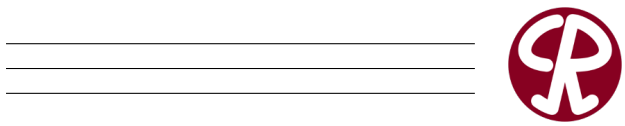
Projekt

Roboty mobilne

Nawigacja robota Pioneer 3-DX z systemem detekcji kolizji w symulacji

Michał Trela 259312

Jan Masłowski 258962



Prowadzący:
Dr inż. Michał Błędowski

Katedra Cybernetyki i Robotyki
Wydziału Elektroniki, Fotoniki i Mikrosystemów
Politechniki Wrocławskiej

22 czerwca 2023

1 Cel projektu

Stworzenie symulacji służącej do nawigacji robota, systemu detekcji kolizji oraz mapowania obszaru na podstawie czujników ultradźwiękowych.

2 Założenia projektowe

Projekt polega na stworzeniu symulacji umożliwiającej teleoperację oraz autonomiczną operację robota typu Pioneer 3-DX oraz obsługę LIDAR'u w celu stworzenia systemu detekcji kolizji oraz mapowania obszaru. Całość projektu zrealizowana będzie za pomocą frameworka ROS2. Stworzone oprogramowanie oraz dokumentacja będą udostępniane za pomocą systemu kontroli wersji Git.

3 Podział pracy

- Konfiguracja i uruchomienie symulacji, teleoperacja, obsługa LIDAR'u i mapowanie obszaru - Michał Trela
- Stworzenie modelu, autonomia, system detekcji kolizji- Jan Masłowski

4 Etapy projektu

Etap I:

- 23.03 - Założenia projektowe

Etap II:

- Przegląd literatury oraz zasobów internetowych [1] [2]
- 13.04 - Stworzenie modelu
- 20.04 - Konfiguracja i uruchomienie symulacji
- 11.05 - Teleoperacja

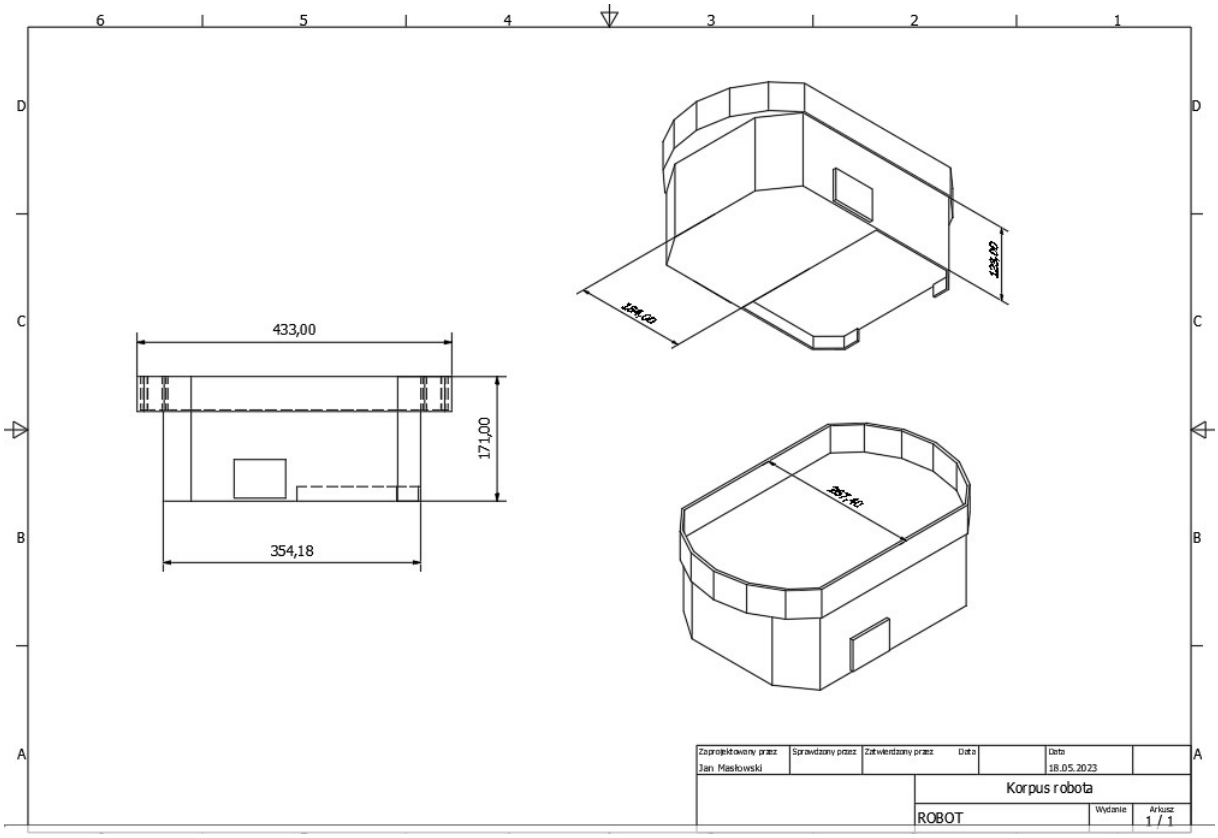
Etap III:

- 01.06 - Autonomia
- 08.06 - Obsługa LIDAR'u i mapowanie obszaru
- 22.06 - System detekcji kolizji

5 Osiągniecia – Etap II

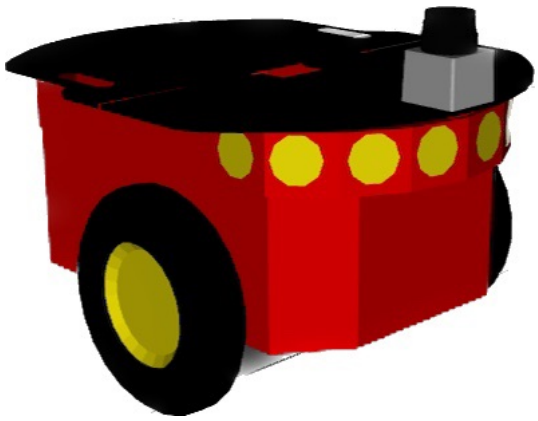
5.1 Model Robota

Model robota Pioneer 3-DX został odwzorowany na podstawie modelu znalezionego w internecie. Wymiary robota zostały pobrane z ogólnodostępnych siatek, a części stworzone w programie CAD – Inventor.



Rysunek 1: Rysunek techniczny korpusu robota.

Model został złożony w formacie URDF (Unified Robotics Description Format) umożliwiając odpowiednią konfigurację w ROS2.



Rysunek 2: Wygląd robota w środowisku symulacyjnym Gazebo.

5.2 Konfiguracja i uruchomienie symulacji

Robot Pioneer jest robotem o napędzie różnicowym co wymagało skonfigurowania odpowiednich narzędzi w celu poprawnego sterowania robotem. Wykorzystana do tego została paczka ros2_control. Napisany został plik konfiguracyjny przekazujący odpowiednie parametry. na bazie odpowiednich parametrów uruchamiany jest nadajnik stanu przegubów oraz kontroler napędu różnicowego

```yaml
controller_manager:
  ros__parameters:
    update_rate: 50  # Hz
    use_sim_time: true

    joint_state_broadcaster:
      type: joint_state_broadcaster/JointStateBroadcaster

    diff_drive_controller:
      type: diff_drive_controller/DiffDriveController

diff_drive_controller:
  ros__parameters:
    left_wheel_names: ['left_wheel_joint']
    right_wheel_names: ['right_wheel_joint']

    wheel_separation: 0.35
    wheel_radius: 0.1

    wheel_separation_multiplier: 1.0
    left_wheel_radius_multiplier: 1.0
    right_wheel_radius_multiplier: 1.0

    open_loop: false
    enable_odom_tf: true

    cmd_vel_timeout: 0.6
    use_stamped_vel: false

    publish_rate: 50.0
```

Rysunek 3: Część pliku konfiguracyjnego

Następnie środowisko symulacji Gazebo wymagało odpowiedniego skonfigurowania pluginów. Należało ustawić odpowiednie interfejsy poleceń dla kół jezdnych oraz interfejsy stanu dla kół jezdnych oraz kół obrotowych.

```
<xacro:macro name="differential_drive" params="name">
  <ros2_control name="GazeboSystem" type="system">
    <hardware>
      <plugin>gazebo_ros2_control/GazeboSystem</plugin>
    </hardware>
    <joint name="left_wheel_joint">
      <command_interface name="velocity">
        <param name="min">-10</param>
        <param name="max">10</param>
      </command_interface>
      <state_interface name="velocity"/>
      <state_interface name="position"/>
    </joint>
    <joint name="right_wheel_joint">
      <command_interface name="velocity">
        <param name="min">-10</param>
        <param name="max">10</param>
      </command_interface>
      <state_interface name="velocity"/>
      <state_interface name="position"/>
    </joint>
    <joint name="swivel_joint">
      <state_interface name="velocity"/>
      <state_interface name="position"/>
    </joint>
    <joint name="swivel_wheel_joint">
      <state_interface name="velocity"/>
      <state_interface name="position"/>
    </joint>
  </ros2_control>

  <gazebo>
    <plugin name="gazebo_ros2_control" filename="libgazebo_ros2_control.so">
      <parameters>$(find pioneer3dx_description)/config/pioneer_control.yaml</parameters>
    </plugin>
  </gazebo>
</xacro:macro>

<xacro:macro name="joint_state_publisher" params="name">
  <gazebo>
  <plugin name="gazebo_ros_joint_state_publisher"
    filename="libgazebo_ros_joint_state_publisher.so">
```

