

Sprawozdanie

STERO

Lab 1

Maciej Groszyk, Mateusz Zembroń

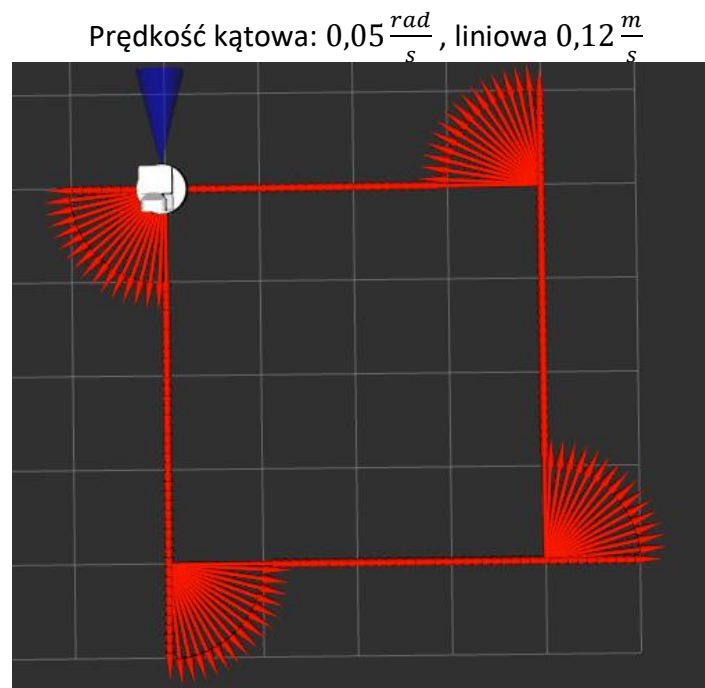
Interpolacja liniowa sterowania pozycyjnego na podstawie wyłącznie zadawanych prędkości

- Program („sq_publisher.py”)

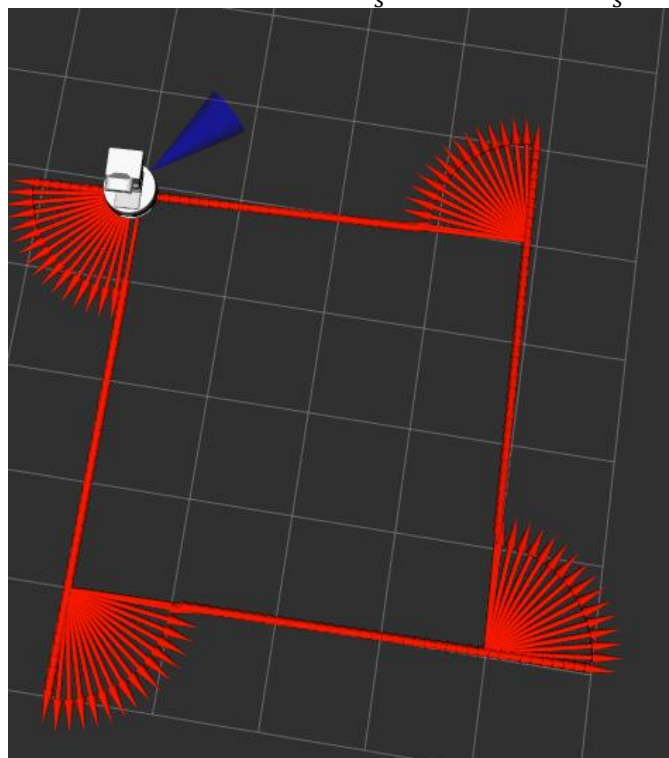
Napisany program zadawał prędkość naprzemiennie liniową oraz kątową w taki sposób aby uzyskać ruch robota po kwadracie. W tej wersji prędkość była zadawana niezależnie od tego gdzie znajdował się robot (czas ruchu w każdym etapie był założony z góry). Dane były publikowane na temacie „key_vel” przez stworzony węzeł o nazwie „stero_vel_pub”.

