Wydział Elektroniki i Technik Informacyjnych
Politechnika Warszawska

STERO

Sprawozdanie z laboratorium i projektu nr 2

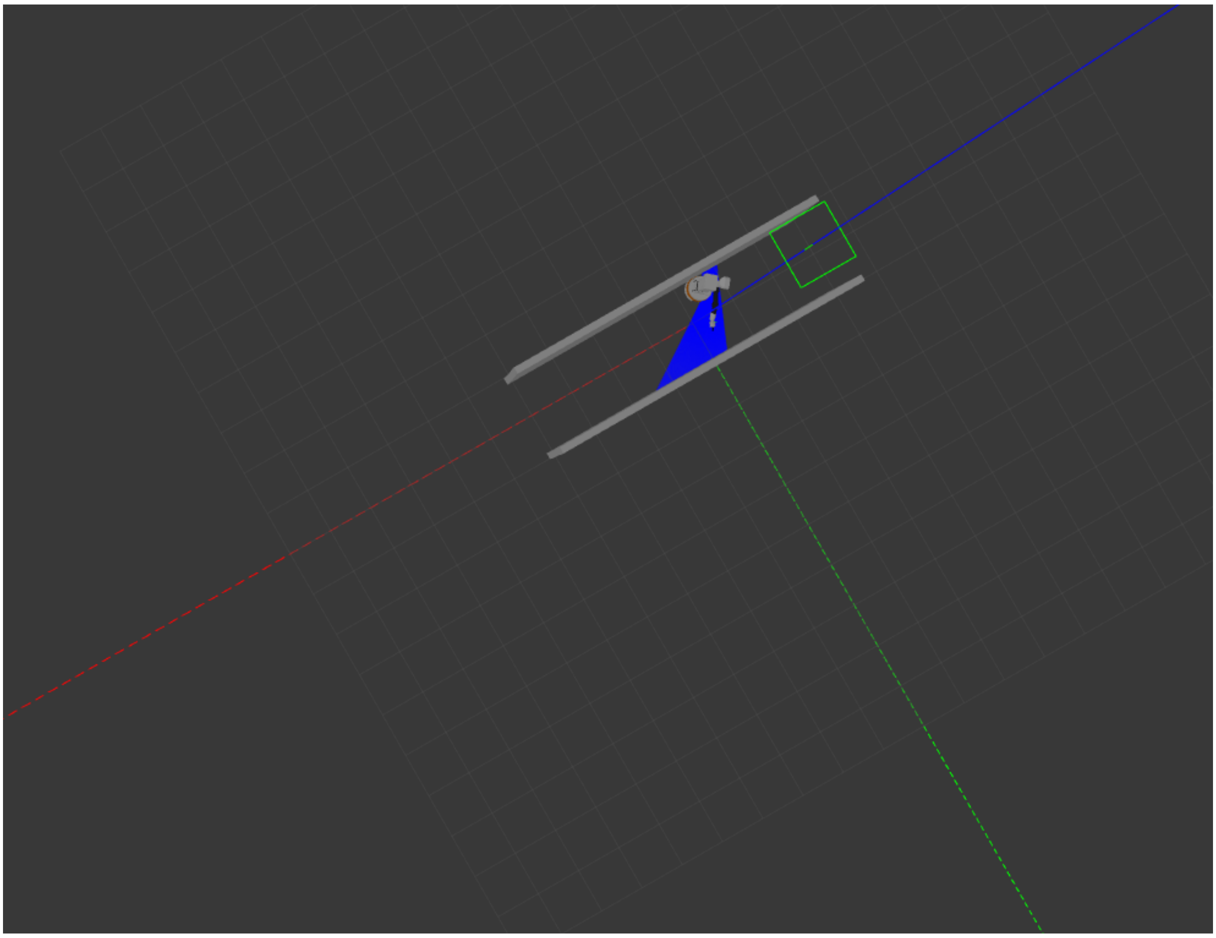
Michał Pióro, Tadeusz Chmielik

Warszawa, 2024

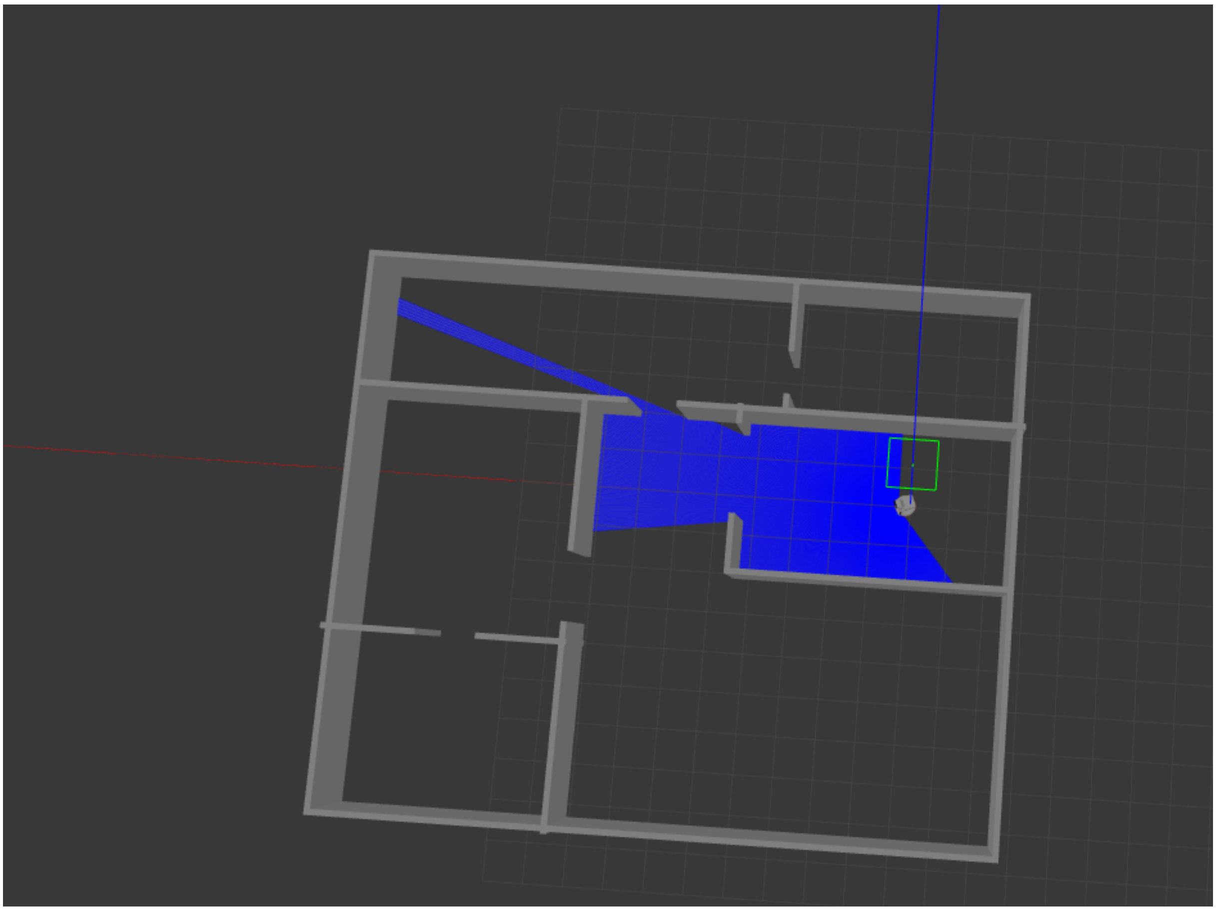
Spis treści

1. Budowa światów w Gazebo

W Gazebo zbudowano 2 światy. Pierwszym z nich jest przedstawimy na zrzucie ekranu 1.1 korytarz (hall.world), który zostanie wykorzystany na 6 laboratoriach oraz przedstawione na kolejnym zrzucie ekranu 1.2 mieszkanie (flat.world)„ które zawiera 6 pomieszczeń z 5 parami drzwi o wymiarach: $1m$, $1{,}5m$, $1{,}75m$, $2m$, $2{,}5m$. To właśnie ten świat został wykorzystany podczas dalszych zadań w tym laboratorium.



Rys. 1.1. Świat korytarz w Gazebo



Rys. 1.2. Świat mieszkanie w Gazebo

2. Uruchomienie symulacji w nowym świecie

W celu uruchomienia symulacji w Gazebo z wykorzystaniem własnego świata umieszczonego w pakiecie lab5 przebudowano oryginalne pliki uruchomieniowe, które następnie umieszczono we własnym pakiecie. Jako pierwszy przebudowano plik *pal_gazebo.launch.py* dodając w nim argument **world_package** co pozwoliło na wybranie pakietu, w którym znajduje się folder *world* ze światami. Zapisano go jako *gazebo.launch.py*

```
import os
from os import environ, pathsep

from ament_index_python.packages import get_package_share_directory

from launch import LaunchDescription
from launch.actions import (
    DeclareLaunchArgument,
    SetEnvironmentVariable,
    ExecuteProcess,
    OpaqueFunction
)
from launch.substitutions import LaunchConfiguration, PathJoinSubstitution


def start_gzserver(context, *args, **kwargs):
    world_package = LaunchConfiguration('world_package').perform(context)
    world_name = LaunchConfiguration('world_name').perform(context)

    # Dynamically get the package path based on the world_package argument
    pkg_path = get_package_share_directory(world_package)
    world_file = os.path.join(pkg_path, 'worlds', world_name + '.world')

    if not os.path.exists(world_file):
        raise FileNotFoundError(f"World file {world_name}.world not found in pac

    params_file = PathJoinSubstitution(
        substitutions=[pkg_path, 'config', 'gazebo_params.yaml'])