Rysunek 4: Konfiguracja pluginu

5.3 Teleoparacja

Do teleoperacji wykorzystywana jest paczka teleop_twist publikująca odpowiednie dane na topic zadając prędkości na koła.

```
linear:
  x: 0.5
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
linear:
  x: 0.5
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
```

Rysunek 5: Wiadomości publikowane na topic \cmd_vel

Publikowane wiadomości są odbierane przez kontroler napędu różnicowego, który następnie zadaje prędkości na koła

6 Etap III

6.1 SLAM

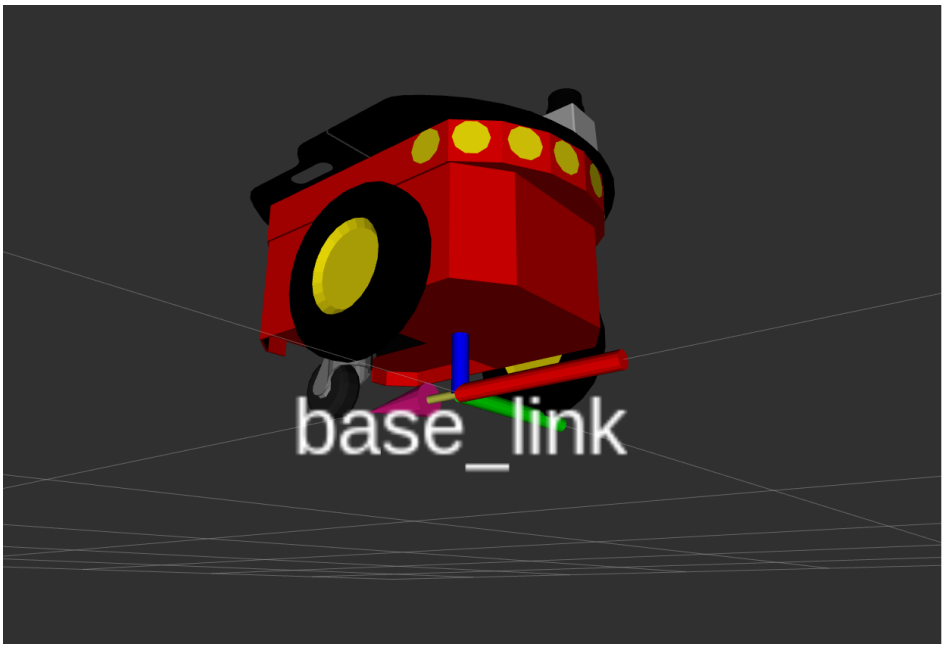
Mapowanie zostało zaimplementowane za pomocą algorytmu SLAM korzystając z paczki ROS2 `slam_toolbox`. Konieczne było stworzenie odpowiedniego pliku konfiguracyjnego w celu poprawnego działania algorytmów.

```yaml
slam_toolbox:
  ros__parameters:

    # ROS Parameters
    odom_frame: odom
    map_frame: map
    base_frame: base_link
    scan_topic: /scan_raw
    mode: localization

    # Plugin params
    solver_plugin: solver_plugins::CeresSolver
    ceres_linear_solver: SPARSE_NORMAL_CHOLESKY
    ceres_preconditioner: SCHUR_JACOBI
    ceres_trust_strategy: LEVENBERG_MARQUARDT
    ceres_dogleg_type: TRADITIONAL_DOGLEG
    ceres_loss_function: None

    # General Parameters
    use_scan_matching: true
    use_scan_barycenter: true
    minimum_travel_distance: 0.5
    minimum_travel_heading: 0.5
    scan_buffer_size: 10
    scan_buffer_maximum_scan_distance: 10.0
    link_match_minimum_response_fine: 0.1
    link_scan_maximum_distance: 1.5
    loop_search_maximum_distance: 3.0
    do_loop_closing: true
    loop_match_minimum_chain_size: 10
    loop_match_maximum_variance_coarse: 3.0
    loop_match_minimum_response_coarse: 0.35
    loop_match_minimum_response_fine: 0.45
```

Listing 1: Fragment pliku konfiguracyjnego `slam_params.yaml`

6.2 Autonomia

Robot porusza się autonomicznie za pomocą błądzenia losowego. Moduł autonomii robota nadaje prędkości liniowe i kątowe na kontroler dyferencjału robota, który nadaje prędkości na poszczególne koła. Domyślnie moduł autonomii nadaje tylko prędkość liniową wzdłuż osi `X`. Co ustaloną ilość czasu wywoływana jest funkcja losująca kierunek obrotu a następnie nadająca odpowiednie prędkości kątowe powodując obrót wokół osi `Z` robota. Lokalnym układem współrzędnych robota, względem którego się orientuje, jest układ `base_link` 6. Oś `X` jest wizualizowana na czerwono, oś `Y` na zielono a oś `Z` na niebiesko.