- Testy

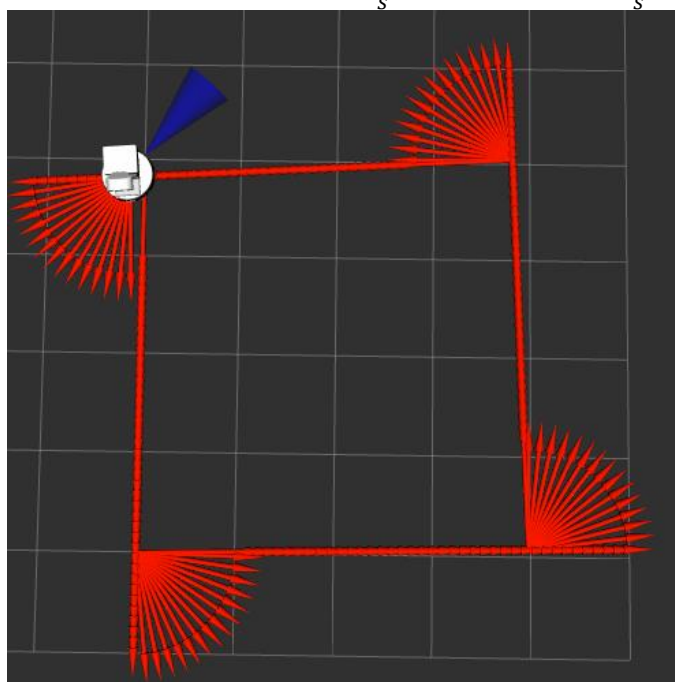
Program sprawdzono dla różnych zestawów prędkości (liniowych i kątowych). Efekty działania programu można zaobserwować m.in. w programie symulacyjnym RViz:



Prędkość kątowa: $0,12 \frac{rad}{s}$, liniowa $0,288 \frac{m}{s}$

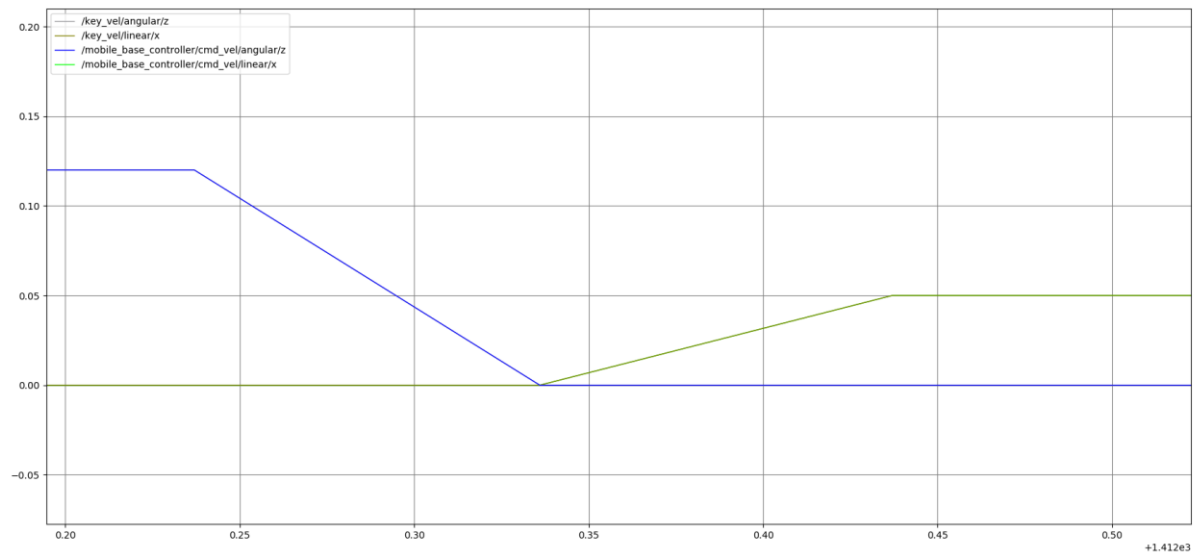


Prędkość kątowa: $0,288 \frac{rad}{s}$, liniowa $0.6912 \frac{m}{s}$

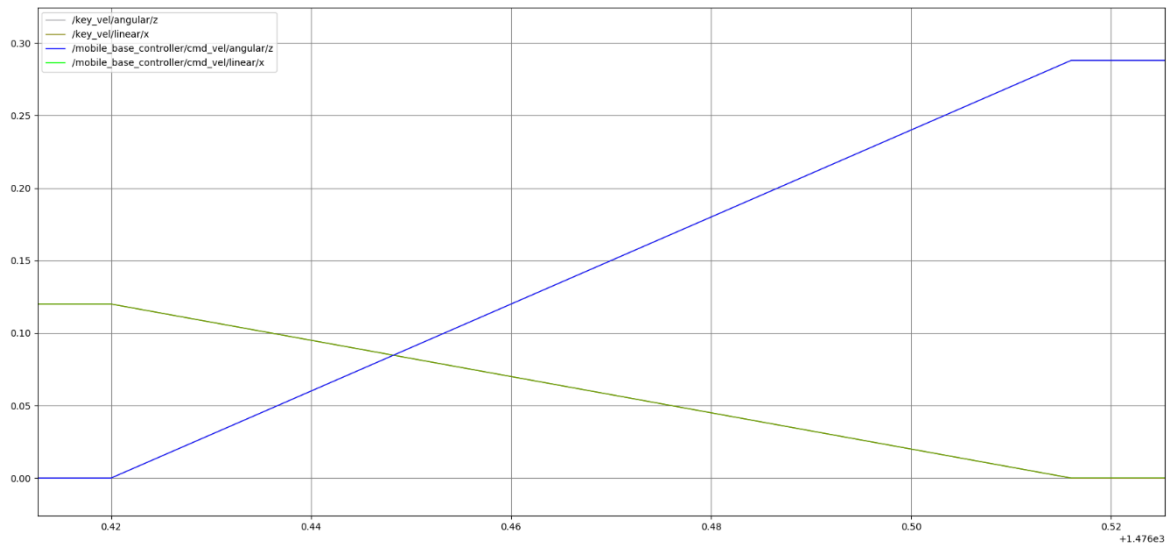


Sprawdzono również zgodność prędkości zadanej w programie z prędkością symulowanego robota. Poniżej zamieszczono przebiegi tych wartości podczas zmiany trybu ruchu robota z jazdy prosto na obrót:

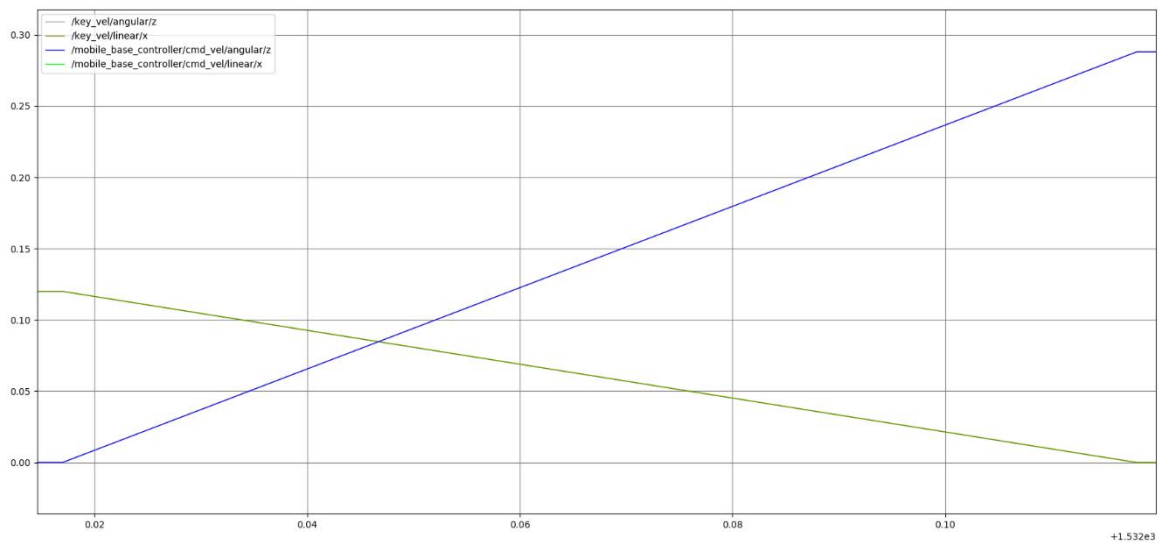
Prędkość kątowna: $0,05 \frac{rad}{s}$, liniowa $0,12 \frac{m}{s}$



Prędkość kątowna: $0,12 \frac{rad}{s}$, liniowa $0,288 \frac{m}{s}$



Prędkość kątowna: $0,288 \frac{rad}{s}$, liniowa $0.6912 \frac{m}{s}$



Uwaga!

Na przebiegu oznaczono kolorem:

- szarym – zadaną prędkość kątową
- ciemnozielonym – zadaną prędkość liniową
- niebieskim – odczytaną liniową kątową robota
- jasnozielonym – odczytaną liniową prędkość robota

! Na zdjęciach widoczne są jedynie dwa przebiegi, ponieważ prędkości zadane i odczytane zgadzały się ze sobą z dużą dokładnością!

Zgodnie z symulacjami, w trybie interpolacji liniowej punktów na podstawie wyłącznie zadawanej prędkości, robot poruszał się z bardzo dużą dokładnością zgodnie z jego zamierzoną trajektorią. Jednak taka forma ruchu jest mało odporna na jakiegokolwiek zmiany środowiska – brak sprzężenia zwrotnego położenia robota.

