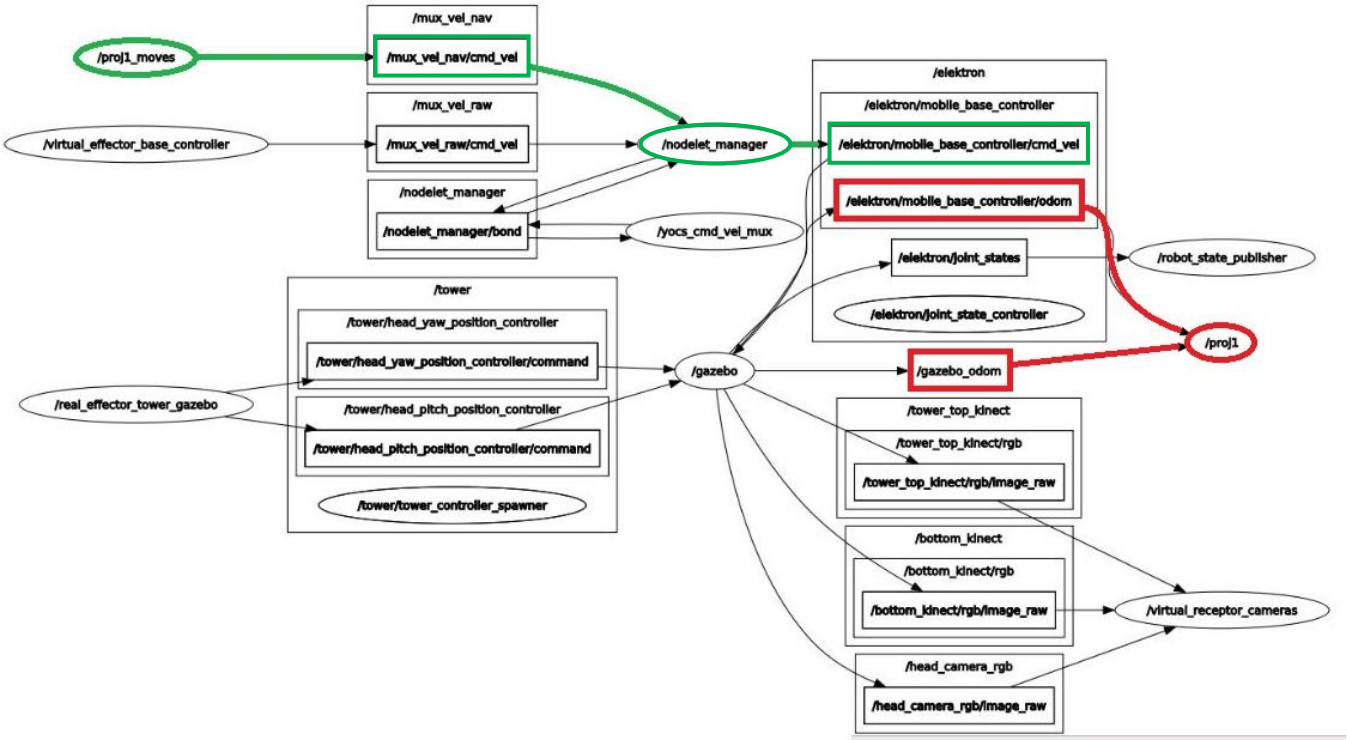
Aby zrealizować to zadanie napisano węzeł klienta - „Lab1_2”, który korzysta z serwisu – „path_follower”. Serwis ten na podstawie danych z odometrii podaje publikuje na topic – „mux_vel_nav/cmd_vel” odpowiednie prędkości do momentu, gdy żądana pozycja nie zostanie osiągnięta – robot przestanie zbliżać się do danego punktu.

Opis algorytmu interpolacji liniowej punktów na podstawie danych odometrii:

1. Otrzymanie pozycji do osiągnięcia
2. Wyliczenie kąta jaki powinien osiągnąć robot
3. Wyznaczenie zwrotu prędkości kątowej
4. Zadanie prędkości kątowej i sprawdzanie czy odpowiednia pozycja została osiągnięta
5. Jeśli pozycja została osiągnięta zatrzymanie robota
6. Zadanie prędkości liniowej i sprawdzanie czy odpowiednia pozycja została osiągnięta
7. Jeśli pozycja została osiągnięta zatrzymanie robota

3. Projekt 1.

3.1. Struktura oprogramowania do zbierania danych

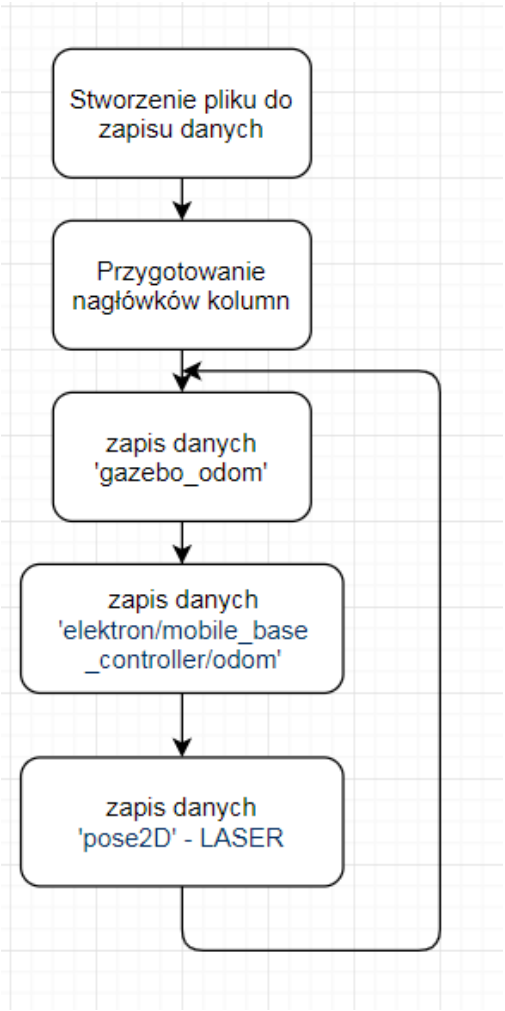


Na **zielono** zaznaczono ścieżkę odpowiadającą za zadawanie pozycji robota.

Na **czerwono** zaznaczono sposób zbierania danych.

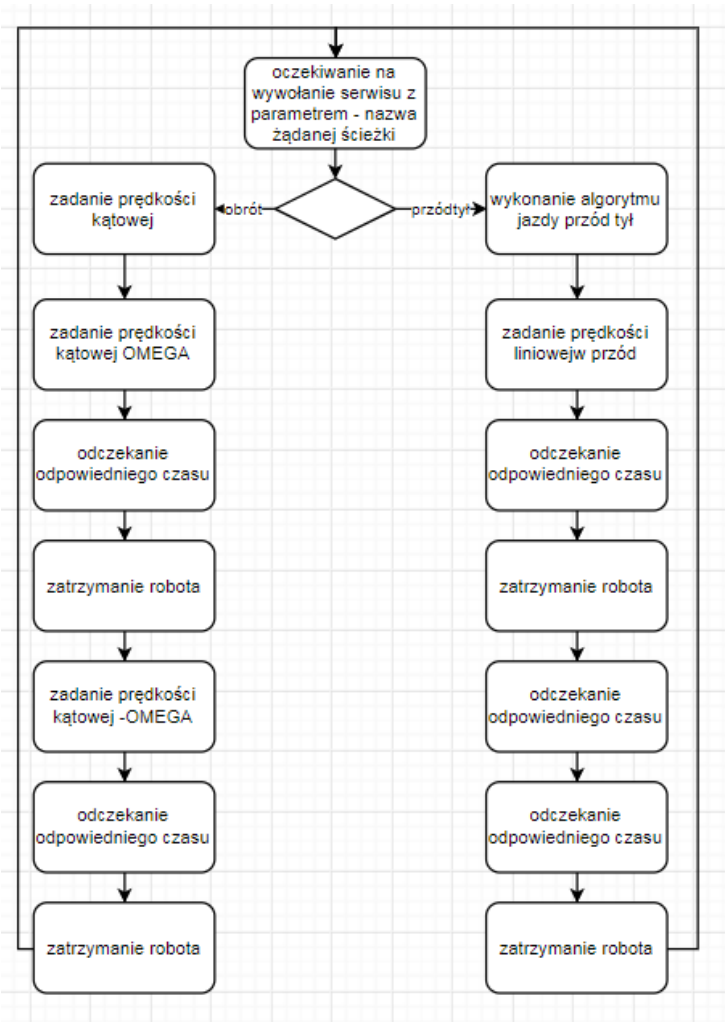
3.2. Opis działania węzła zbierającego dane

Węzeł ‘proj1’ w pętli zbiera odpowiednią ilość próbek danych z trzech źródeł – odometria globalna (gazebo), tune lub diff controller (w zależności od ustawienia), laser. Wszystkie dane zapisuje do pliku w odpowiednich kolumnach.



3.3. Opis działania węzła sterującego robotem

W projekcie należało zrealizować trzy rodzaje ruchu – obrót, jazdę w przód i tył oraz jazdę po kwadracie. Dwie pierwsze ścieżki zostały zaimplementowane w node ‘proj1_moves’. Kwadrat został zrealizowany przy pomocy serwisu z laboratorium 1, który interpoluje ruch na podstawie odczytów z odometrii. Podawano kolejne współrzędne kwadratu.