Rysunek 6: Część pliku konfiguracyjnego

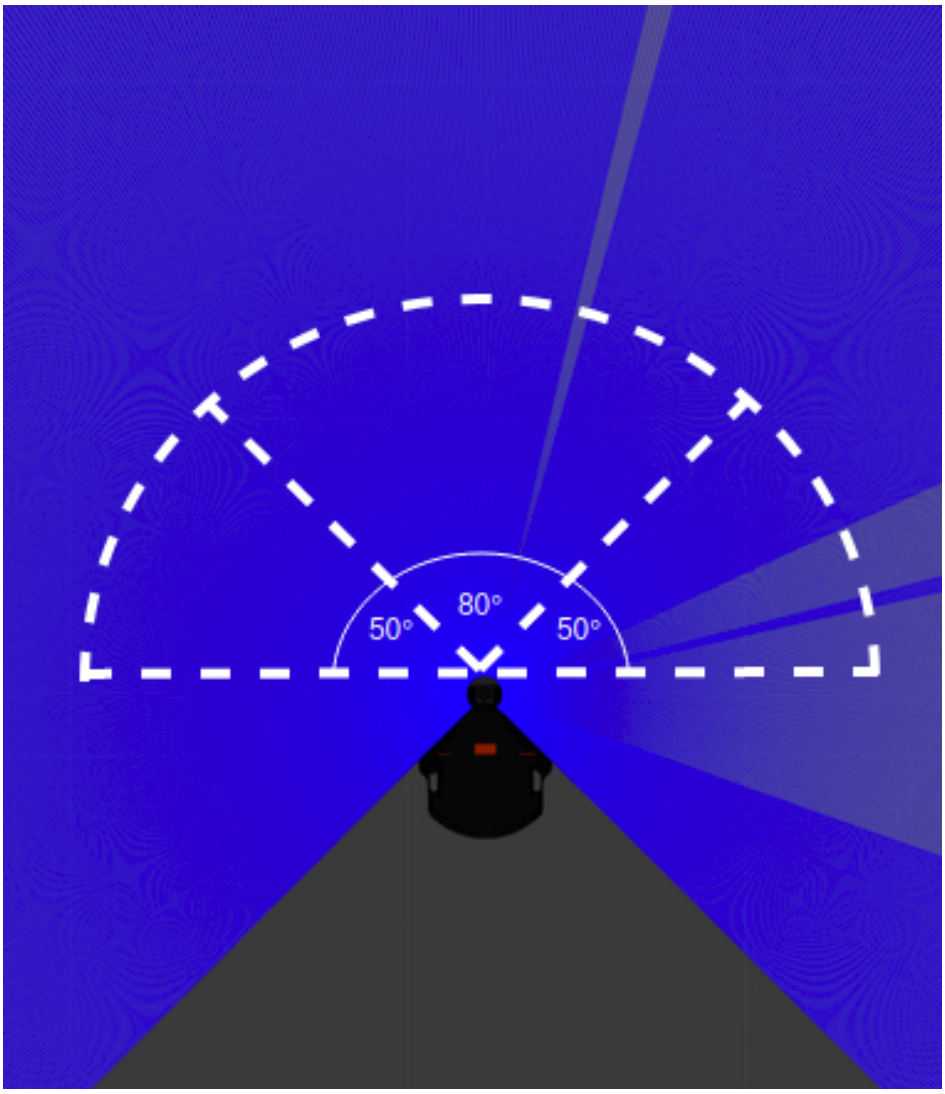
```
def random_turn_timer_callback(self):
    duration = 4
    start_time = time.time()
    left_flag = random.choice([True, False])

    while time.time() - start_time < duration:
        cmd = Twist()
        if left_flag:
            cmd.angular.z = 0.5
        else:
            cmd.angular.z = -0.5

        self.velocity_publisher.publish(cmd)

        time.sleep(0.1)
```

Listing 2: Metoda wywołująca losowy obrót

6.3 Zapobieganie kolizji

Zapobieganie kolizji wykorzystuje pomiary odległości z LIDAR'u. Wykorzystywane są trzy zakresy 15 wycięte z tabeli danych otrzymanych z LIDAR'u:

- Sektor środkowy: pole widzenia o kącie ok. 80°
- Sektor lewy i prawy: pole widzenia o kącie ok. 50°