Interpolacja liniowa sterowania pozycyjnego na podstawie danych z odometrii

- Program („*odometry_node.py*”)

Napisany program opiera się o węzeł „*stero_listener*” odczytujący aktualną pozycję robota (poprzez subskrypcję na temacie „*mobile_base_controller/odom*”). Zaimplementowany algorytm decyduje o zadaniu prędkości kątowej lub liniowej w zależności od odczytanej odległości od ostatniego rogu interpolowanego kwadratu. Następnie prędkość zadawana jest robotowi poprzez publikację danych na temacie „*key_vel*”.

```

sequenceDiagram
    participant A as /mobile_base_controller/odom
    participant B as /stereo_listener_10603_1635337923486
    participant C as /mobile_base_controller/cmd_vel
    participant D as /twist_mux
    participant E as /mobile_base_controller/cmd_vel
    participant F as /twist_marker

    A->>B: /key_vel
    B->>C: /mobile_base_controller/cmd_vel
    C->>D: /twist_mux
    D->>E: /mobile_base_controller/cmd_vel
    E->>F: /twist_marker

    A->>G: /head_controller/follow_joint_trajectory/status
    Note over G: (off-diagram)
  
```

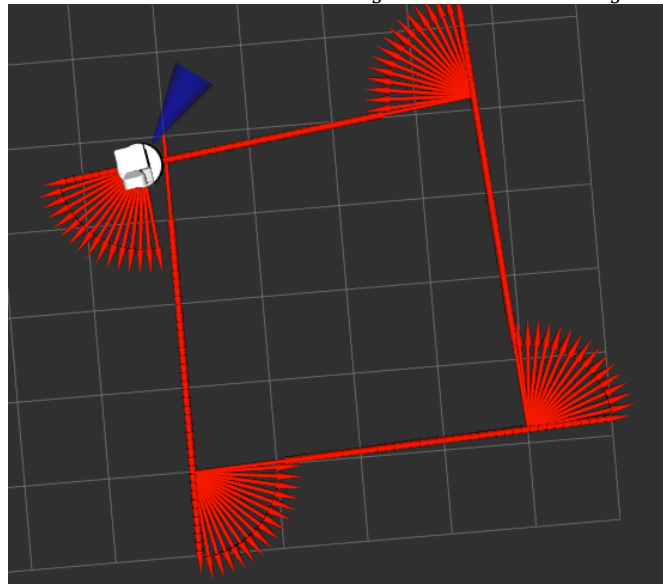
The diagram illustrates a ROS2 system architecture with the following components and dependencies:

- Nodes (Ovals):**
 - `/default_controllers_spawner`
 - `/bringup_controllers_spawner_imu`
 - `/gazebo`
 - `/xrtion`
 - `/xrtion/rgb/image_proc`
 - `/robot_state_publisher`
 - `/head_controller`
 - `/play_motion`
 - `/trajectory_execution_event`
 - `/move_group`
 - `/twist_mux`
 - `/twist_marker`
 - `/fs_already_there`
 - `/image_raw_to_rect_color_relay`
- Topics (Rectangles):**
 - `/head_controller/command`
 - `/head_controller/point_head_action`
 - `/head_controller/follow_joint_trajectory/goal`
 - `/head_controller/follow_joint_trajectory/status`
 - `/torso_controller/follow_joint_trajectory/cancel`
 - `/joint_states`
 - `/tf`
 - `/tf_static`
 - `/tf_lookup`
 - `/mobile_base_controller/odom`
 - `/stero_listener_10603_1635337923486`
 - `/mobile_base_controller/cmd_vel`
 - `/key_vel`
 - `/joy_vel`
 - `/joystick_relay`
 - `/xrtion/rgb/image_raw`
- Dependencies (Arrows):**
 - `/default_controllers_spawner` and `/bringup_controllers_spawner_imu` point to `/gazebo`.
 - `/gazebo` points to `/robot_state_publisher` (topic `/joint_states`), `/head_controller` (topic `/head_controller/command`), `/play_motion` (topic `/tf_static`), `/trajectory_execution_event` (topic `/tf_static`), `/move_group` (topic `/tf_static`), `/twist_mux` (topic `/key_vel`), `/twist_marker` (topic `/joy_vel`), `/fs_already_there` (topic `/xrtion/rgb/image_raw`), and `/image_raw_to_rect_color_relay` (topic `/xrtion/rgb/image_raw`).
 - `/robot_state_publisher` points to `/head_controller` (topic `/head_controller/point_head_action`).
 - `/head_controller` points to `/play_motion` (topic `/head_controller/follow_joint_trajectory/goal`).
 - `/play_motion` points to `/trajectory_execution_event` (topic `/head_controller/follow_joint_trajectory/status`).
 - `/trajectory_execution_event` points to `/move_group` (topic `/torso_controller/follow_joint_trajectory/cancel`).
 - `/move_group` points to `/twist_mux` (topic `/head_controller/follow_joint_trajectory/status`).
 - `/twist_mux` points to `/twist_marker` (topic `/mobile_base_controller/cmd_vel`).
 - `/stero_listener_10603_1635337923486` points to `/twist_mux` (topic `/stero_listener_10603_1635337923486`).
 - `/joystick_relay` points to `/twist_mux` (topic `/joystick_relay`).