    # Command to start the gazebo server.
    gazebo_server_cmd_line = [
        'gzserver', '-s', 'libgazebo_ros_init.so',
        '-s', 'libgazebo_ros_factory.so', world_file,
        '--ros-args', '--params-file', params_file]

    # If debugging is required, launch with gdb.
    debug = LaunchConfiguration('debug').perform(context)
    if debug == 'True':
```

```
        gazebo_server_cmd_line = (
            ['xterm', '-e', 'gdb', '-ex', 'run', '--args'] +
            gazebo_server_cmd_line
        )

    start_gazebo_server_cmd = ExecuteProcess(
        cmd=gazebo_server_cmd_line, output='screen')

    return [start_gazebo_server_cmd]


def generate_launch_description():
    declare_world_package = DeclareLaunchArgument(
        'world_package', default_value='lab5',
        description="Specify the package containing the world file"
    )
    declare_world_name = DeclareLaunchArgument(
        'world_name', default_value='',
        description="Specify world name, we'll convert to full path"
    )
    declare_debug = DeclareLaunchArgument(
        'debug', default_value='False',
        choices=['True', 'False'],
        description='If debug, start the gazebo world in a gdb session in an xter
    )

    start_gazebo_server_cmd = OpaqueFunction(function=start_gzserver)

    start_gazebo_client_cmd = ExecuteProcess(
        cmd=['gzclient'], output='screen')

    # Create the launch description and populate
    ld = LaunchDescription()

    ld.add_action(declare_world_package)
    ld.add_action(declare_world_name)
    ld.add_action(declare_debug)

    ld.add_action(start_gazebo_server_cmd)
    ld.add_action(start_gazebo_client_cmd)

    return ld
```

Jako kolejny przebudowano plik uruchomieniowy *tiago_gazebo.launch.py* tworząc plik *stero_navigation.launch.py* w tym pliku zmieniono wykorzystywany plik uruchomieniowy *pal_gazebo.launch.py* na stworzyną przez nas wersję, która pozwoliła na wybranie odpowiedniego pakietu dzięki czemu umożliwione zostało uruchomienie własnego świata z mieszkaniem oraz korytarza oraz każdego innego umieszczonego w odpowiednim folderze w pakiecie. Działającą symulację przedstawiono dla korytarza na zrzucie ekranu 2.1 a dla mieszkania 2.2.