3.4. Sposób analizy danych

Porównywano cztery możliwe konfiguracje:

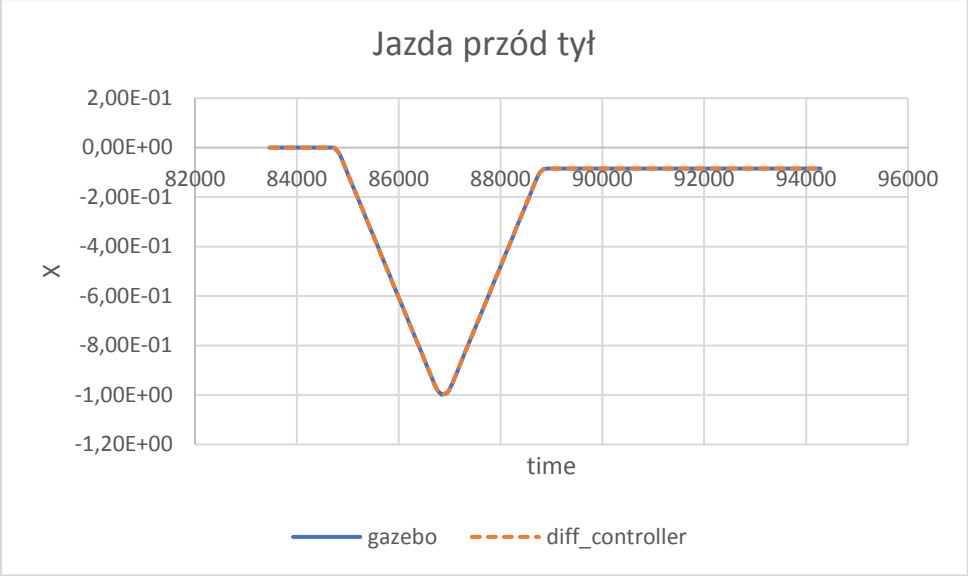
1. diff_controller
2. diff_controller + laser
3. tune_controller
4. tune_controller + laser

Dla każdej z powyższych sytuacji zebrano 4 zestawy danych. Wykonano wykresy położenia od czasu porównując ścieżki z danymi z odom (gazebo), które traktowano jako referencyjne. Dla każdej sytuacji wyliczono błędy i zebrano w tabeli.

3.5. Wykresy i wnioski

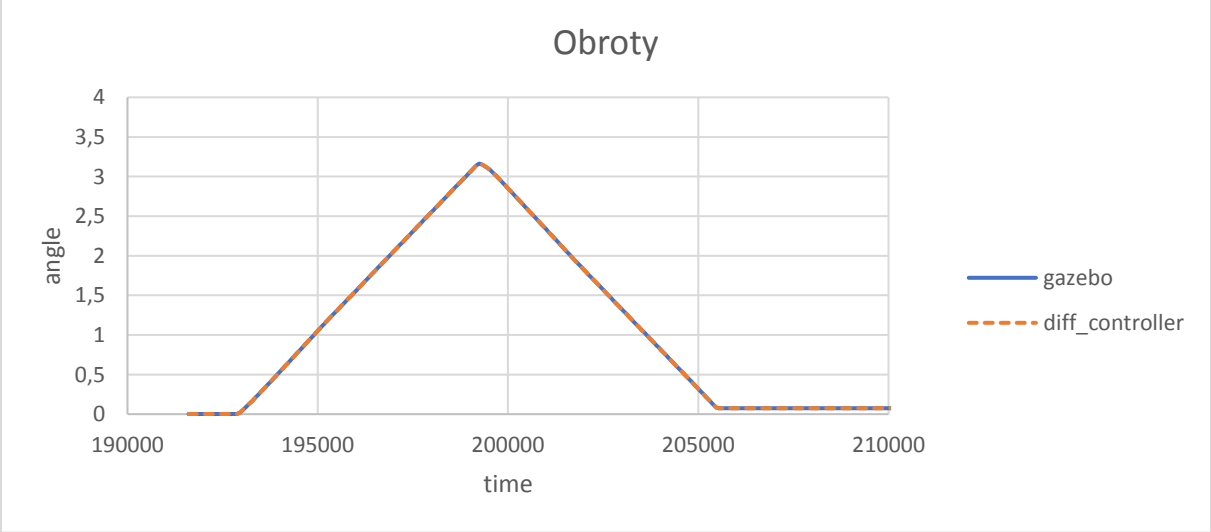
Wyniki diff_controller

Jazda przód – tył



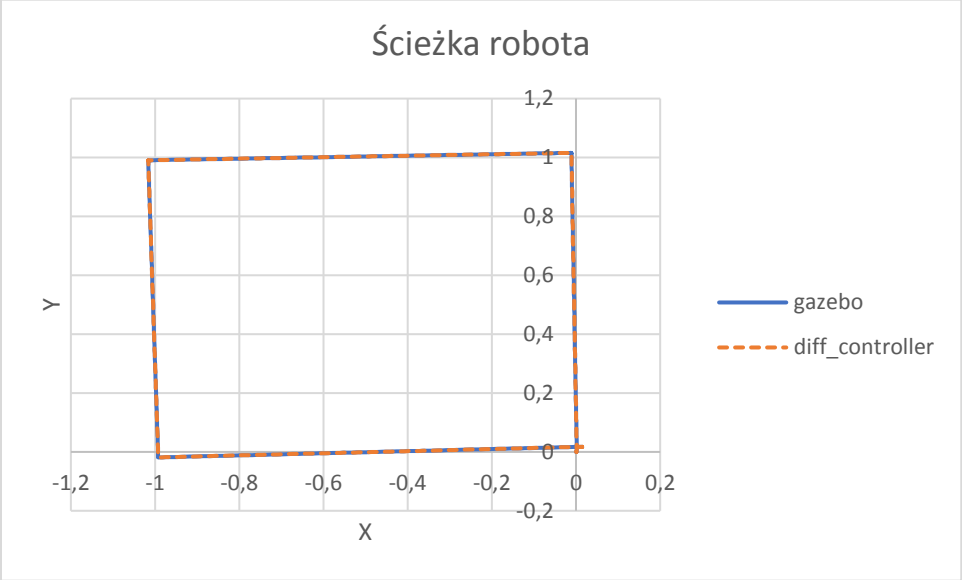
Błędy				
	1	**2**	**3**	**4**
X_gazebo	-9,99E-01	-0,99999	-0,99923	-0,99979
X_diff	-9,98E-01	-0,99942	-0,99986	-0,99772
dX	0,000903	0,000576	0,000632	0,002074

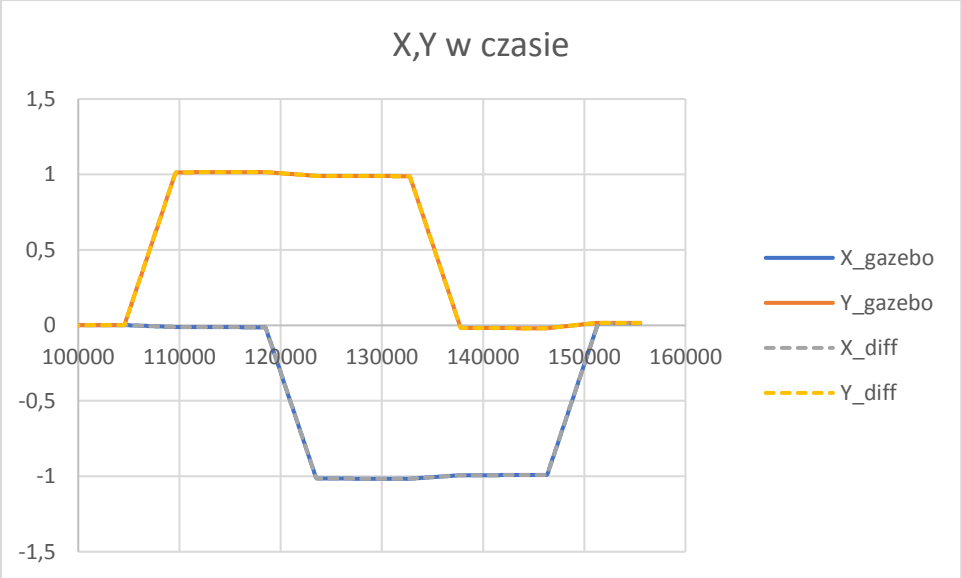
Obroty



Błędy				
	1	**2**	**3**	**4**
Th_gazebo	3,161624	3,706281	3,706726	3,707141
Th_tune	3,161651	3,149678	3,148793	3,149489
dTh	2,69E-05	0,556603	0,557933	0,557652

Test kwadratu:

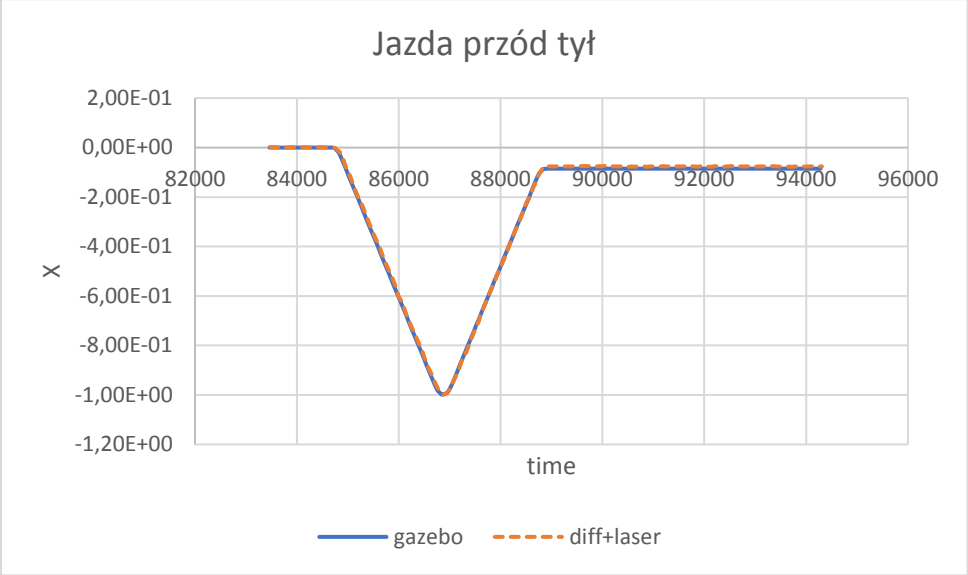




Błędy				
	1	**2**	**3**	**4**
dX	1,2E-10	3,86E-11	1,51E-10	9,54E-11
dY	1,29E-11	1E-11	5,7E-11	2,65E-11
dTh	6,04E-10	3,76E-11	2,47E-10	2,06E-10