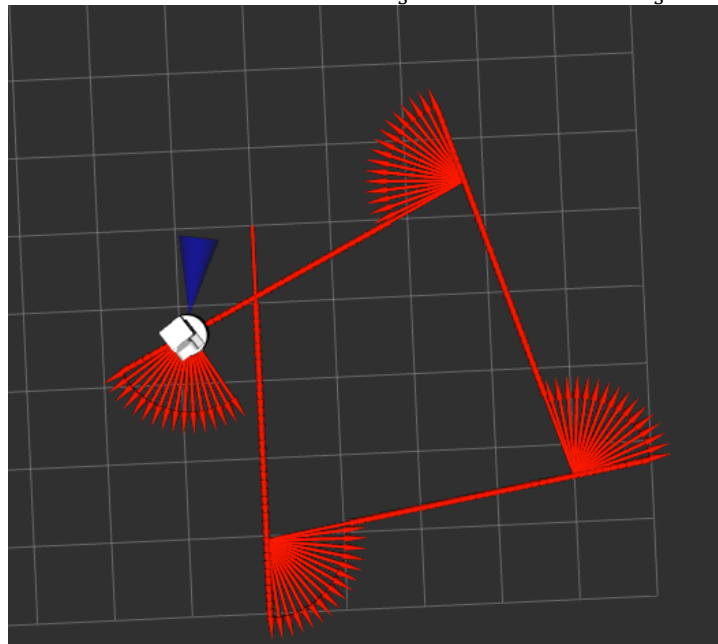
Działanie programu sprawdzono jak poprzednio za pomocą symulacji. Efekty interpolacji w programie RViz:

A 3D visualization of a square path on a dark gray grid floor. The path is defined by four red lines forming a square. At each of the four corners of the square, there is a fan-shaped arrangement of many thin red lines radiating outwards. At the top-left corner, a small white cube is positioned, and a blue cone points upwards from it.