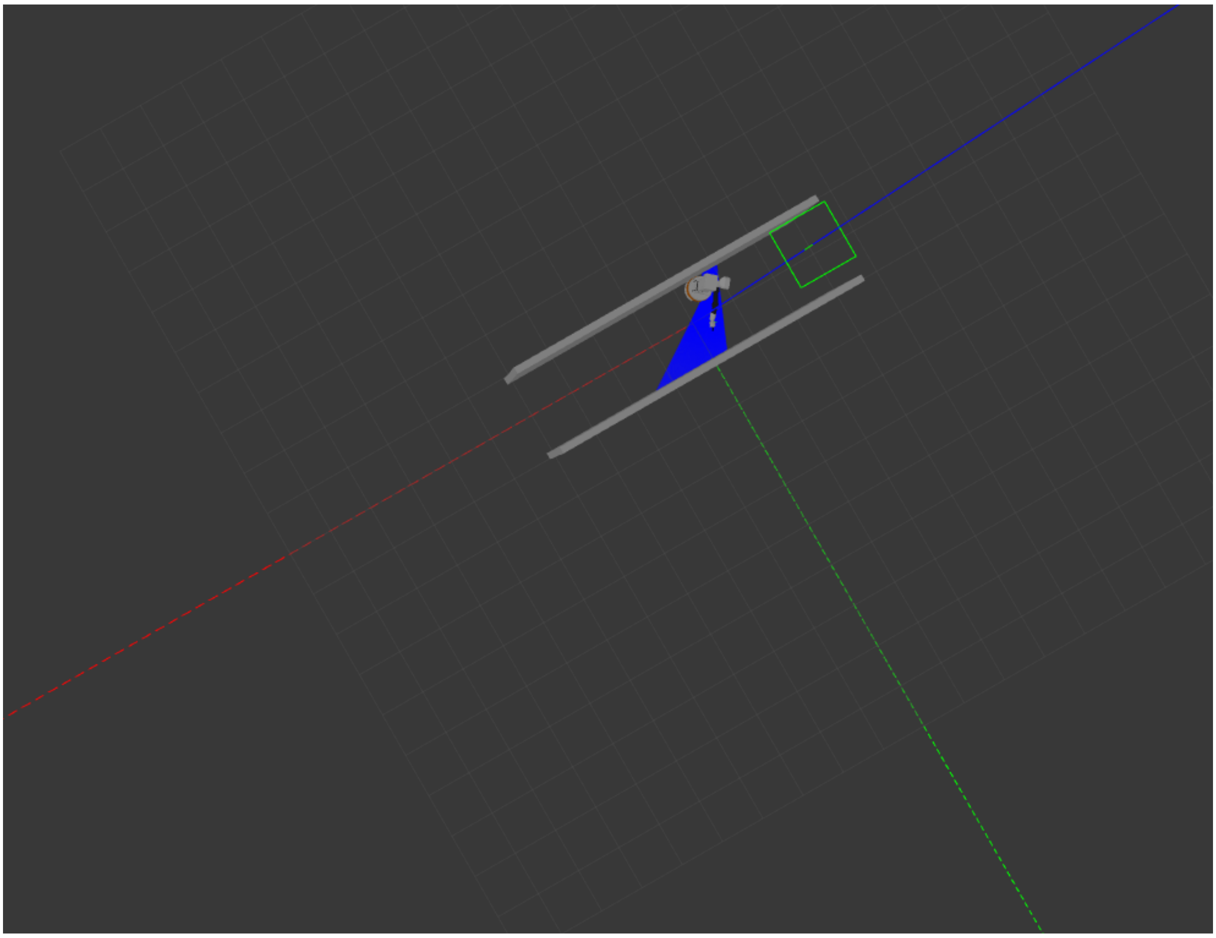
```
# Copyright (c) 2024 PAL Robotics S.L. All rights reserved.
```

```
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

import os
from os import environ, pathsep
from ament_index_python.packages import get_package_prefix
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument, SetEnvironmentVariable, SetLau
from launch.conditions import IfCondition
from launch.substitutions import LaunchConfiguration
from launch_pal.include_utils import include_scoped_launch_py_description
from launch_pal.arg_utils import LaunchArgumentsBase
from dataclasses import dataclass
from launch_pal.robot_arguments import CommonArgs
from launch_ros.actions import Node
from tiago_description.launch_arguments import TiagoArgs
from launch_pal.actions import CheckPublicSim


@dataclass(frozen=True)
class LaunchArguments(LaunchArgumentsBase):

    base_type: DeclareLaunchArgument = TiagoArgs.base_type
    has_screen: DeclareLaunchArgument = TiagoArgs.has_screen
    arm_type: DeclareLaunchArgument = TiagoArgs.arm_type
    end_effector: DeclareLaunchArgument = TiagoArgs.end_effector
    ft_sensor: DeclareLaunchArgument = TiagoArgs.ft_sensor
    wrist_model: DeclareLaunchArgument = TiagoArgs.wrist_model
    camera_model: DeclareLaunchArgument = TiagoArgs.camera_model
    laser_model: DeclareLaunchArgument = TiagoArgs.laser_model

    navigation: DeclareLaunchArgument = CommonArgs.navigation
    advanced_navigation: DeclareLaunchArgument = CommonArgs.advanced_navigation
    slam: DeclareLaunchArgument = CommonArgs.slam
    moveit: DeclareLaunchArgument = CommonArgs.moveit
    world_name: DeclareLaunchArgument = CommonArgs.world_name
    world_package: DeclareLaunchArgument = DeclareLaunchArgument(
        'world_package', default_value='lab5', description="Package containing w
    )
    namespace: DeclareLaunchArgument = CommonArgs.namespace
    tuck_arm: DeclareLaunchArgument = CommonArgs.tuck_arm
```