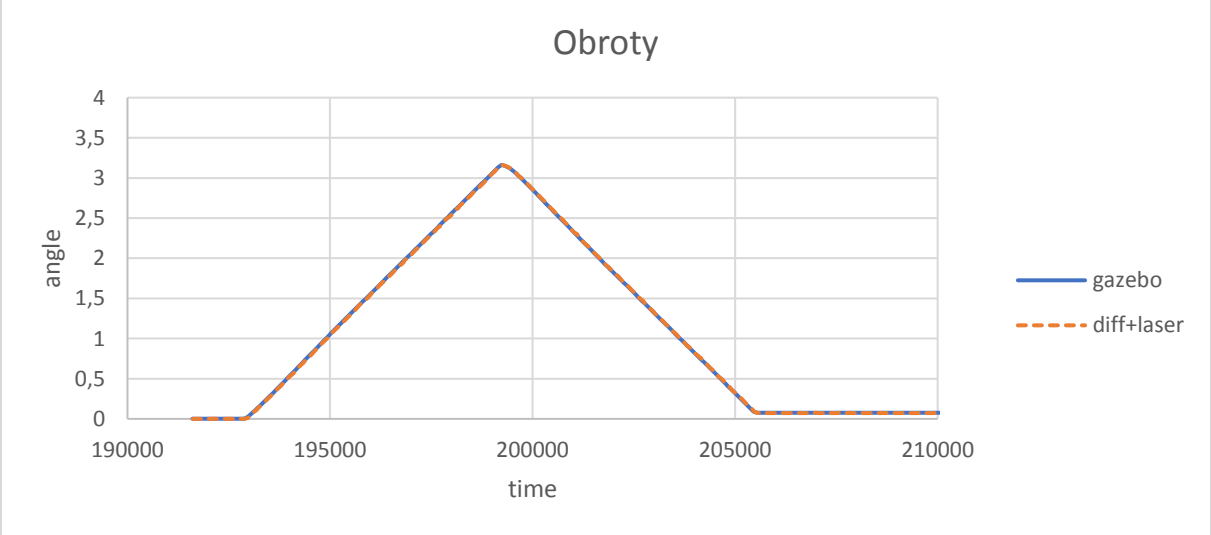
Wyniki diff_controller+laser

Jazda przód – tył



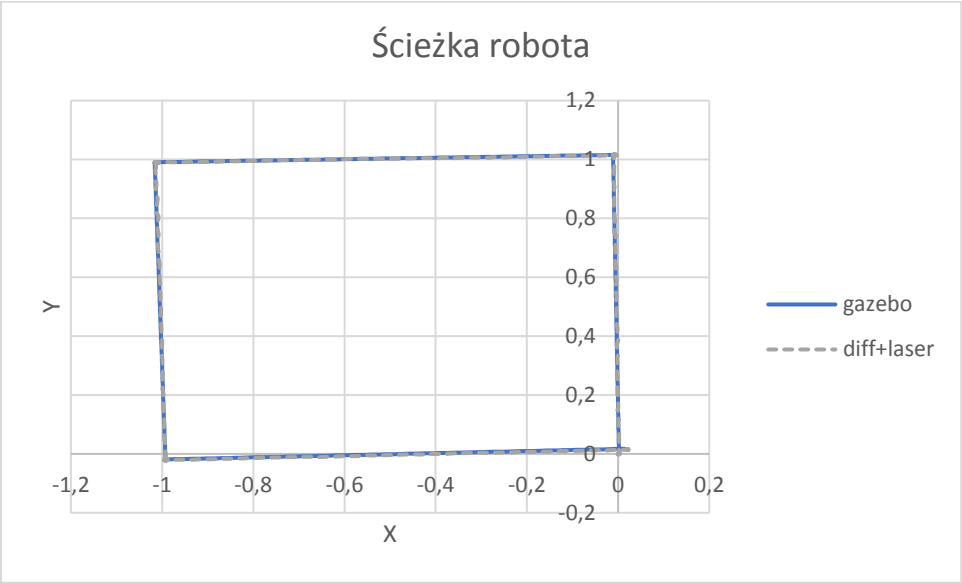
Błędy				
	1	2	3	4
X_gazebo	-0,99858	-0,99999	-0,99923	-0,99979
X_diff+las	-1,00166	-1,00075	-0,99794	-0,99668
dX	0,003073	0,000755	0,001294	0,003111

Obroty



Błędy				
	1	**2**	**3**	**4**
Th_gazebo	3,161624	3,706281	3,706726	3,707141
Th_laser	3,159184	3,704644	3,703258	3,679063
dTh	0,00244	0,001637	0,003468	0,028078

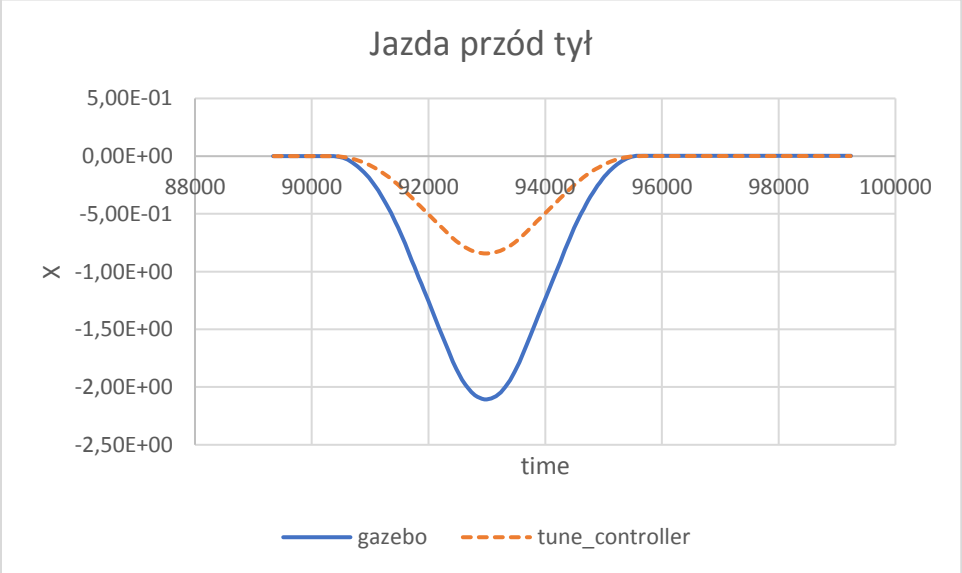
Test kwadratu



Błędy				
	1	**2**	**3**	**4**
dX	0,005962	0,002991	0,000493	0,003698
dY	0,0021	0,006199	0,00109	0,00441
dTh	0,002981	0,008954	0,015812	0,02094

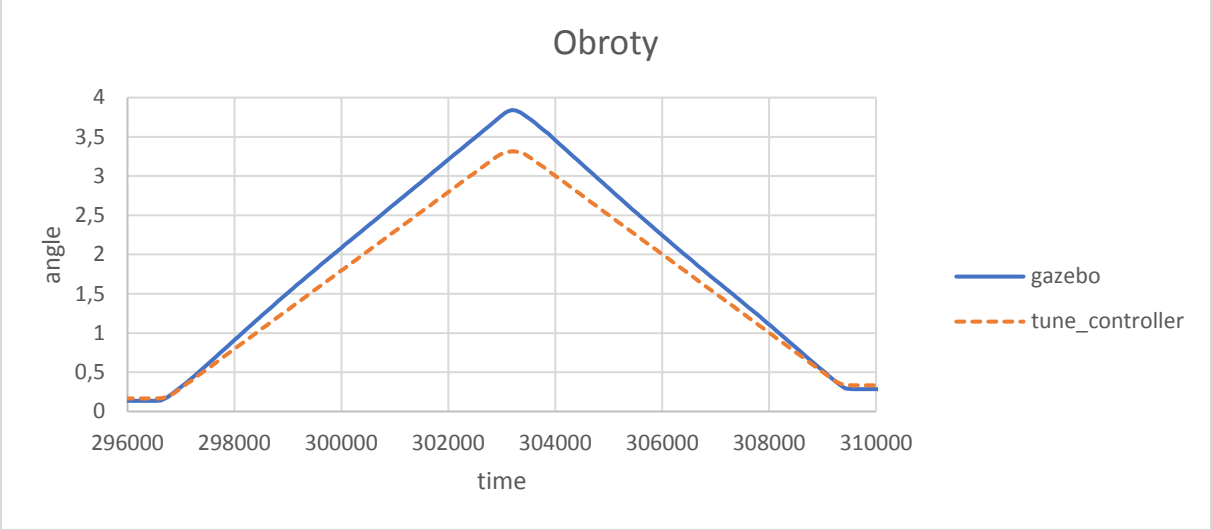
Wyniki tune_controller

Jazda przód - tył



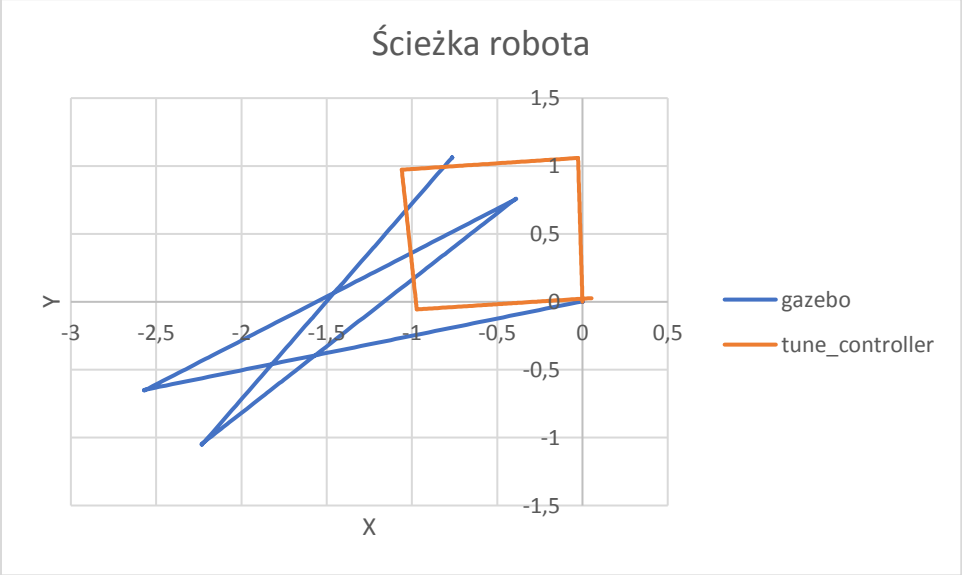
Błędy				
	1	**2**	**3**	**4**
X_gazebo	-2,10687	-2,10816	-2,10856	-2,10892
X_tune	-0,84255	-0,84118	-0,8435	-0,84332
dX	1,264319	1,266986	1,265062	1,265596