Prędkość kątowna: $0,12 \frac{rad}{s}$, liniowa $0,288 \frac{m}{s}$

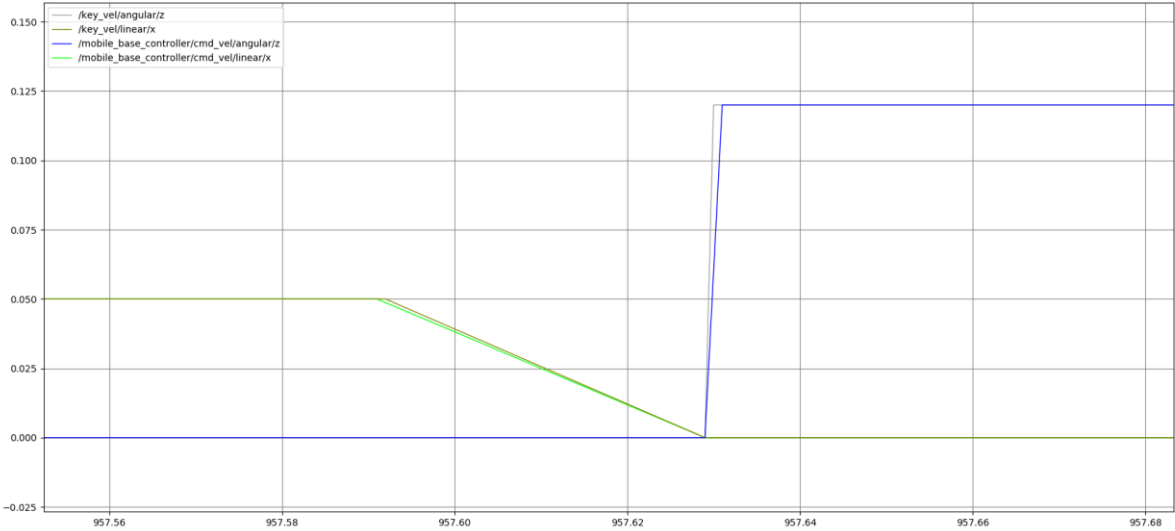


Prędkość kątowna: $0,288 \frac{rad}{s}$, liniowa $0.6912 \frac{m}{s}$

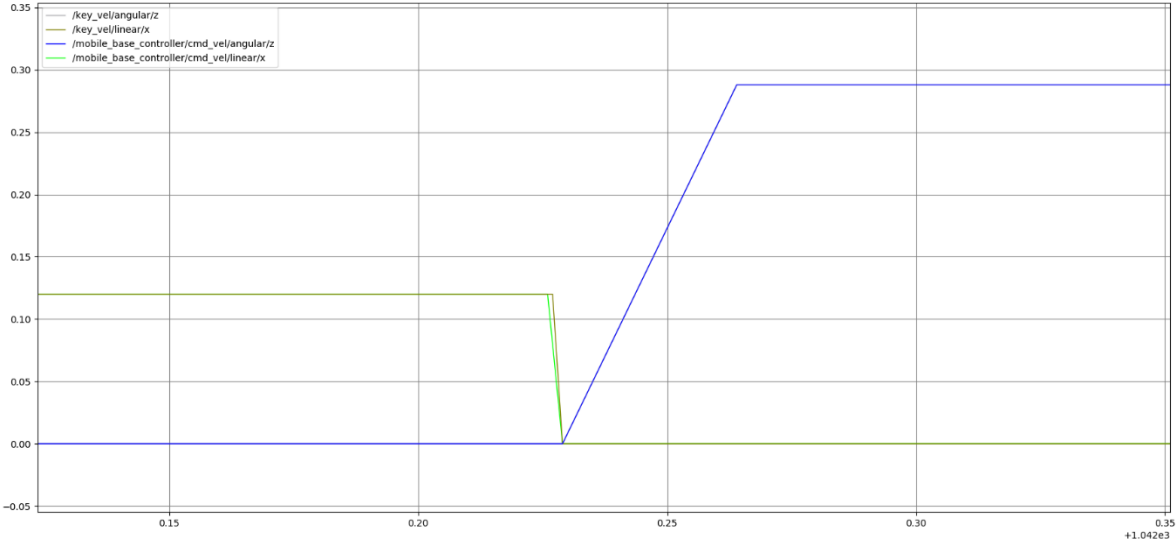


Analogicznie jak wcześniej sprawdzeniu podlegała zgodność zadanej prędkości z faktyczną prędkością robota:

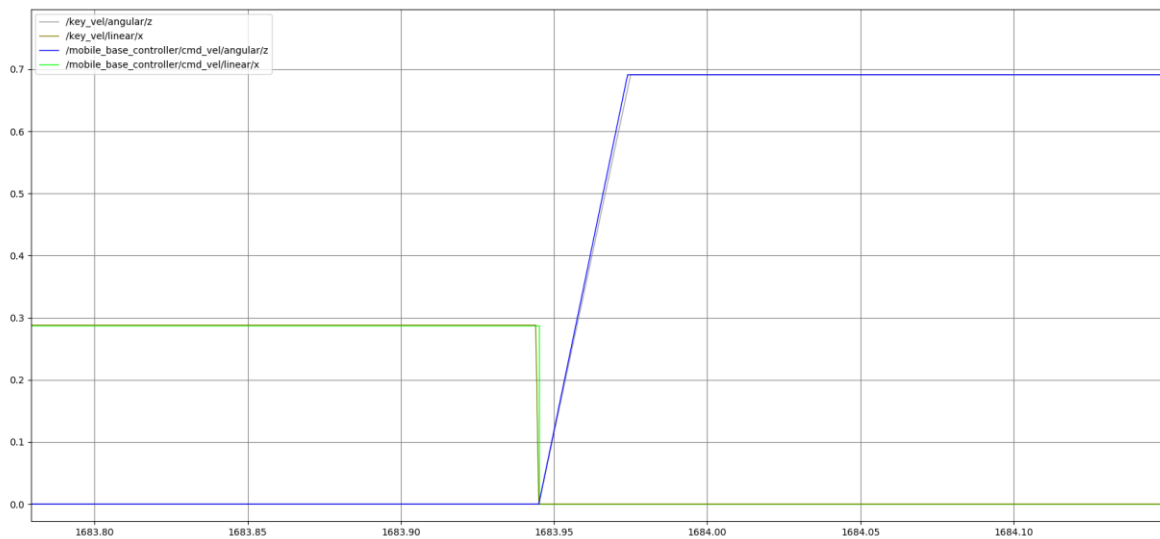
Prędkość kątowa: $0,05 \frac{rad}{s}$, liniowa $0,12 \frac{m}{s}$



Prędkość kątowa: $0,12 \frac{rad}{s}$, liniowa $0,288 \frac{m}{s}$



Prędkość kątowa: $0,288 \frac{rad}{s}$, liniowa $0.6912 \frac{m}{s}$



Uwaga!