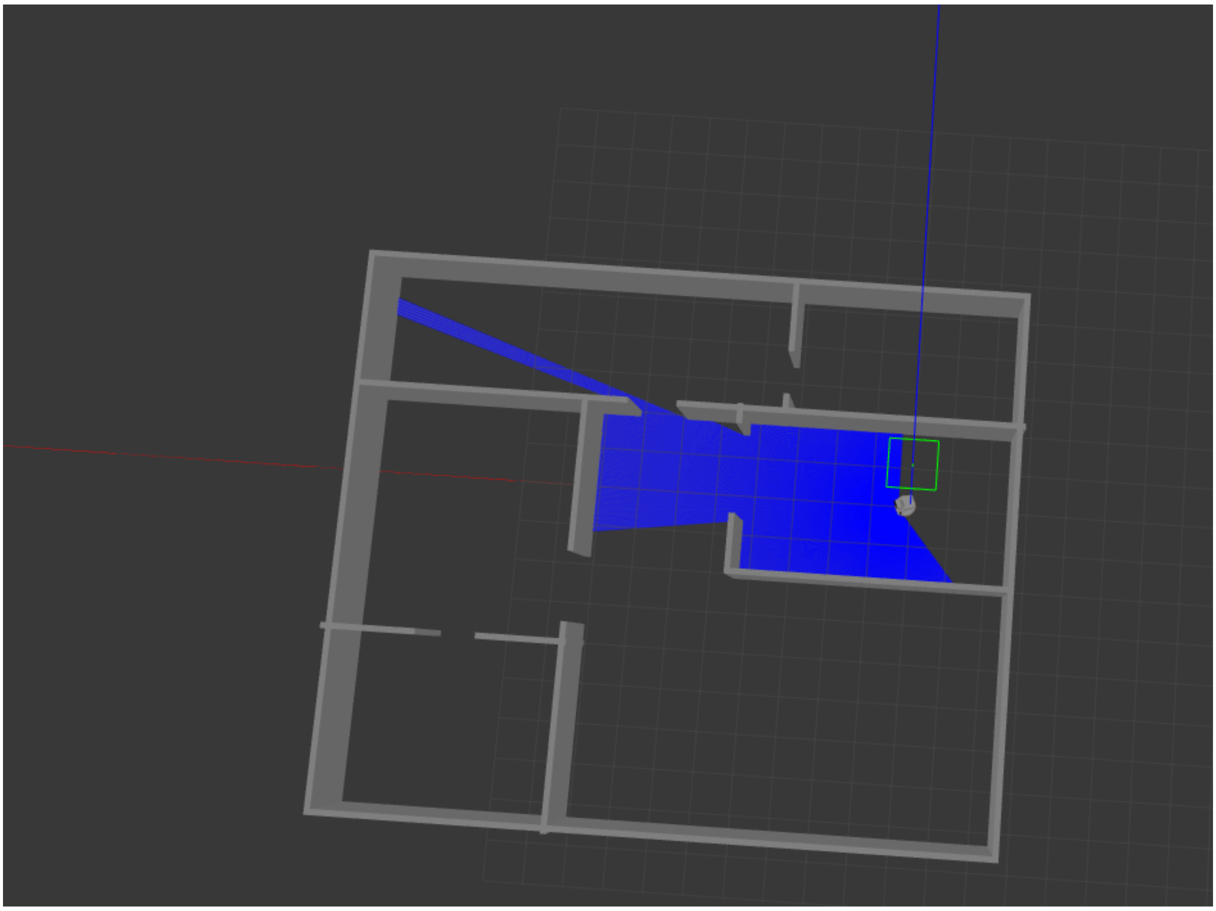
```
    is_public_sim: DeclareLaunchArgument = CommonArgs.is_public_sim
    use_grasp_fix_plugin: DeclareLaunchArgument = TiagoArgs.use_grasp_fix_plugin

def generate_launch_description():

    # Create the launch description and populate
    ld = LaunchDescription()
    launch_arguments = LaunchArguments()

    launch_arguments.add_to_launch_description(ld)

    declare_actions(ld, launch_arguments)

    return ld


def declare_actions(
    launch_description: LaunchDescription, launch_args: LaunchArguments
):
    # Set use_sim_time to True
    set_sim_time = SetLaunchConfiguration("use_sim_time", "True")
    launch_description.add_action(set_sim_time)

    # Shows error if is_public_sim is not set to True when using public simulatio
    public_sim_check = CheckPublicSim()
    launch_description.add_action(public_sim_check)

    robot_name = 'tiago'
    packages = ['tiago_description', 'pmb2_description',
                'pal_hey5_description', 'pal_gripper_description',
                'pal_robotiq_description']

    model_path = get_model_paths(packages)

    gazebo_model_path_env_var = SetEnvironmentVariable(
        'GAZEBO_MODEL_PATH', model_path)

    gazebo = include_scoped_launch_py_description(
        pkg_name='lab5',
        paths=['launch', 'gazebo.launch.py'],
        env_vars=[gazebo_model_path_env_var],
        launch_arguments={
            "world_name": launch_args.world_name,
            "model_paths": packages,
            "resource_paths": packages,
        })

    launch_description.add_action(gazebo)

    navigation = include_scoped_launch_py_description(
        pkg_name='lab5',
```

```
        paths=['launch', 'tiago_nav_bringup.launch.py'],
        launch_arguments={
            "robot_name":  robot_name,
            "is_public_sim": launch_args.is_public_sim,
            "laser":  launch_args.laser_model,
            "base_type": launch_args.base_type,
            "world_name": launch_args.world_name,
            'slam': launch_args.slam,
            'use_sim_time': LaunchConfiguration('use_sim_time'),
            "use_grasp_fix_plugin": launch_args.use_grasp_fix_plugin,
        },
        condition=IfCondition(LaunchConfiguration('navigation')))

    launch_description.add_action(navigation)

    advanced_navigation = include_scoped_launch_py_description(
        pkg_name='tiago_advanced_2dnav',
        paths=['launch', 'tiago_advanced_nav_bringup.launch.py'],
        launch_arguments={
            "base_type": launch_args.base_type,
        },
        condition=IfCondition(LaunchConfiguration('advanced_navigation')))

    launch_description.add_action(advanced_navigation)

    move_group = include_scoped_launch_py_description(
        pkg_name='tiago_moveit_config',
        paths=['launch', 'move_group.launch.py'],
        launch_arguments={
            "robot_name": robot_name,
            "use_sim_time": LaunchConfiguration("use_sim_time"),
            "namespace": launch_args.namespace,
            "base_type": launch_args.base_type,
            "arm_type": launch_args.arm_type,
            "end_effector": launch_args.end_effector,
            "ft_sensor": launch_args.ft_sensor
        },
        condition=IfCondition(LaunchConfiguration('moveit')))

    launch_description.add_action(move_group)

    robot_spawn = include_scoped_launch_py_description(
        pkg_name='tiago_gazebo',
        paths=['launch', 'robot_spawn.launch.py'],
        launch_arguments={
            'robot_name': robot_name,
            'base_type': launch_args.base_type}
    )

    launch_description.add_action(robot_spawn)
```

```
    tiago_bringup = include_scoped_launch_py_description(
        pkg_name='tiago_bringup', paths=['launch', 'tiago_bringup.launch.py'],
        launch_arguments={
            "use_sim_time": LaunchConfiguration("use_sim_time"),
            "arm_type": launch_args.arm_type,
            "laser_model": launch_args.laser_model,
            "camera_model": launch_args.camera_model,
            "base_type": launch_args.base_type,
            "wrist_model": launch_args.wrist_model,
            "ft_sensor": launch_args.ft_sensor,
            "end_effector": launch_args.end_effector,
            "has_screen": launch_args.has_screen,
            "is_public_sim": launch_args.is_public_sim,
            "use_grasp_fix_plugin": launch_args.use_grasp_fix_plugin,
        }
    )

    launch_description.add_action(tiago_bringup)

    tuck_arm = Node(package='tiago_gazebo',
                    executable='tuck_arm.py',
                    emulate_tty=True,
                    output='both',
                    condition=IfCondition(LaunchConfiguration('tuck_arm')))

    launch_description.add_action(tuck_arm)

    return


def get_model_paths(packages_names):
    model_paths = ""
    for package_name in packages_names:
        if model_paths != "":
            model_paths += pathsep

        package_path = get_package_prefix(package_name)
        model_path = os.path.join(package_path, "share")

        model_paths += model_path

    if 'GAZEBO_MODEL_PATH' in environ:
        model_paths += pathsep + environ['GAZEBO_MODEL_PATH']

    return model_paths
```