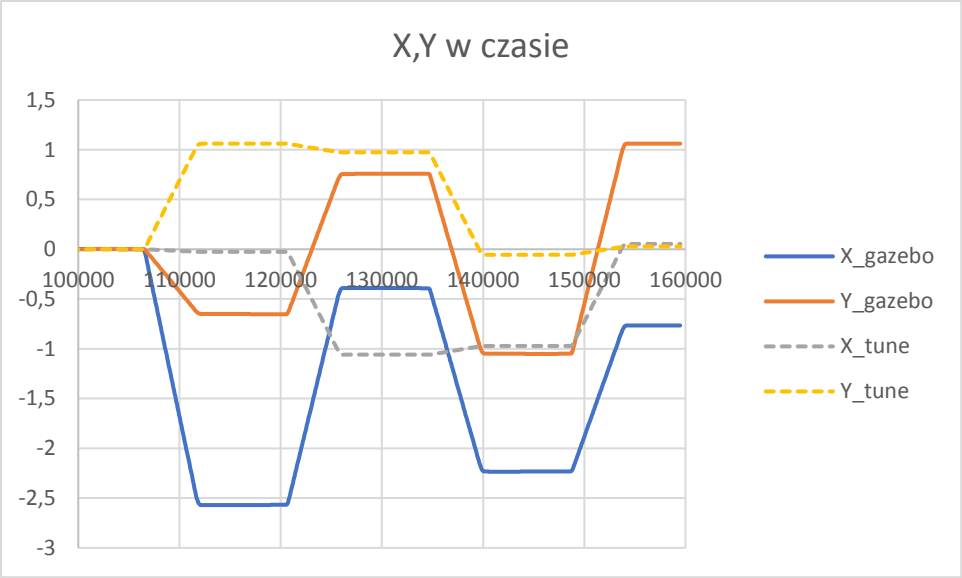
Obroty



Błędy				
	1	**2**	**3**	**4**
Th_gazebo	3,7062	3,706281	3,706726	3,707141
Th_tune	3,150032	3,149678	3,148793	3,149489
dTh	0,556168	0,556603	0,557933	0,557652

Kwadrat



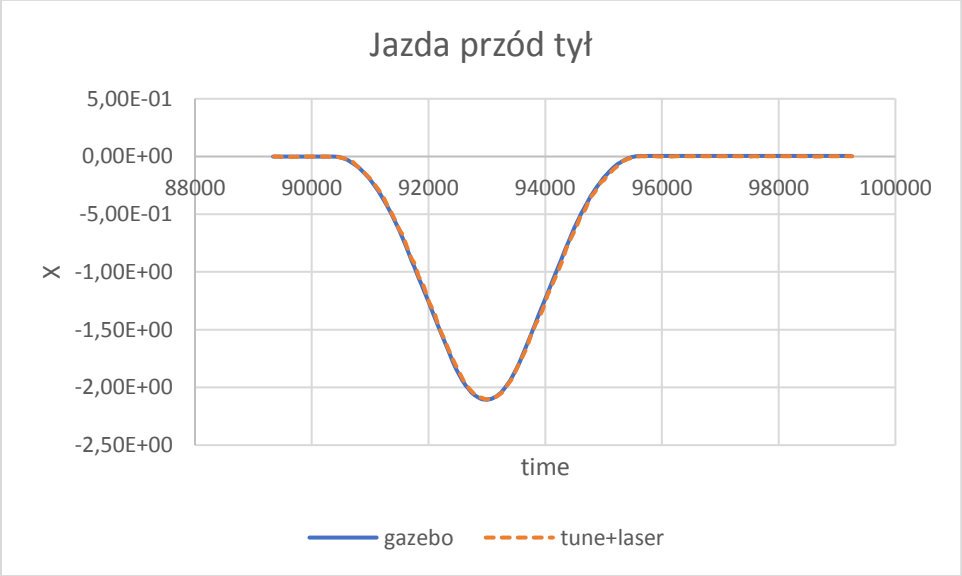


Błędy odometrii na koniec ruchu wzdłuż osi X, Y oraz błąd kąta Th zebrano w tabeli:

Błędy				
	1	**2**	**3**	**4**
dX	0,818299	0,768029	0,816031	0,787381
dY	1,035247	1,039316	1,038314	1,024797
dTh	0,883478	0,86411	0,887171	0,878886

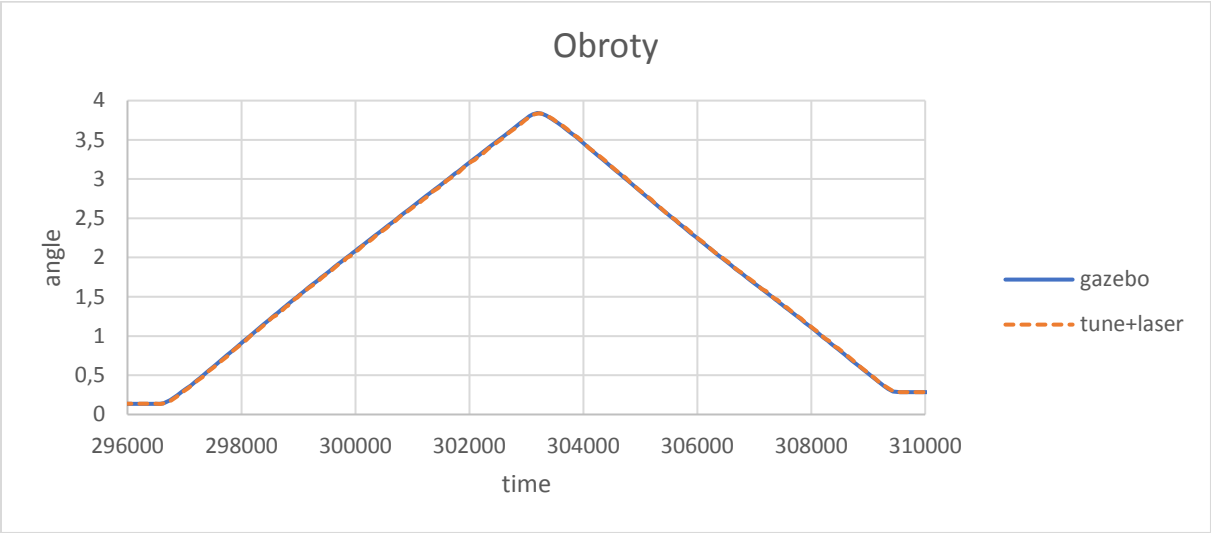
Wyniki tune_controller+laser

Jazda przód - tył



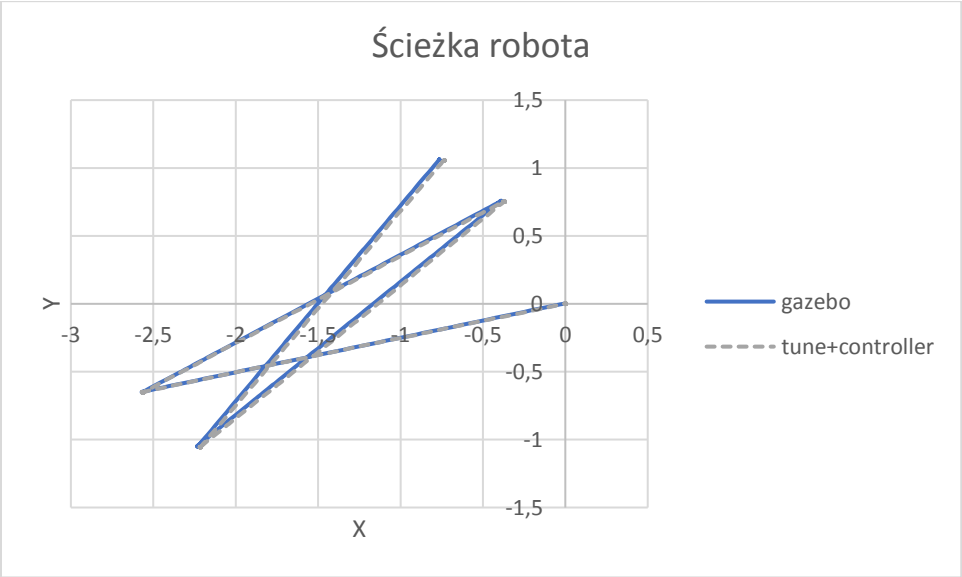
Błędy				
	1	2	3	4
X_gazebo	-2,10687	-2,10816	-2,10856	-2,10892
X_laser	-2,10071	-2,10482	-2,1137	-2,10998
dX	0,006162	0,003345	0,005139	0,001061

Obroty



Błędy				
	1	**2**	**3**	**4**
Th_gazebo	3,7062	3,706281	3,706726	3,707141
Th_laser	3,702424	3,704644	3,703258	3,679063
dTh	0,003776	0,001637	0,003468	0,028078

Kwadrat



Błędy				
	1	**2**	**3**	**4**
dX	0,031508	0,009387	0,007873	0,020679
dY	0,005756	0,001392	0,004072	4,37E-06
dTh	0,004725	0,002066	0,003516	0,003057

Wnioski

Trzy konfiguracje: diff_controller, diff_controller + laser, tune_controller + laser dają błędy rzędu 0,1-0,3%. Lokalizacja robota przy ich użyciu jest bardzo dokładna, robot w zadowalający sposób wykonuje zadane ścieżki. Potwierdzają to wykresy z nałożoną trajektorią referencyjną (gazebo odom). Użycie tune_controller powoduje znaczne zwiększenie błędów – rzędu 50-70%. Ścieżka (dobrze widoczne jest to w przypadku testu kwadratu) zupełnie nie pokrywa się ze ścieżką zadaną. Należy zatem podjąć próbę kalibracji tune_controller.

3.6. Sposób kalibracji sterownika tune_controller

3.6.1. Przedstawienie sposobu kalibracji

Parametry, które można było poddać kalibracji to promień kół robota oraz rozstaw kół. Parametry zmieniano i poddawano robiąc testy – jazdy przód-tył oraz obrotów. Testy przebiegały podobnie jak w punkcie 3.5.

3.6.2. Podjęte kroki

1. Analiza danych z punktu 3.5. dla testu jazdy przód-tył. Robot pokonywał zbyt duże odległości. Wniosek: promień koła ustawiono na zbyt dużą wartość, gdyż przebyta odległość jest wprost proporcjonalna do promienia koła.
2. Zmniejszanie promienia koła, aż do momentu osiągnięcia zadowalających wyników.
3. Analiza danych z punktu 3.5. dla testu obrotu. Robot wykonywał obroty o zbyt duży kąt.
4. Sprawdzenie w jaki sposób zmiana rozstawu kół wpływa na test obrotu. Wniosek: zwiększenie rozstawu kół zmniejsza kąt o jaki obraca się robot.
5. Zmniejszanie rozstawu kół aż do momentu uzyskania zadowalających wyników.