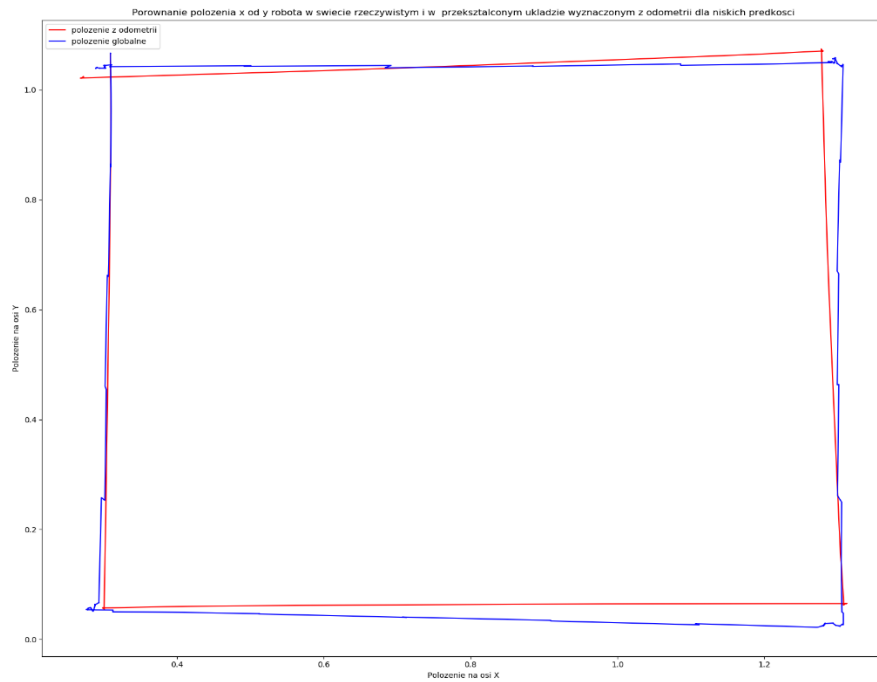
Na przebiegu oznaczono kolorem:

- szarym – zadaną prędkość kątową
- ciemnozielonym – zadaną prędkość liniową
- niebieskim – odczytaną liniową kątową robota
- jasnozielonym – odczytaną liniową prędkość robota

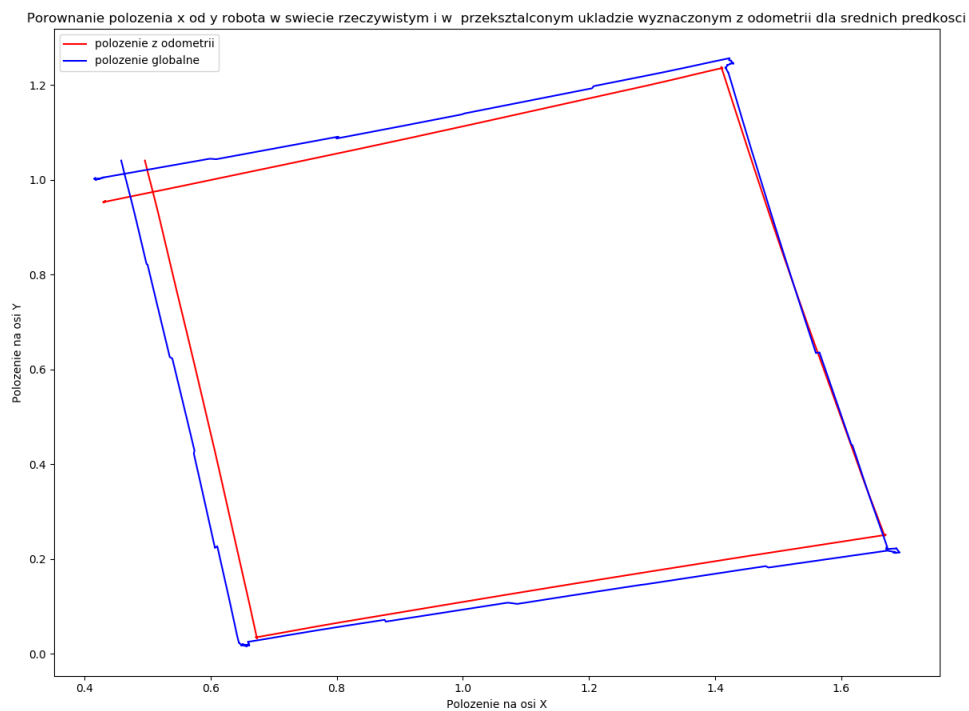
Obserwując przebiegi można zauważyć, że występowało nieznaczne opóźnienie między zadaną prędkością, a faktyczną prędkością robota. Większą różnicę widać w symulowanym ruchu robota. Przy wysokiej prędkości interpolacja kwadratu była mocno zniekształcona.

Dodatkową formą sprawdzenia algorytmu było przetestowanie go na realnym robocie Tiago. Robot miał za zadanie przejechać po interpolowanym kwadracie o boku długości 1m. Podczas testów nagrano lokalizacje robota na podstawie samej odometrii oraz fuzji różnych czujników(położenie globalne). Porównanie ich widoczne jest na wykresach poniżej:

Prędkość kątowa: $0,05 \frac{rad}{s}$, liniowa $0,12 \frac{m}{s}$

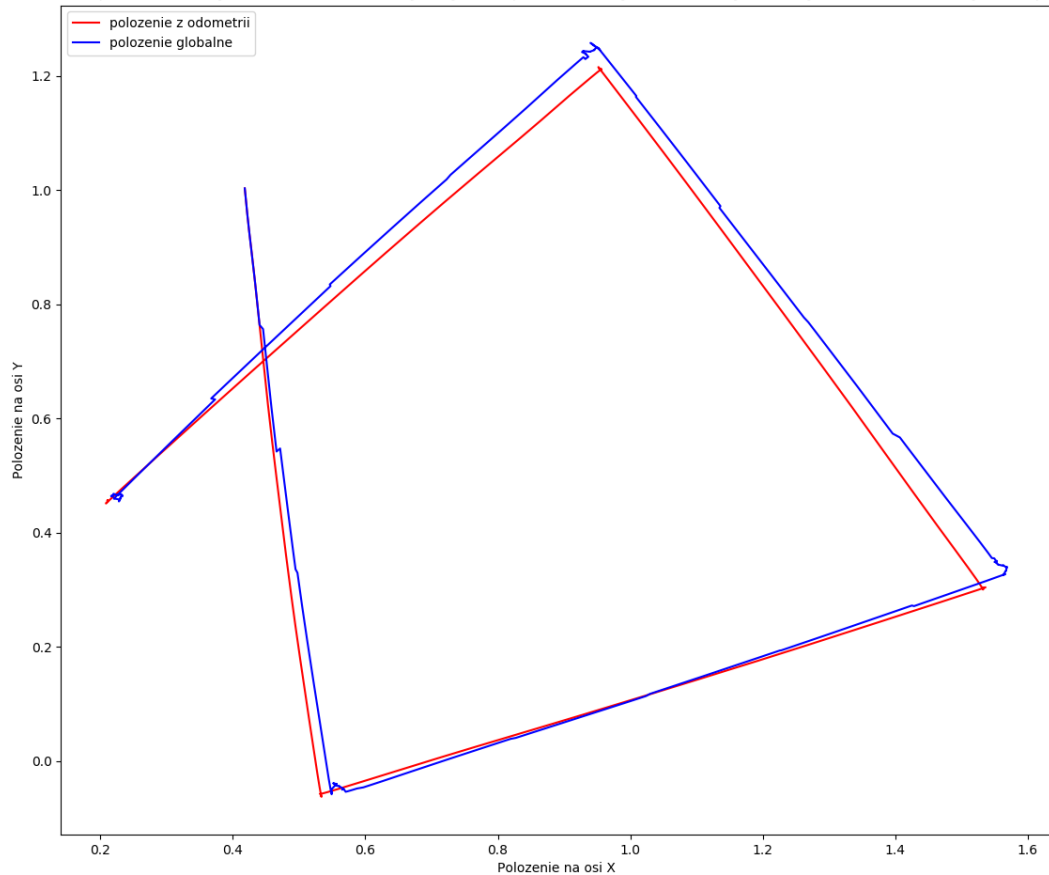


Prędkość kątowa: $0,12 \frac{rad}{s}$, liniowa $0,288 \frac{m}{s}$



Prędkość kątowna: $0,288 \frac{rad}{s}$, liniowa $0.6912 \frac{m}{s}$

Porównanie położenia x od y robota w świecie rzeczywistym i w przekształconym układzie wyznaczonym z odometrii dla wysokich prędkości



Dla każdego z przejazdów wyznaczono skumulowaną sumę modułów błędów oraz błąd średni.

Błąd średni został wyliczony zgodnie ze wzorem:

$$MAE = \frac{1}{n} \sum_{i=1}^n |y - \gamma|$$

Gdzie:

- y – położenie robota z odometrii
- γ – położenie robota w globalnym układzie współrzędnych
- n – liczba próbek

Tabela błędów uzyskanych w przejazdach

Prędkości w przejazdach	Błąd całkowity			Średni błąd		
	Położenia X	Położenia Y	Rotacji Z	Położenia X	Położenia Y	Rotacji Z
$L = 0,05 \frac{rad}{s}$ $A = 0,12 \frac{m}{s}$	118,622m	134,92m	12074°	0,016m	0,018m	1,66°
$L = 0,12 \frac{rad}{s}$ $A = 0,288 \frac{m}{s}$	106,1m	82,875m	4177°	0,03m	0,023m	1,18°
$L = 0,288 \frac{rad}{s}$ $A = 0,6912 \frac{m}{s}$	21,19m	29,29m	1403,5°	0,011m	0,015m	0.37°

Wnioski

W realnych zastosowaniach rzadko można użyć ustalonego schematu ruchu. Z tego powodu, pomimo, iż algorytm oparty na odometrii gorzej poradził sobie z postawionym zadaniem to ma on o wiele szersze zastosowanie. Sama odometria w porównaniu z położeniem pochodzącym z fuzji czujników dawała bardzo zbliżone do niego rezultaty. Świadczy to o wysokiej jakości