Rysunek 7: Zakresy sektorów detekcji

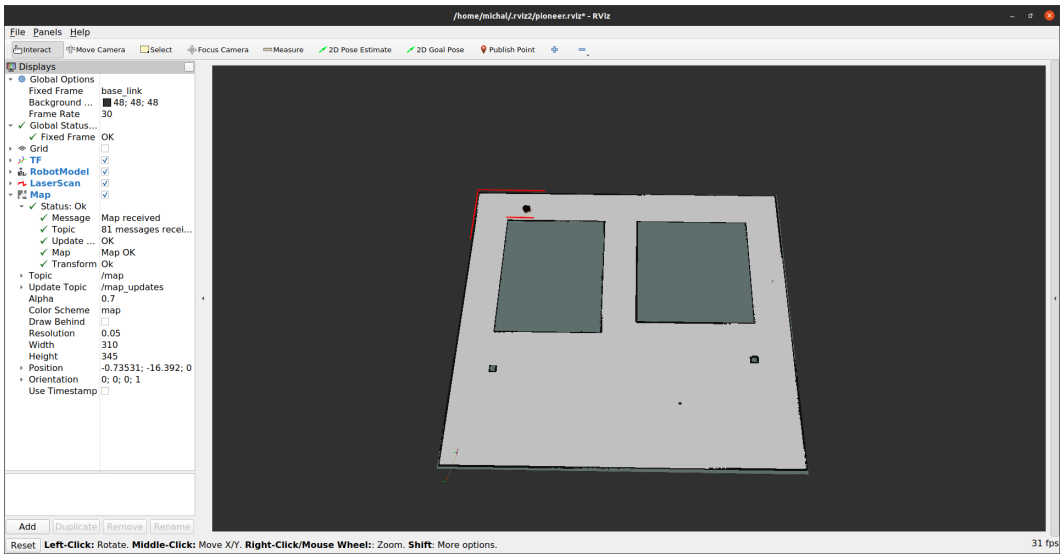
W momencie gdy jakikolwiek odczyt z sektora środkowego zwróci odległość mniejszą niż 1m, zadana jest zerowa prędkość liniowa i znajdywana jest średnia odległość w sektorze lewym i porównywana ze średnią odległością w sektorze prawym. Tam gdzie średnia jest mniejsza, tam ustalany jest kierunek obrotu. Obrót jest kontynuowany dopóki nie będzie żadnej odległości mniejszej od 1m w sektorze środkowym.

```
def lidar_callback(self, msg: LaserScan):
    lidar_points = len(msg.ranges)

    if lidar_points == 0:
        return

    cmd = Twist()
    cmd.linear.x = 0.5

    middle_start = lidar_points // 2 - 100
    middle_end = middle_start + 200

    right_end = middle_start
    right_start = right_end - 100

    left_start = middle_end
    left_end = left_start + 100

    if any(value < 1.0 for value in msg.ranges[middle_start:
        middle_end]):
        cmd.linear.x = 0.0

        right_average = sum(
            msg.ranges[right_start:right_end]) / len(msg.ranges[
                right_start:right_end])
        left_average = sum(
            msg.ranges[left_start:left_end]) / len(msg.ranges[
                left_start:left_end])

        if left_average > right_average:
            cmd.angular.z = 0.5
        else:
            cmd.angular.z = -0.5

    self.velocity_publisher.publish(cmd)
```

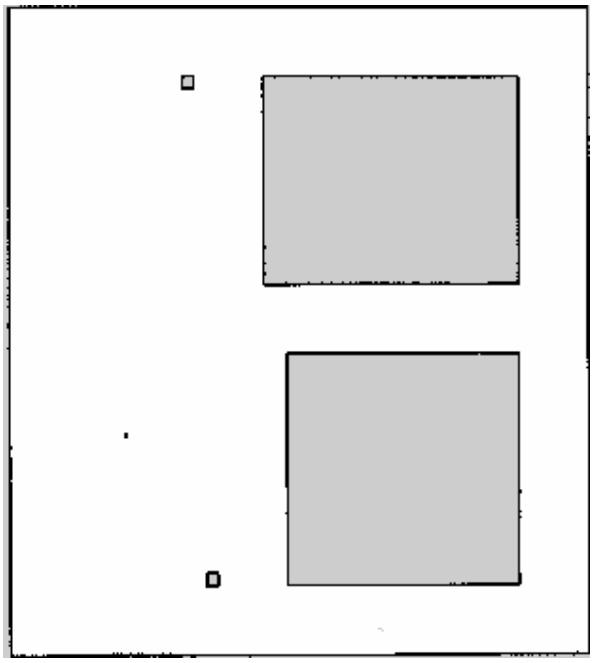
Listing 3: Metoda wywołująca losowy obrót

6.4 Mapowanie terenu

Tworzenie mapy odbywa się poprzez `slam_toolbox`. Algorytm mapowania odczytuje dane otrzymane z LIDAR'u, oblicza położenie punktów, czyli przeszkód na podstawie odległości, kąta padania oraz danych z odometrii robota czyli globalnej orientacji i położenia. Następnie łączy punkty ze sobą tworząc mapę terenu w postaci `OccupancyGrid`. Robot autonomicznie poruszając się po terenie wysyła dane, z których tworzona jest mapa. Mapowanie kończy się wtedy, kiedy przez zadany okres czasu, nie są dodawane nowe punkty do mapy.