Zmieniano parametry:

- **wheel_radius_multiplier**
- **wheel_separation_multiplier**

Dobrane parametry:

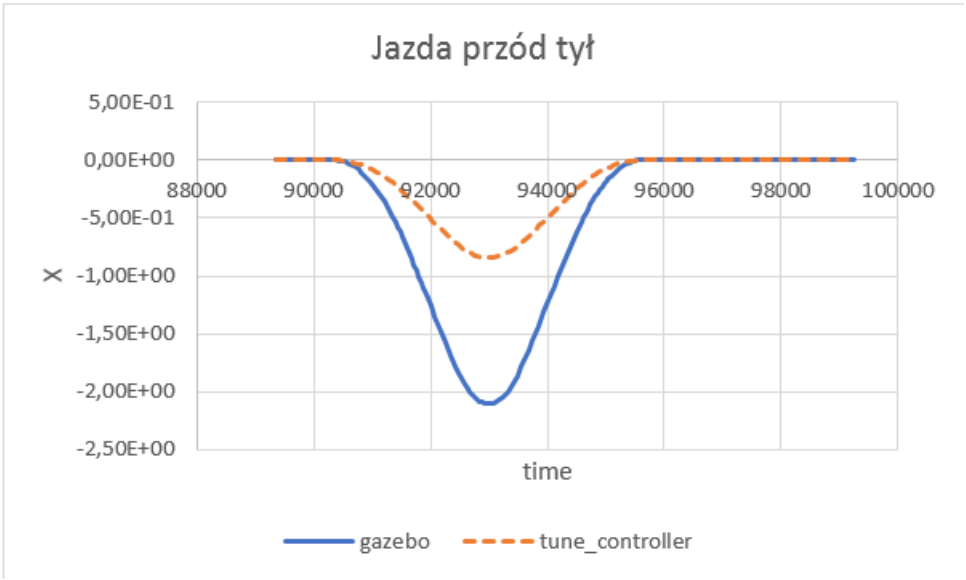
wheel_separation : 0.285 **wheel_radius** : 0.02	**wheel_separation_multiplier:** 1.1 # default: 1.0 **wheel_radius_multiplier :** 2.4 # default: 1.0

3.6.3. Weryfikacja kalibracji

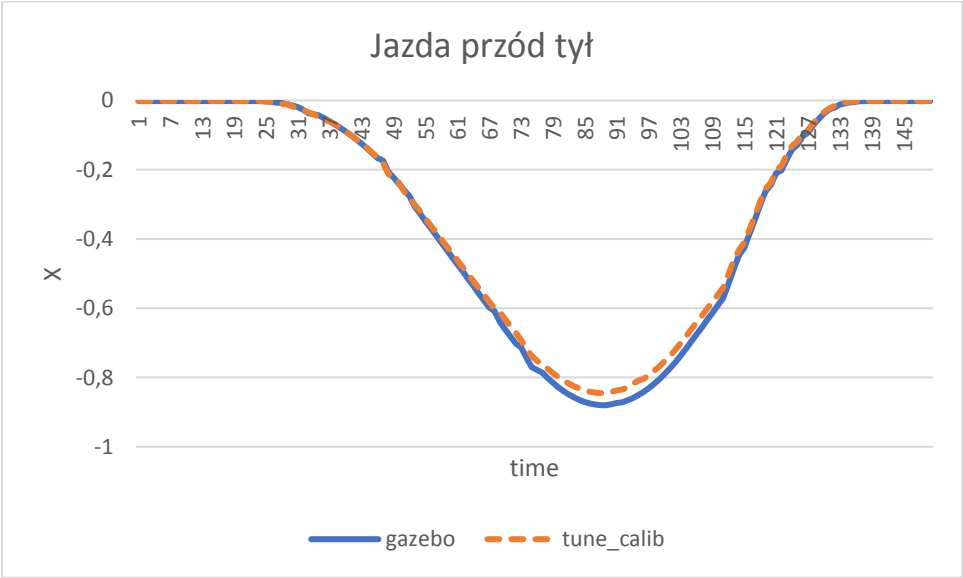
Poniżej zaprezentowano wykresy położenia robota od czasu oraz ścieżkę jazdy po kwadracie dla kontrolera przed i po kalibracji:

Jazda przód-tył

Przed kalibracją:

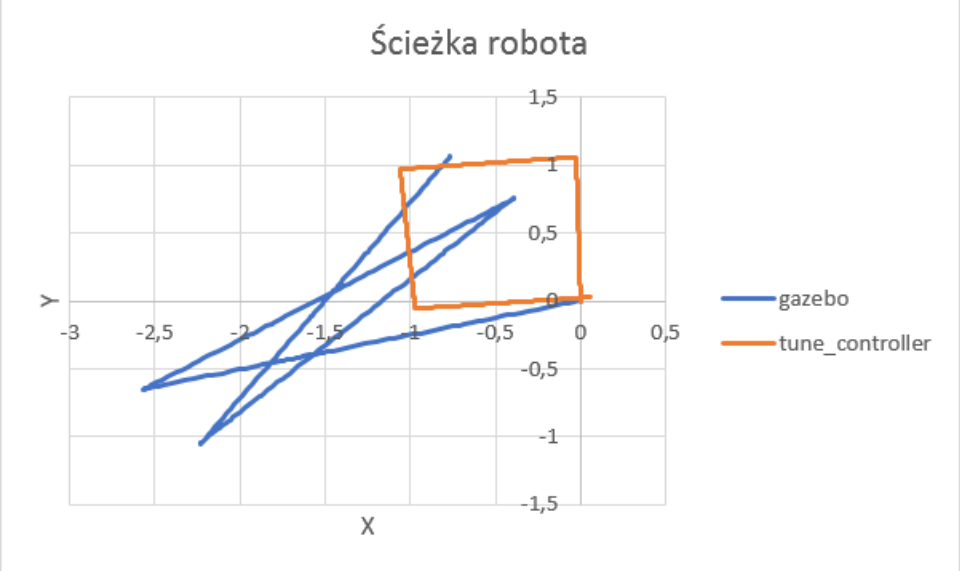


Po kalibracji:

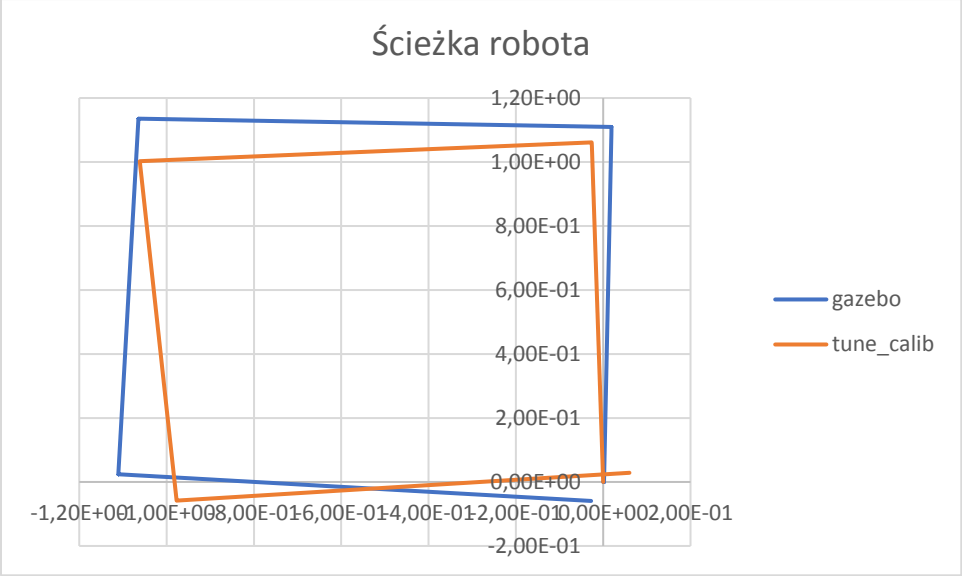


Kwadrat

Przed kalibracją

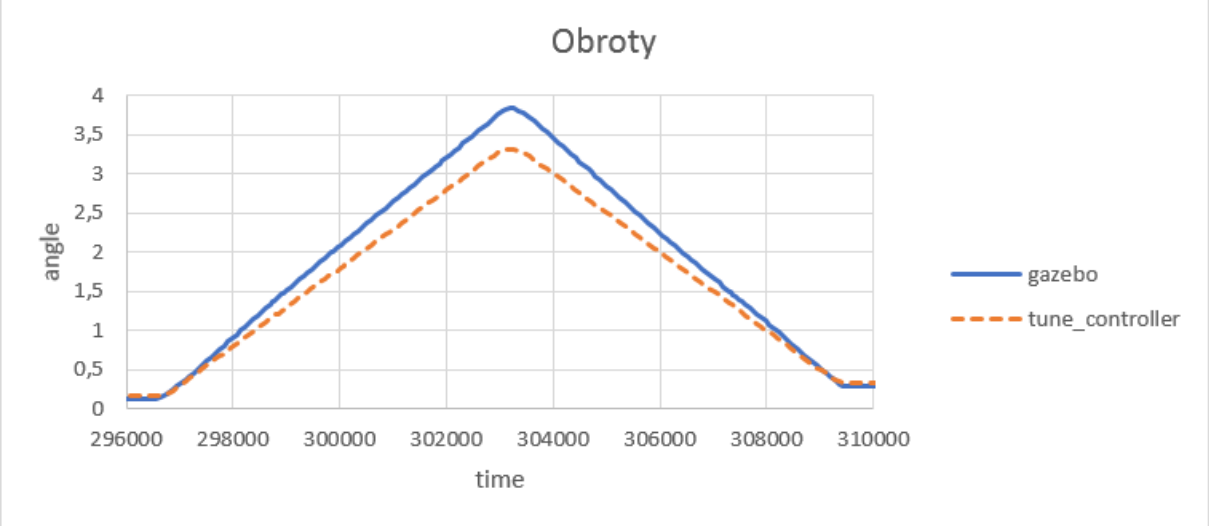


Po kalibracji:

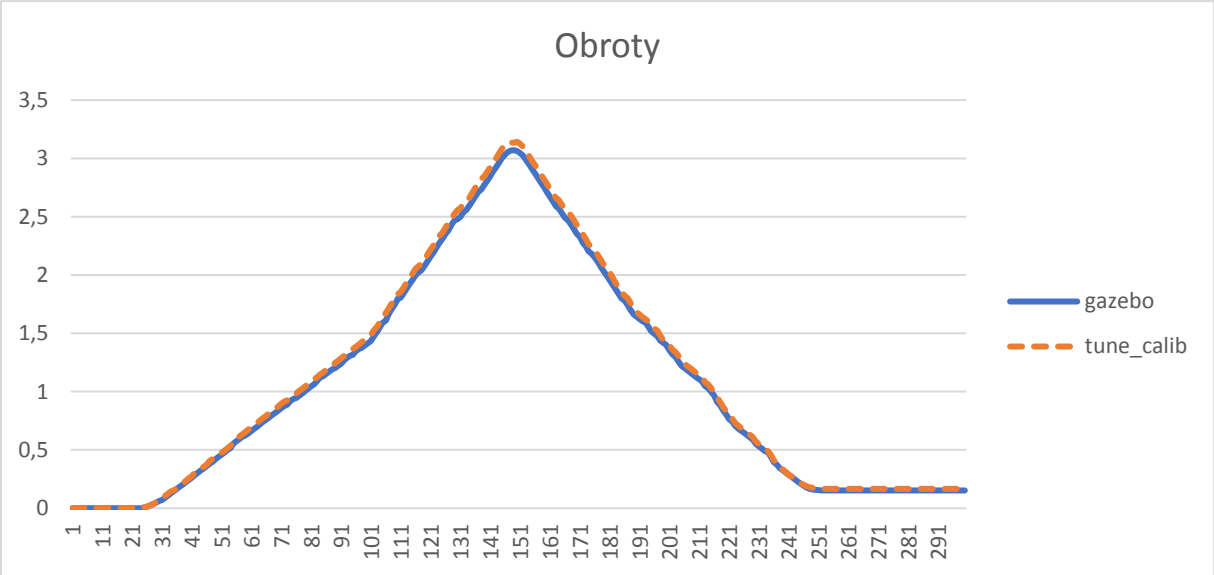


Obroty

Przed kalibracją



Po kalibracji:



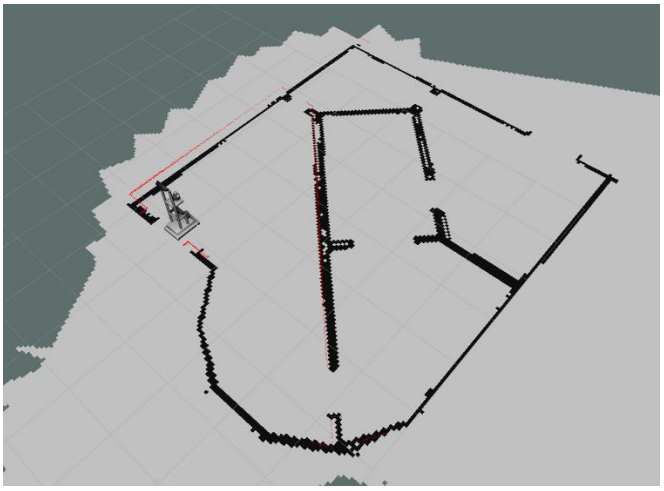
Wnioski

Wykresy ewidentnie wskazują na poprawę jakości sterowania robotem po kalibracji kontrolera. Steruje on robotem zadowalająco, jednak jest znacząco mniej dokładny niż drugi testowany – diff_controller.

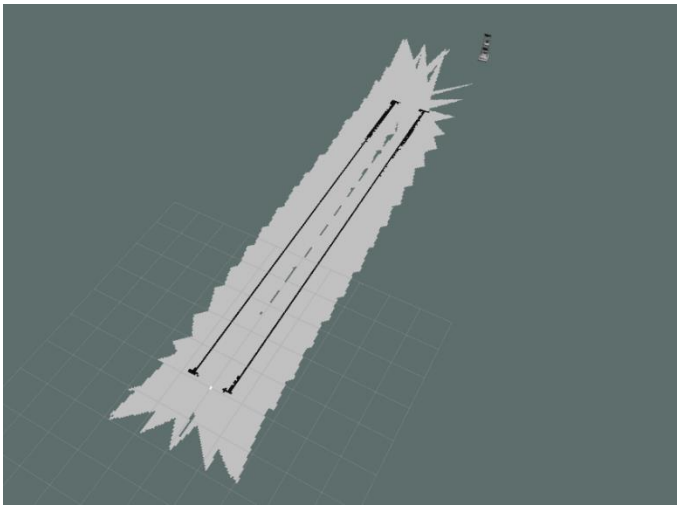
4. Laboratorium 2.

4.1. Stworzone środowiska i ich mapy

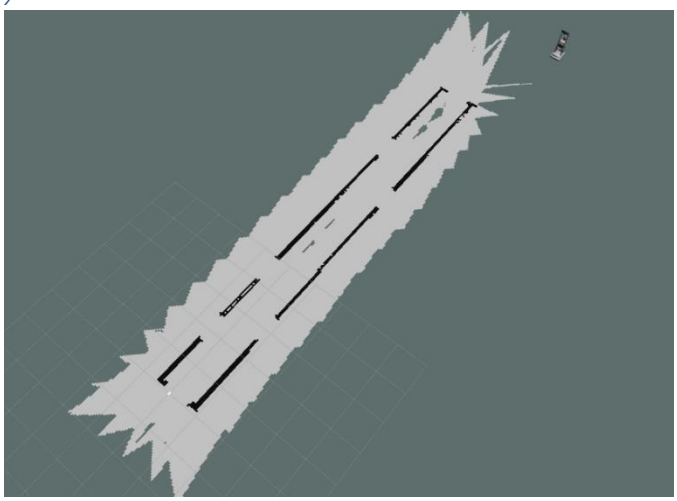
Środowisko do testowania systemu nawigacji o różnej szerokości przejść



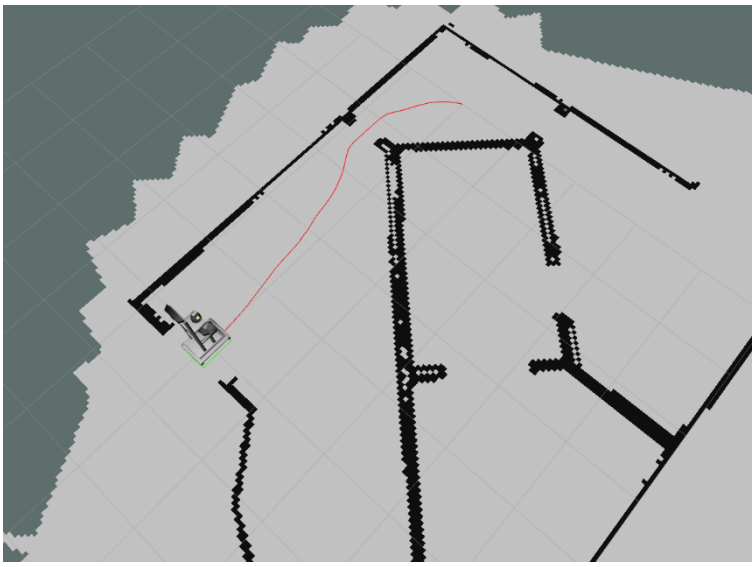
Wąski korytarz

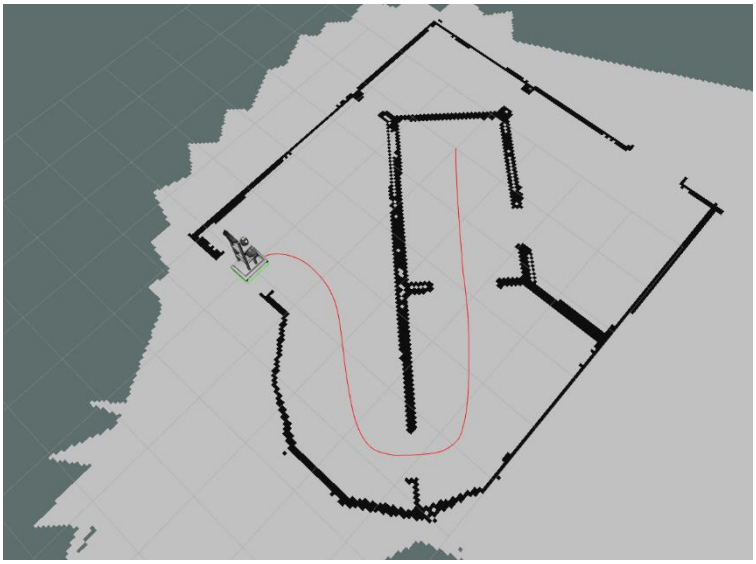


Korytarz z asymetrycznymi drzwiami



4.2. Przykładowe ścieżki zaplanowane w środowiskach





4.3. Pliki uruchomieniowe symulacji

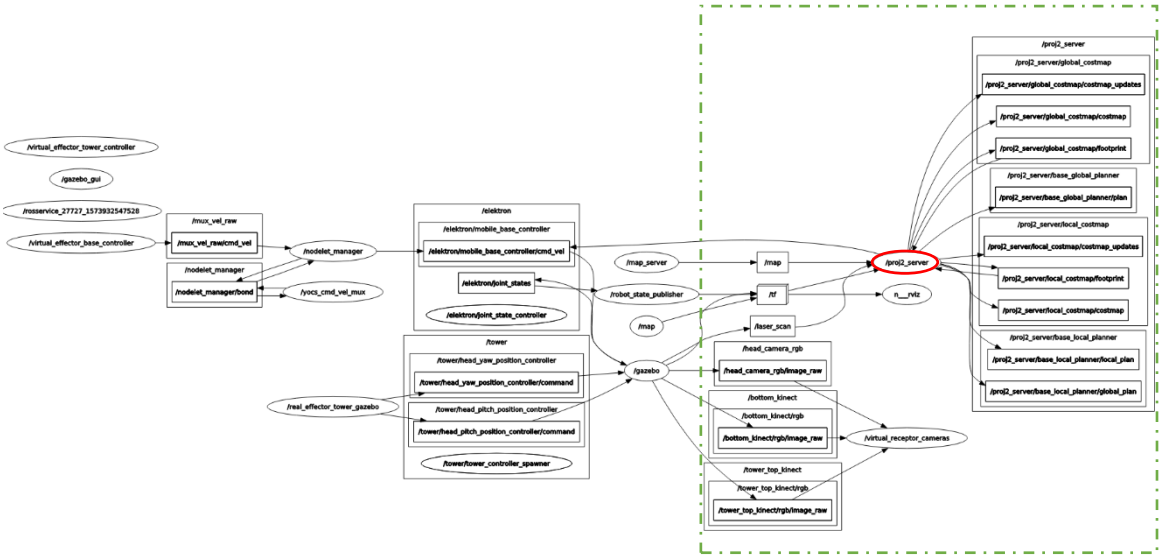
Przy uruchamianiu symulacji ładowano mapy na serwer map, tworzymy statyczną transformację między ramkami map i odom (nie są względem siebie przesunięte, więc parametry ustawiano na same 0), uruchomiano node'a 'Lab2" z argumentem – końcem ścieżki .