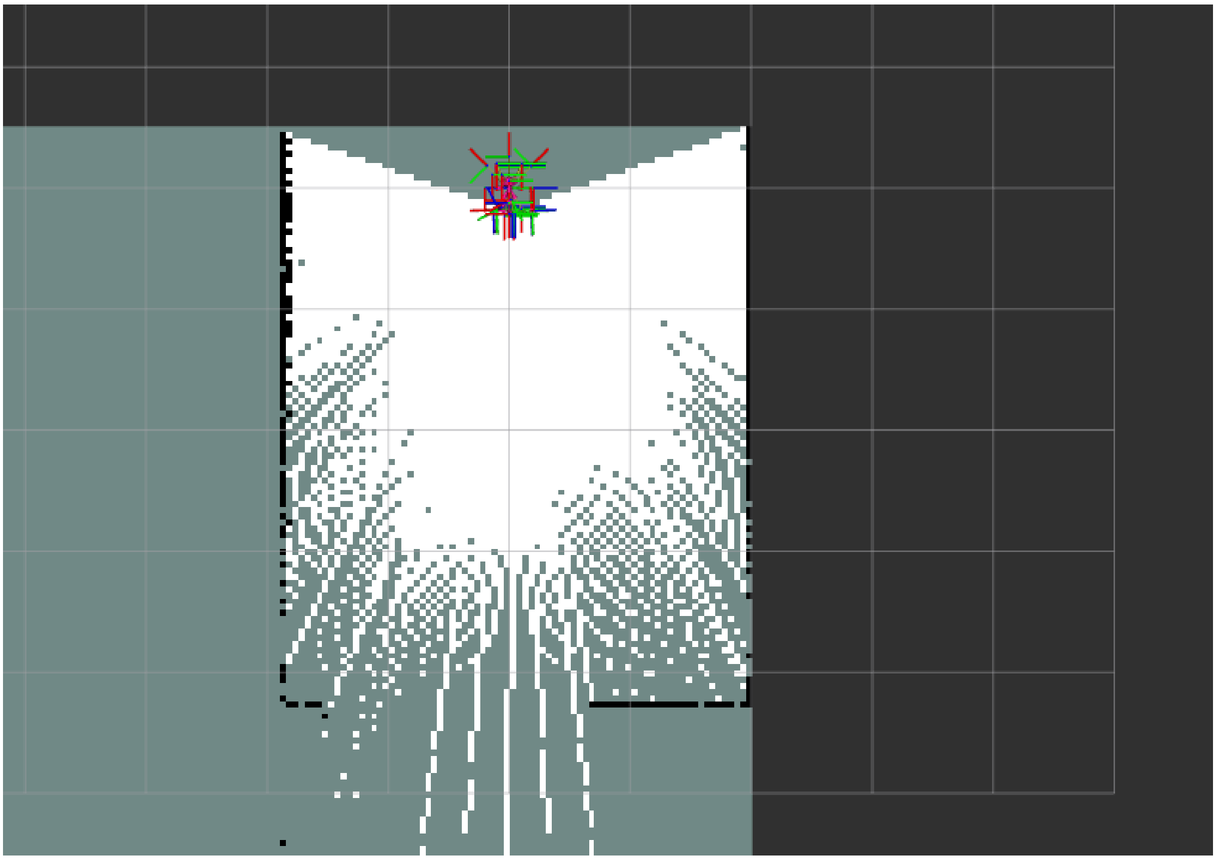
Rys. 2.1. Świat korytarz w symulacji Tiago w Gazebo



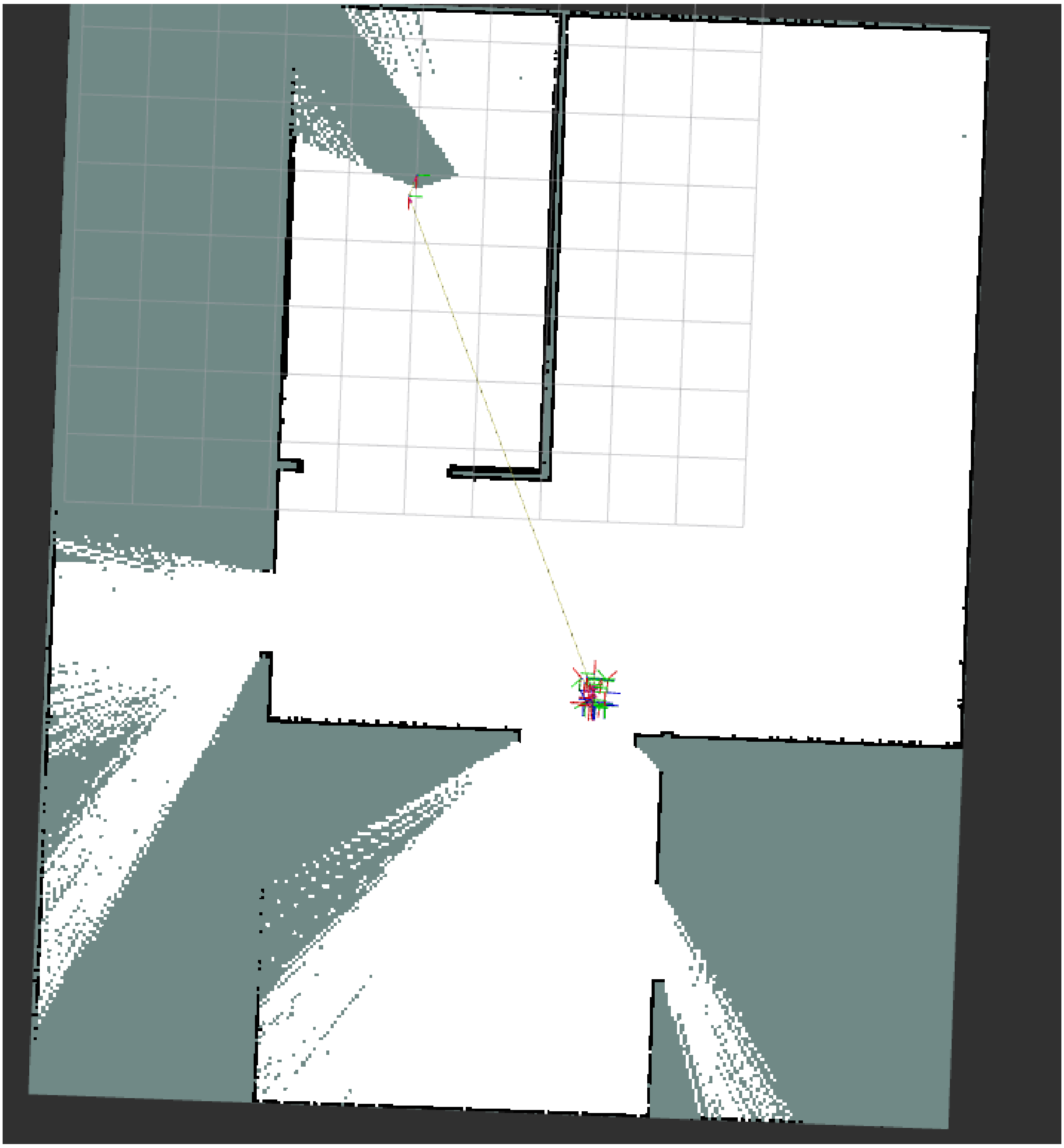
Rys. 2.2. Świat mieszkanie w symulacji Tiago w Gazebo