Rysunek 8: Wizualizacja robota i mapy w programie RViz2

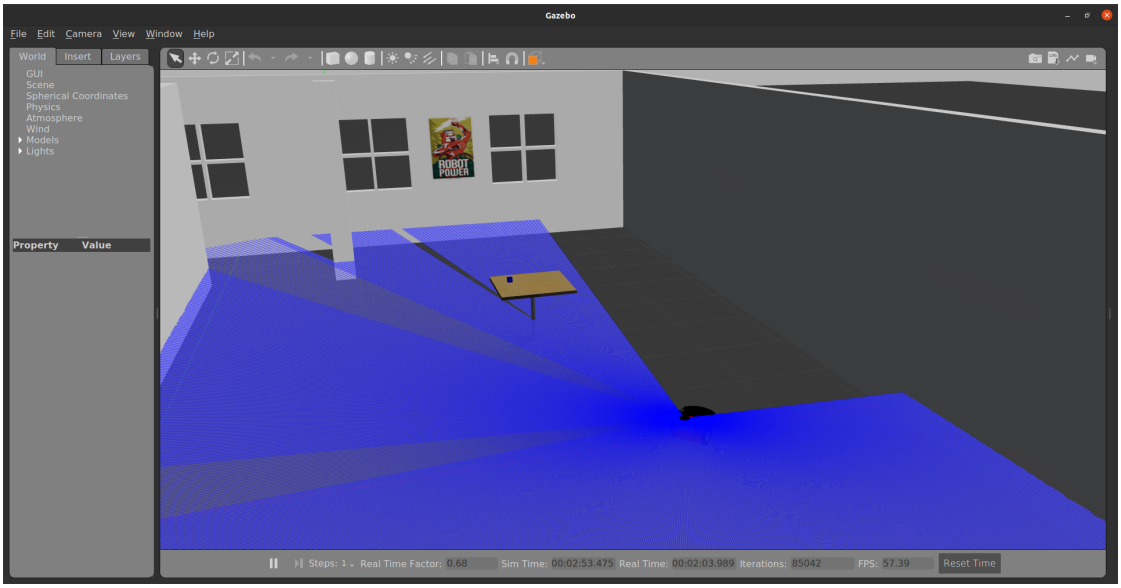


Rysunek 9: Stworzona mapa terenu

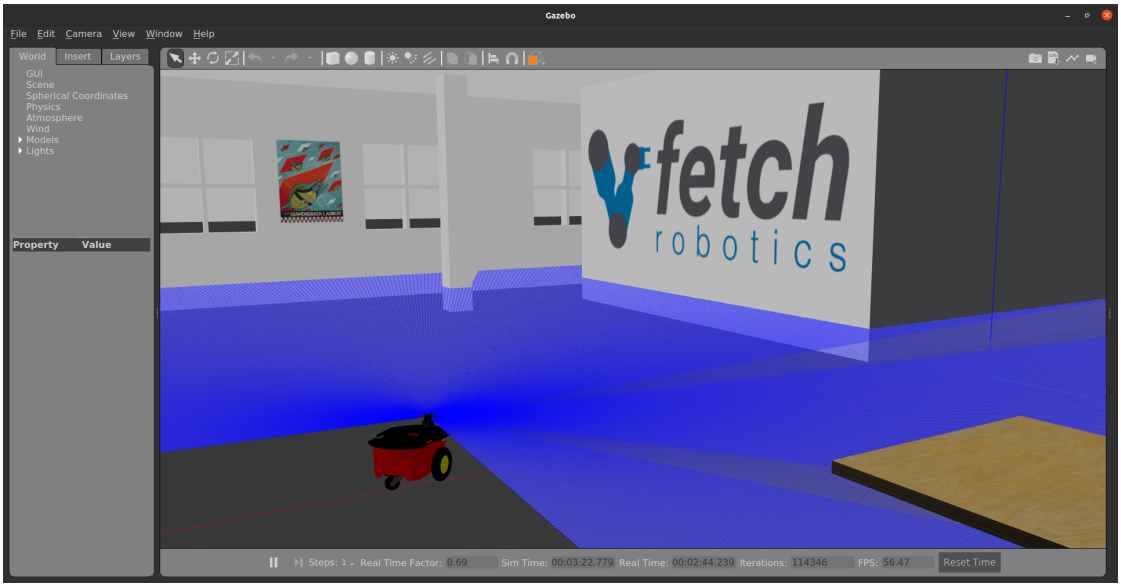
6.5 Efekt końcowy

Efektem końcowym projektu jest symulacja robota Pioneer 3-DX w środowisku Gazebo. Porusza się i mapuje teren autonomicznie wykorzystując błądzenie losowe, czujniki odległości oraz algorytm wyboru kierunku skrętu w sytuacji napotkania przeszkody.