Węzeł 'Lab2' służy do realizacji ścieżki zaplanowanej w środowisku. Korzysta z planera, który pobiera odpowiednią mapę z serwera map.

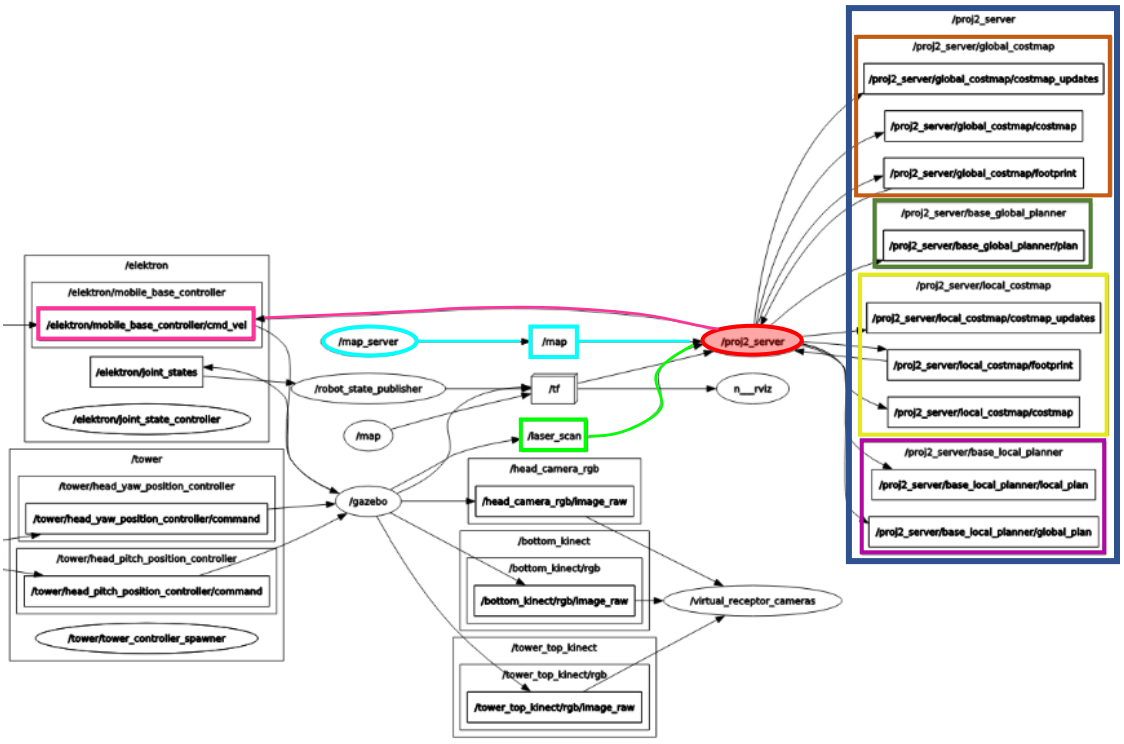
Zaplanowana ścieżka posiadała zbyt wiele punktów, dlatego aby robot jeździł płynnie wybierano co 10 punkt. Zawsze zostawiano ostatni punkt ścieżki. Robot osiągał kolejne pozycje przy użyciu serwisu z Laboratorium 1 – 'path_follower'

5. Projekt 2.

5.1.Struktura sterownika robota

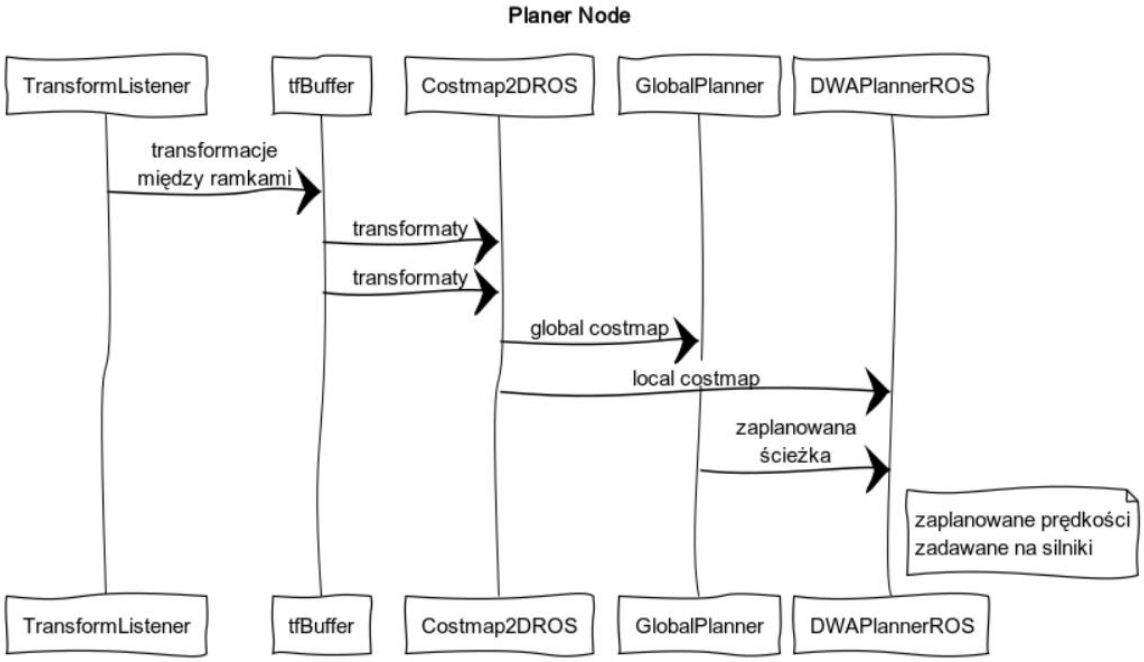


Na powyższym schemacie zaznaczono **sekcję** obrazującą strukturę sterownika robota, a w niej **węzeł planujący** 'proj2_server'. Poniżej opisano poszczególne elementy struktury sterownika bardziej szczegółowo.



Sekcja planowania składa się z czterech podsekcji – **planera globalnego**, który korzysta z **globalnej mapy kosztów** oraz **planera lokalnego**, który korzysta z **lokalnej mapy kosztów**. Obydwie mapy kosztów pobierają z **węzła planującego** ‘proj2_server’ dane o wymiarach robota (footprint). Dodatkowo globalna mapa kosztów jest tworzona na podstawie **mapy statycznej** udostępnianej przez **serwer map**, a lokalna mapa kosztów na podstawie odczytów z **czujnika (lasera)**. Wyznaczone prędkości węzeł planujący **zadaje na silniki robota**.

5.2. Opis działania węzła planującego



Algorytm działania węzła planującego:

1. Stworzenie bufora transformat między odpowiednimi układami.
2. Stworzenie globalnej mapy kosztów
3. Stworzenie lokalnej mapy kosztów
4. Stworzenie globalnego planera
5. Stworzenie lokalnego planera
6. Po wywołaniu serwisu ze współrzędnymi celu rozpoczyna się algorytm nawigacji:
    6.1. Stworzenie planu globalnego
    6.2. Włączenie planera lokalnego
    6.3. Publikowanie planu globalnego, aby można było dokonać wizualizacji zaplanowanej ścieżki
    6.4. Sprawdzenie czy planerowi lokalnemu udało się wyznaczyć kolejne prędkości
        6.4.1. Jeśli **tak** prędkości są zadawane na silniki robota
        6.4.2. Jeśli **nie** robot rozpoczyna algorytm recovery
    6.5. Sprawdzenie czy robot osiągnął cel
        6.5.1. Jeśli **tak** – koniec algorytmu
        6.5.2. Jeśli **nie** – powrót do punktu 6.4.

Algorytm recovery:

1. Wyczyszczenie lokalnej mapy kosztów
2. Wykonanie obrotu w miejscu o pewien kąt
3. Cofnięcie się o daną odległość
4. Sprawdzenie czy lokalny planer wyznacza kolejne prędkości
    4.1. Jeśli **tak** koniec algorytmu
    4.2. Jeśli **nie** powrót do punktu 1.