3. Budowanie mapy środowiska (SLAM)

W celu zbudowania mapy korzystają ze SLAM wykorzystano stworzony plik uruchomieniowy dodając do niego argument $slam := True$ oraz *teleop_twist_keyboard.py* z pakietu *teleop_twist_keyboard*, który pozwolił na sterowanie robotem przy pomocy klawiatury tak aby zbudować całą mapę. Proces te przedstawiono na zrzutach ekranu 3.1, 3.2. Strukturę węzłów algorytmu SLAM przedstawiono na zrzucie ekranu 3.3.



Rys. 3.1. Początek budowania mapy mieszkania



Rys. 3.2. W trakcie budowania mapy mieszkania

Rys. 3.3. Graf węzłów algorytmu SLAM

4. Wczytywanie nowej mapy środowiska

W celu uruchomienia symulacji wraz z zbudowaną jego mapą stworzono własny plik uruchomieniowy *tiago_nav_bringup.launch.py* na bazie pliku o tej samej nazwie. Zmieniono go tak aby umożliwiał wczytanie mapy z folderu *maps* z pakietu tworzonego na tym laboratorium. następnie wykorzystano ten plik w pliku uruchomieniowym *stero_navigation.launch.py* tak aby wzystko zadziałało i faktycznie było wywołane. Efekt wczytania mapy przedstawiono na zrzucie ekranu 5.10.