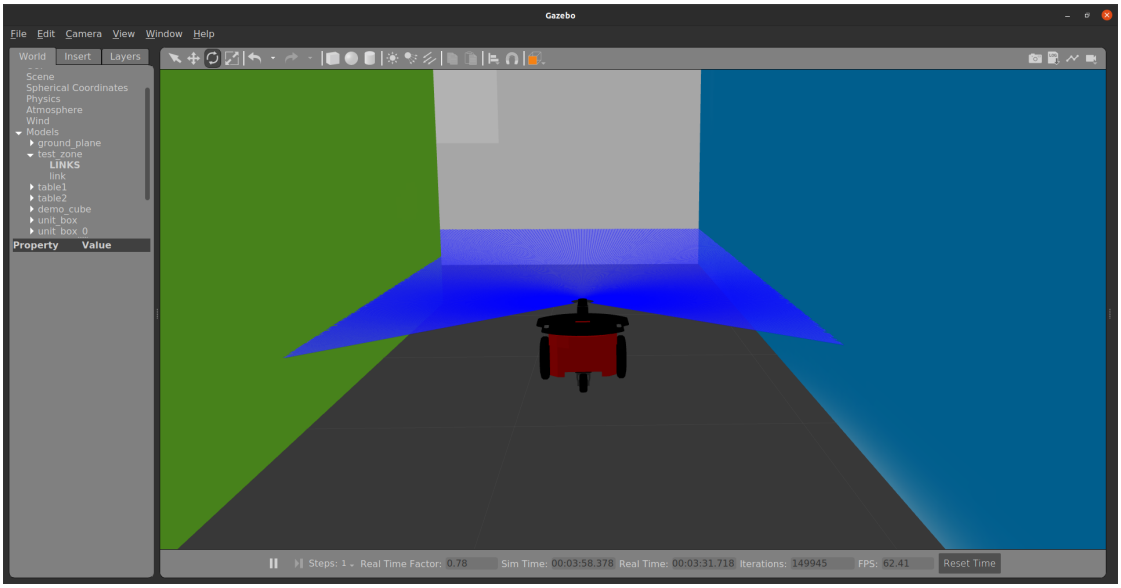
6.5.1 Symulacja



Rysunek 10: Symulacja robota w środowisku Gazebo

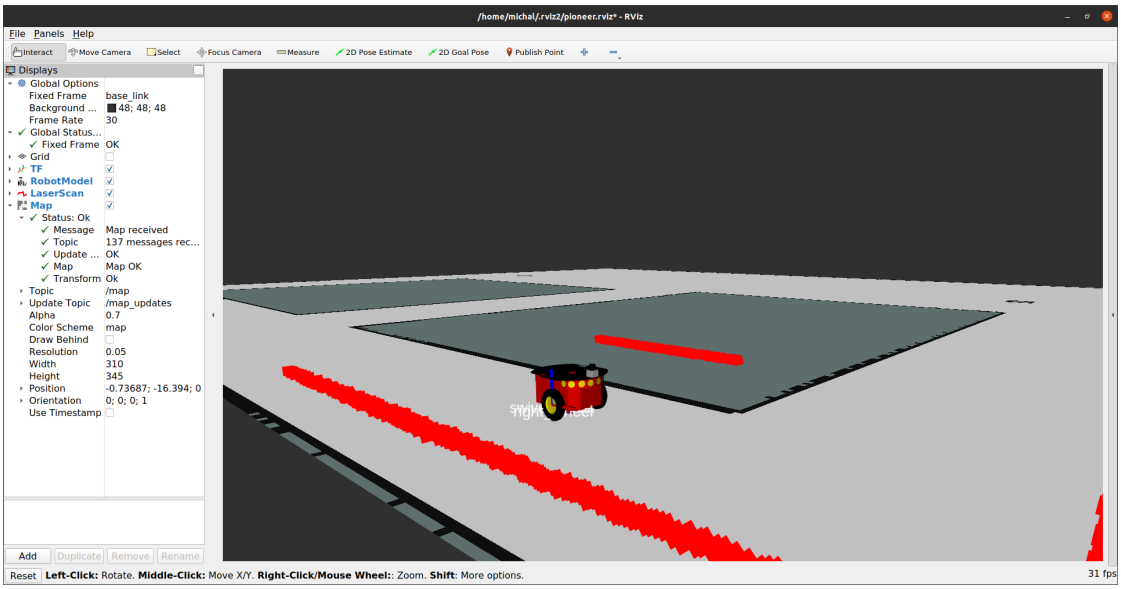


Rysunek 11: Symulacja robota w środowisku Gazebo

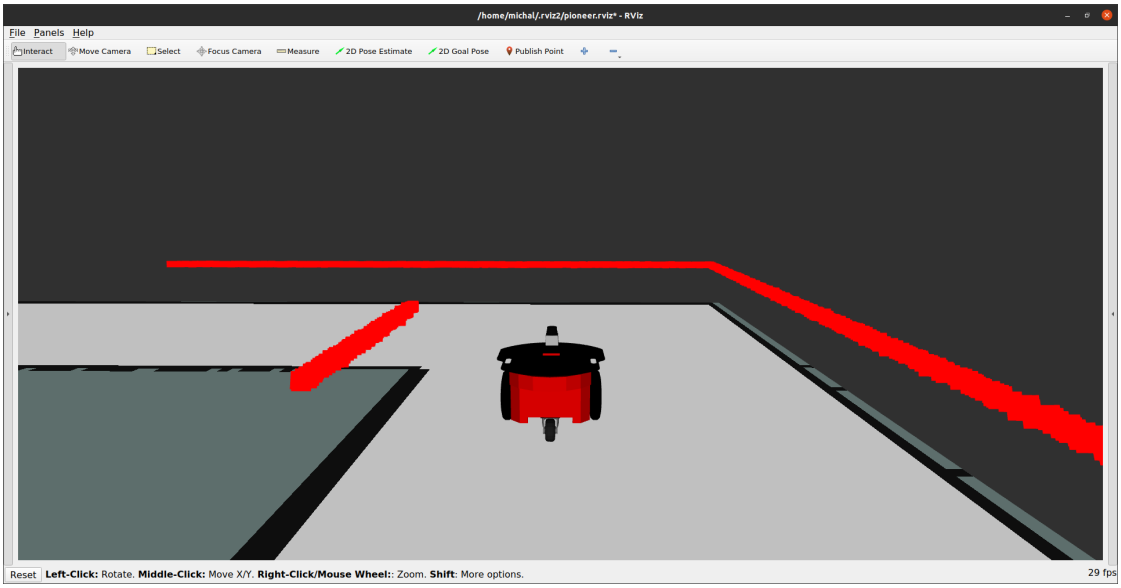


Rysunek 12: Symulacja robota w środowisku Gazebo

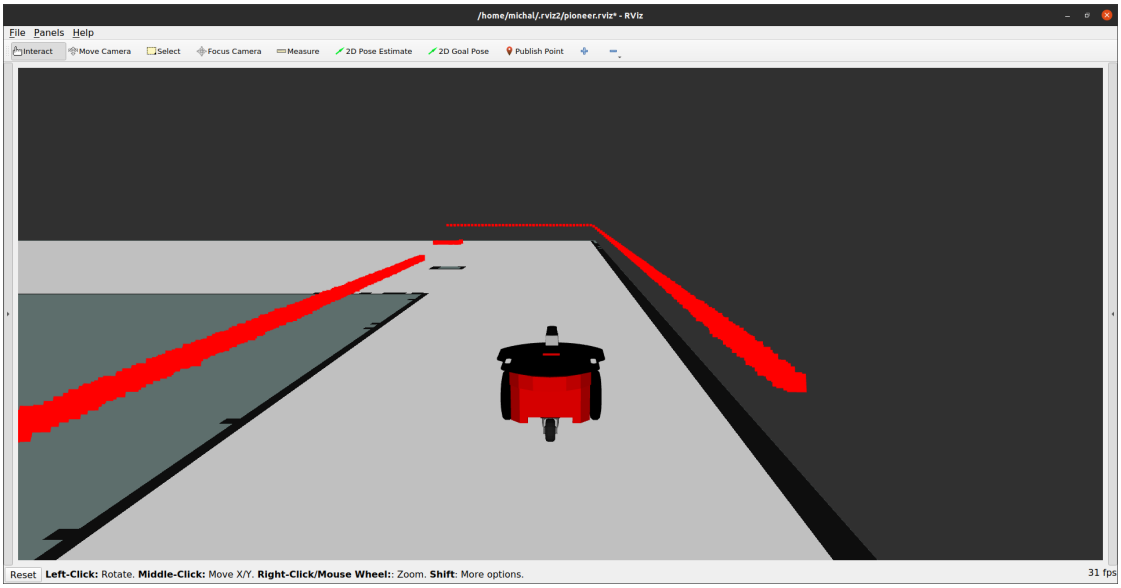
6.5.2 Wizualizacja



Rysunek 13: Wizualizacja robota w RViz2



Rysunek 14: Wizualizacja robota w RViz2



Rysunek 15: Wizualizacja robota w RViz2

6.5.3 Mapowanie

Prezentacja mapowania została nagrana i umieszczona w serwisie YouTube.

Literatura

[1] Marco Matteo Bassa. *A very informal journey through ROS 2*. Marco Matteo Bassa, 2023.

[2] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022.