5.3. Pliki konfiguracyjne map kosztów oraz lokalnego planera

Lokalna mapa kosztów

```
local_costmap: # parametry lokalnej mapy kosztow
  global_frame: odom # uklad wspolrzednych, wzgledem ktorego dodawane jest polozenie mapy kosztow
  robot_base_frame: base_link # uklad wspolrzednych bazy robota
  update_frequency: 5.0 # czestotliwosc odswiezania mapy
  publish_frequency: 2.0 # czestotliwosc publikowania mapy na topic
  static_map: false # mapa nie jest statyczna, pochodzi z odczytow
  rolling_window: true # czy poczatek ukladu wsp. mapy jest srodkiem robota (tak)
  width: 6.0 # szerokosc mapy-okna
  height: 6.0 # wysokosc mapy-okna
  resolution: 0.05 # rozdzielczosc mapy (w metrach/piksel)
  plugins:
    # - {name: static_map,       type: "costmap_2d::StaticLayer"}
    - {name: obstacle_map,        type: "costmap_2d::VoxelLayer"}
    - {name: inflation_map,        type: "costmap_2d::InflationLayer"}
  obstacle_map:
    observation_sources: laser_scan_sensor # zrodlo odczytow danych obserwacyjnych do tworzenia warstwy przeszkod
    laser_scan_sensor: {
      observation_persistence: 0.1,
      sensor_frame: base_laser_link,
      data_type: LaserScan,
      topic: /laser_scan,
      # expected_update_rate: 0.1,
      track_unknown_space: true,
      marking: true,
      clearing: true,
      min_obstacle_height: -5,
      max_obstacle_height: 5,
      obstacle_range: 2.5, # max odleglosc odczytu lasera
      raytrace_range: 3.0 # max odleglosc uznana za pusta miedzy laserem a odczytem

      } # definicja czujnika
    track_unknown_space: true
  inflation_map:
    cost_scaling_factor: 10
    inflation_radius: 2
```

Globalna mapa kosztów

```
global_costmap: # parametry mapy globalnej
  global_frame: map # uklad wspolrzednych mapy
  robot_base_frame: base_link # uklad wspolrzednych bazy robota
  update_frequency: 10 # czestotliwosc odswiezania (niewwykorzystywana)
  publish_frequency: 2.0 # czestotliwosc publikowania mapy na topic
  static_map: true # mapa nie jest statyczna, pochodzi z odczytow
  rolling_window: false # czy poczatek ukladu wsp. mapy jest srodkiem robota (tak)
  plugins:
    - {name: static_map,       type: "costmap_2d::StaticLayer"}
    - {name: inflation_map,        type: "costmap_2d::InflationLayer"}
  inflation_map:
    cost_scaling_factor: 10
    inflation_radius: 2
```

Planer lokalny

```
base_local_planner:
  # ilość próbek prędkości
  vx_samples: 50
  vy_samples: 1
  vth_samples: 41
  # granice predkosci robota
  max_vel_x: 0.15
  min_vel_x: 0.03
  max_vel_y: 0
  min_vel_y: 0
  max_vel_th: 1.7
  min_vel_th: -1.7
  # parametry algorytmu planera
  sim_time: 6
  path_distance_bias: 1
  goal_distance_bias: 0.5
  occdist_scale: 0.005
  # granice przyspieszen robota
  acc_lim_th: 8
  acc_lim_trans: 3
  # tolerancja osiągnięcia celu
  yaw_goal_tolerance: 1000
  xy_goal_tolerance: 0.1
  # czy robot jest holonomiczny
  holonomic_robot: false
```

5.4. Wyjaśnienie zastosowanych parametrów

Większość ustawionych parametrów jest wyjaśniona poprzez komentarz w kodzie. Poniżej przedstawiono parametry, których dobór związany był z jakością działania systemu nawigacji.

Lokalna mapa kosztów:

```
  inflation_map:
    cost_scaling_factor: 10
    inflation_radius: 2
```

- inflation_radius – odległość od rzeczywistej przeszkody w której istnieje przestrzeń o niezerowym koszcie. Jednym z parametrów algorytmu użytego planera, była minimalizacja kosztów. Zwiększenie tego parametru powodowało jazdę robota w większej odległości od przeszkód. Parametr musi zostać tak dobrany, aby robot nie jeździł zbyt blisko przeszkód.
- cost_scaling_factor – gradient kosztów w obszarze inflation_radius. Im wyższa wartość parametru tym gradient jest bardziej stromy. Ustawienie go na wartość 10, powoduje, że koszt jest wysoki jedynie blisko przeszkód (kolor czerwony rviz) i gwałtownie spada. Większość obszaru inflation_radius ma podobny, niski koszt (kolor niebieski rviz). Szczególnie w wąskich przejściach koszt musiał być odpowiednio niski w odpowiednio szerokim obszarze. W innym wypadku robot zatrzymywał się przed przejściem.

Globalna mapa kosztów:

Parametry inflation_radius, cost_scaling_factor zostały ustawione tak samo jak w mapie lokalnej. W innym przypadku system nawigacji działał niepoprawnie – w skrajnym przypadku planer globalny planował ścieżkę niemożliwą do osiągnięcia dla planera lokalnego.

Planer lokalny:

- *ilości próbek prędkości* stosowane w algorytmi DWA do przeszukiwania możliwych zadanych prędkości zostały dobrane metodą prób i błędów. W szczególności ilość próbek prędkości y została ustawiona na 1, gdyż jest ona zawsze zerowa (wyjaśnienie poniżej).
- *max_vel_y, min_vel_y* zostały ustawione na 0. Uniemożliwia to planerowi zadawania prędkości wyłącznie w osi y. Robot jest nieholonomiczny i nie może wykonać takiego polecenia.
- *max_vel_x, min_vel_x, max_vel_th, min_vel_th* zostały dobrane tak, aby planer nie zadawał zbyt małych prędkości – powodowało to przestoje w ruchu, a jednocześnie, żeby ruch nie odbywał się zbyt szybko (ograniczeni prędkościowe robota wynikające z konstrukcji)
- *parametry algorytmu planera* były kluczowymi parametrami, które należało dostroić aby robot poruszał się zgodnie z oczekiwaniami:
    - *sim time* – czas symulowania ścieżki. Ustalenie zbyt małej wartości powodowało częste zakleszczenia. Zbyt długi czas symulacji powodował problem ze znalezieniem ścieżki i jej gubienie.
    - *path_distance_bias* – parametr określający jak istotne ma być trzymanie się ścieżki planera globalnego w algorytmie planowania ścieżki. Kiedy planer globalny został odpowiednio skonfigurowany (planował sensowne ścieżki) robot był w stanie jeździć jedynie na tym parametrze (inne 0), dlatego ma najwyższą wagę w algorytmie.
    - *goal_distance_bias* – parametr określający jak istotne ma być zbliżanie się do celu przez robota w algorytmie planowania ścieżki.. Gdy był ustawiony na zbyt wysoką wartość, robot przy specyficznych ścieżkach (ścieżka oddalała się od celu) gubił ścieżkę i zawracał.
    - *occdist_scale* – parametr określający jak istotne ma być omijanie przestrzeni o niezerowym koszcie w algorytmie planowania ścieżki. Wwiększenie tego parametru uniemożliwia robotowi przejeżdżanie przez wąskie przejścia.

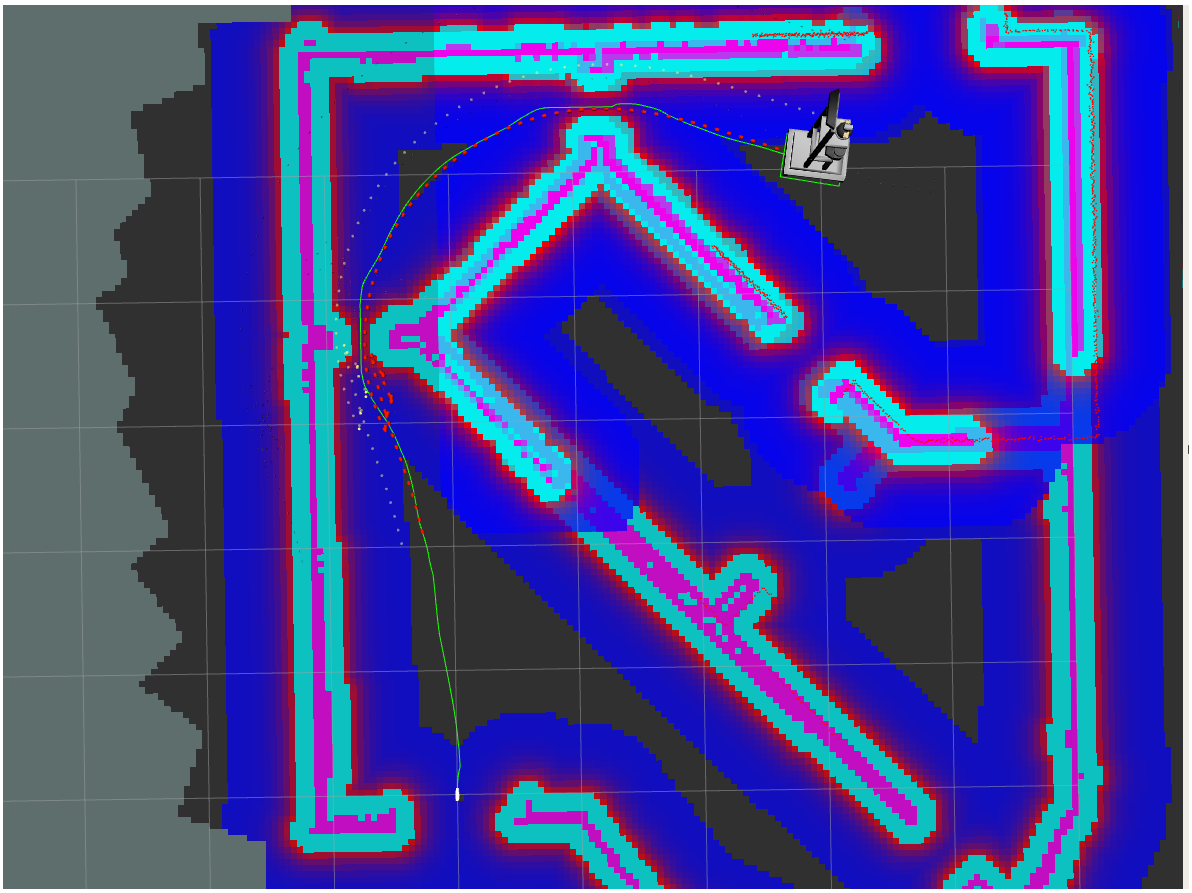
5.5. Weryfikacja działania

Zielona linia to ścieżka zaplanowana przez planer globalny. Czerwone strzałki pokazują pozycje które przyjmował robot, w trakcie wykonywania ruchu. Szare kropki pokazują położenie czubka robota.

Wnioski

Robot osiągnął zadaną pozycję. Poruszał się po zaplanowanej ścieżce.

Zielona linia to ścieżka zaplanowana przez planer globalny. Czerwone punkty pokazują położenia które przyjmował robot, w trakcie wykonywania ruchu. Szare kropki pokazują położenie czubka robota.



Wnioski

Przy przejeździe przez pierwsze, wąskie przejście sterownik robota użył algorytmu recovery – widać to przez nagromadzenie punktów przed przejazdem. Po kilku próbach planerowi lokalnemu udało się zaplanować odpowiednią ścieżkę i robot przejechał przez drzwi.