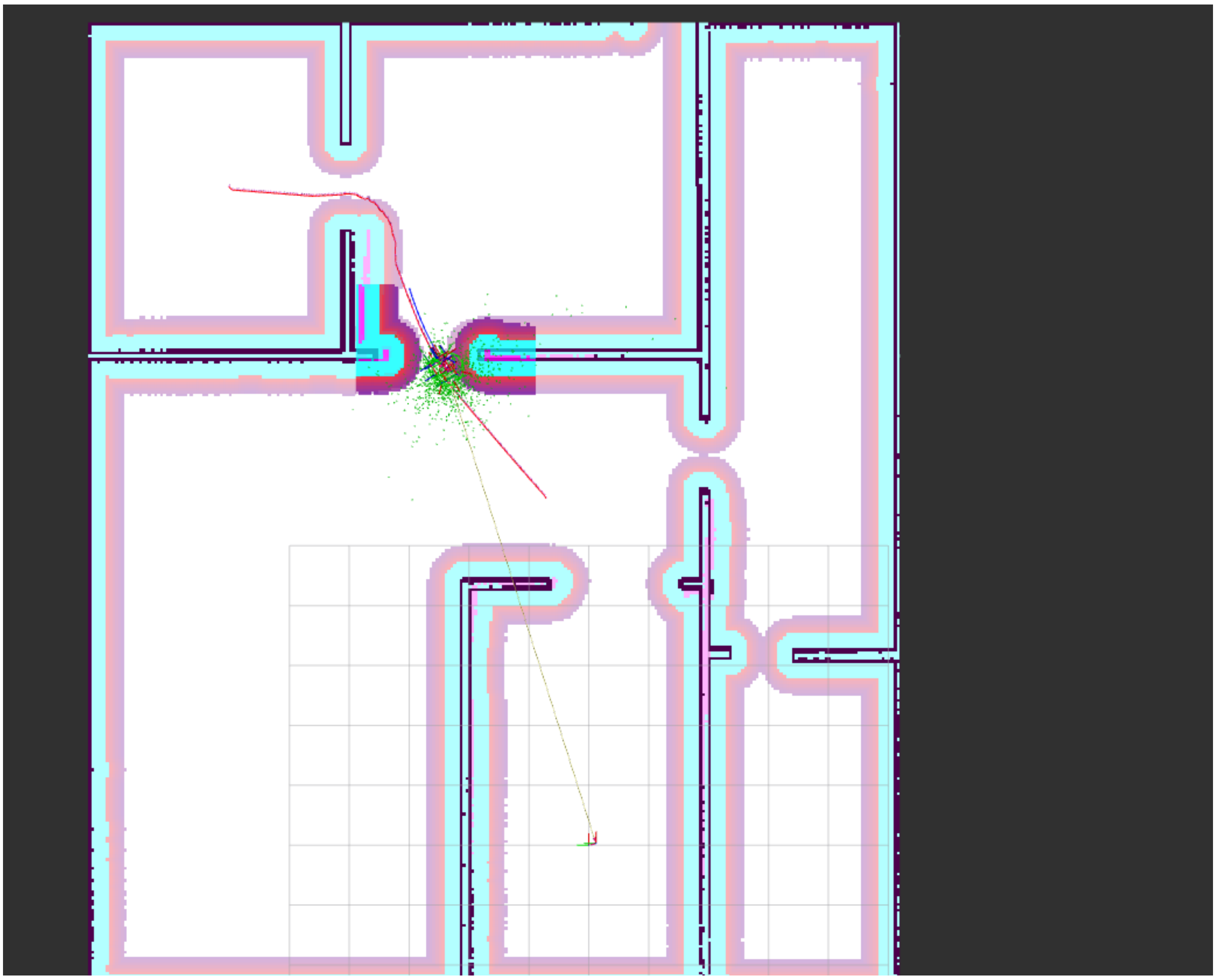
```
# Copyright (c) 2024 PAL Robotics S.L. All rights reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

import os
from ament_index_python.packages import get_package_share_directory

from launch import LaunchDescription
from launch.actions import (
    DeclareLaunchArgument,
    OpaqueFunction,
)

from launch_pal.robot_arguments import CommonArgs
from launch_pal.arg_utils import LaunchArgumentsBase
from dataclasses import dataclass
from tiago_description.launch_arguments import TiagoArgs
from launch_pal.include_utils import include_scoped_launch_py_description
from launch_pal.arg_utils import read_launch_argument
from launch.conditions import IfCondition, UnlessCondition
from launch.substitutions import LaunchConfiguration
from launch_ros.actions import Node


@dataclass(frozen=True)
class LaunchArguments(LaunchArgumentsBase):
```

```
    is_public_sim: DeclareLaunchArgument = CommonArgs.is_public_sim
    world_name: DeclareLaunchArgument = CommonArgs.world_name
    base_type: DeclareLaunchArgument = TiagoArgs.base_type
    slam: DeclareLaunchArgument = CommonArgs.slam


def generate_launch_description():

    # Create the launch description and populate
    ld = LaunchDescription()
    launch_arguments = LaunchArguments()

    launch_arguments.add_to_launch_description(ld)

    declare_actions(ld, launch_arguments)

    return ld


def public_nav_function(context, *args, **kwargs):
    base_type = read_launch_argument("base_type", context)
    world_name = read_launch_argument("world_name", context)
    actions = []
    tiago_2dnav = get_package_share_directory("tiago_2dnav")
    param_file = os.path.join(tiago_2dnav, "params", "tiago_" + base_type + "_na

    pal_maps = get_package_share_directory("lab5")

    map_path = os.path.join(pal_maps, "maps", world_name, "map.yaml")
    rviz_config_file = os.path.join(tiago_2dnav, "config", "rviz", "navigation.r

    nav_bringup_launch = include_scoped_launch_py_description(
        pkg_name="nav2_bringup",
        paths=['launch', "navigation_launch.py"],
        launch_arguments={
            "params_file": param_file,
            "use_sim_time": "True"
        }
    )

    slam_bringup_launch = include_scoped_launch_py_description(
        pkg_name="nav2_bringup",
        paths=["launch", "slam_launch.py"],
        launch_arguments={
            "params_file": param_file,
            "use_sim_time": "True"
        },
        condition=IfCondition(LaunchConfiguration("slam")),

    )
```

```
    loc_bringup_launch = include_scoped_launch_py_description(
        pkg_name="nav2_bringup",
        paths=["launch", "localization_launch.py"],
        launch_arguments={
            "params_file": param_file,
            "map": map_path,
            "use_sim_time": "True"
        },
        condition=UnlessCondition(LaunchConfiguration("slam")),
    )
    rviz_bringup_launch = include_scoped_launch_py_description(
        pkg_name="nav2_bringup",
        paths=["launch", "rviz_launch.py"],
        launch_arguments={
            "rviz": rviz_config_file
        },
    )

    actions.append(nav_bringup_launch)
    actions.append(slam_bringup_launch)
    actions.append(loc_bringup_launch)
    actions.append(rviz_bringup_launch)
    return actions


def private_nav_function(context, *args, **kwargs):
    base_type = read_launch_argument("base_type", context)
    actions = []
    tiago_2dnav = get_package_share_directory("tiago_2dnav")

    remappings_file = os.path.join(
        tiago_2dnav, "params", "tiago_" + base_type + "_remappings_sim.yaml")

    nav_bringup_launch = include_scoped_launch_py_description(
        pkg_name="pal_nav2_bringup",
        paths=['launch', "nav_bringup.launch.py"],
        launch_arguments={
            "params_pkg": "tiago_2dnav",
            "params_file": "tiago_" + base_type + "_nav.yaml",
            "robot_name": "tiago",
            "remappings_file": remappings_file,
        })
    slam_bringup_launch = include_scoped_launch_py_description(
        pkg_name="pal_nav2_bringup",
        paths=["launch", "nav_bringup.launch.py"],
        launch_arguments={
            "params_pkg": "tiago_2dnav",
            "params_file": "tiago_slam.yaml",
            "robot_name": "tiago",
            "remappings_file": remappings_file,
        },
```

```
        condition=IfCondition(LaunchConfiguration("slam"))
    )

    loc_bringup_launch = include_scoped_launch_py_description(
        pkg_name="pal_nav2_bringup",
        paths=["launch", "nav_bringup.launch.py"],
        launch_arguments={
            "params_pkg": "tiago_2dnav",
            "params_file": "tiago_" + base_type + "_loc.yaml",
            "robot_name": "tiago",
            "remappings_file": remappings_file,
        },
        condition=UnlessCondition(LaunchConfiguration("slam"))
    )

    laser_bringup_launch = include_scoped_launch_py_description(
        pkg_name="pal_nav2_bringup",
        paths=["launch", "nav_bringup.launch.py"],
        launch_arguments={
            "params_pkg": "tiago_laser_sensors",
            "params_file": base_type + "_laser_pipeline_sim.yaml",
            "robot_name": "tiago",
            "remappings_file": remappings_file,
        }
    )

    rviz_node = Node(
        package="rviz2",
        executable="rviz2",
        arguments=["-d", os.path.join(
            tiago_2dnav,
            "config",
            "rviz",
            "navigation.rviz",
        )],
        output="screen",
    )

    actions.append(nav_bringup_launch)
    actions.append(slam_bringup_launch)
    actions.append(loc_bringup_launch)
    actions.append(laser_bringup_launch)
    actions.append(rviz_node)

    return actions


def declare_actions(launch_description: LaunchDescription, launch_args: LaunchAr

    launch_description.add_action(
        OpaqueFunction(
```

```
                function=public_nav_function,
                condition=IfCondition(LaunchConfiguration("is_public_sim"))
            )
        )

        launch_description.add_action(
            OpaqueFunction(
                function=private_nav_function,
                condition=UnlessCondition(LaunchConfiguration("is_public_sim"))
            )
        )
```



Rys. 4.1. Wczytana mapa mieszkania w rviz-ie

5. Badanie producenckiego systemu nawigacji w świecie 'mieszkanie'

5.1. Węzeł nawigujący po pomieszczeniach w mieszkaniu

Nowy węzeł *simple_nav*, który pobierał z konsoli nazwę pomieszczenia, a następnie wysyłał odpowiednią pozycję do systemu nawigacji zaimplementowano w pythonie, co przedstawiono poniżej, zaś działanie tego systemu zaprezentowano na kolejnych zrzutach ekranu **??**, **??**...

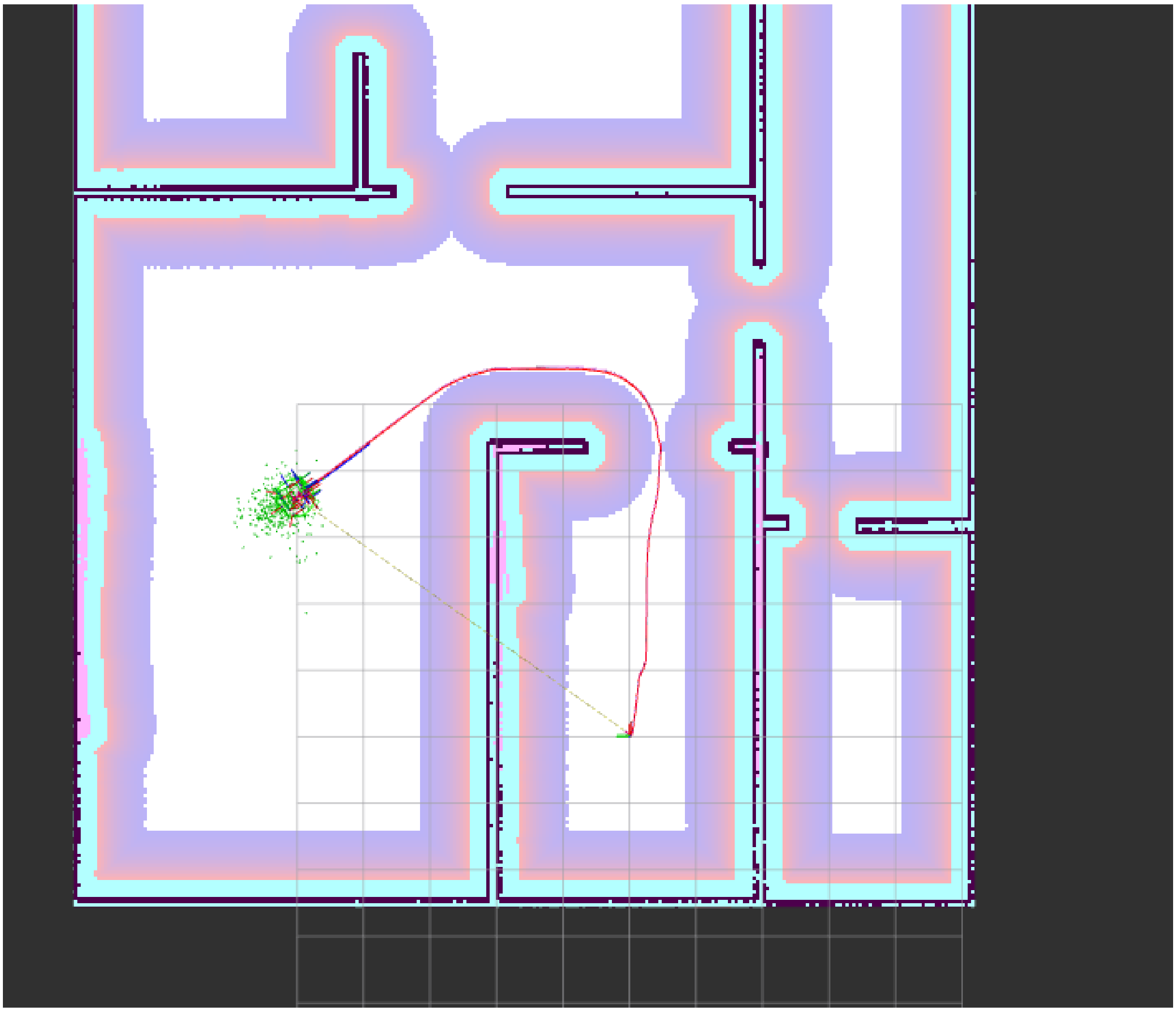
```
import rclpy
from rclpy.node import Node
from nav2_msgs.action import NavigateToPose
from rclpy.action import ActionClient
from geometry_msgs.msg import PoseStamped
from geometry_msgs.msg import Quaternion

class NavigationNode(Node):
    def __init__(self):
        super().__init__('navigation_node')
        self._action_client = ActionClient(self, NavigateToPose, 'navigate_to_pos
        self.get_logger().info("Navigation node started, waiting for action serv

    def send_navigation_goal(self, x, y, z, qx, qy, qz, qw):
        # Tworzymy komunikat typu PoseStamped
        goal_pose = PoseStamped()
        goal_pose.header.frame_id = 'map'  # Układ odniesienia
        goal_pose.header.stamp = self.get_clock().now().to_msg()

        # Ustawiamy pozycję i orientację celu
        goal_pose.pose.position.x = x
        goal_pose.pose.position.y = y
        goal_pose.pose.position.z = z
        goal_pose.pose.orientation = Quaternion(x=qx, y=qy, z=qz, w=qw)

        goal = NavigateToPose.Goal()
        goal.pose = goal_pose

        # Wysyłamy cel
        self._action_client.wait_for_server()
        self._action_client.send_goal_async(goal)

    def get_pose_for_room(self, room_name):
        """
        Funkcja zwraca pozycję (x, y, z) oraz orientację (quaternion) w zależnoś
        """
        room_name = room_name.lower()
```

```
        # Definiowanie pozycji nawigacji dla każdego pokoju
        if room_name == 'hall':
            return 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0
        elif room_name == 'kitchen':
            return 3.5, -4.5, 0.0, 0.0, 0.0, 0.0, 1.0
        elif room_name == 'wc':
            return 1.0, -4.0, 0.0, 0.0, 0.0, 0.0, 1.0
        elif room_name == 'living':
            return 2.0, 7.0, 0.0, 0.0, 0.0, 0.0, 1.0
        elif room_name == 'bedroom':
            return 9.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0
        elif room_name == 'storage':
            return 11.0, 6.0, 0.0, 0.0, 0.0, 0.0, 1.0
        else:
            return None  # Jeśli nie rozpoznamy pokoju


def main(args=None):
    rclpy.init(args=args)

    navigation_node = NavigationNode()

    # Pętla do wyboru pokoju przez użytkownika
    while True:
        place = input("Wprowadź nazwę miejsca (lub wpisz 'koniec' aby zakończyć)

        if place.lower() == 'koniec':
            break

        # Pobieramy pozycję dla wybranego pokoju
        position = navigation_node.get_pose_for_room(place)

        if position:
            navigation_node.send_navigation_goal(*position)
            print(f"Wysyłam do pokoju: {place}")
        else:
            print(f"Nie rozpoznałem pokoju o nazwie {place}. Proszę spróbować po

    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

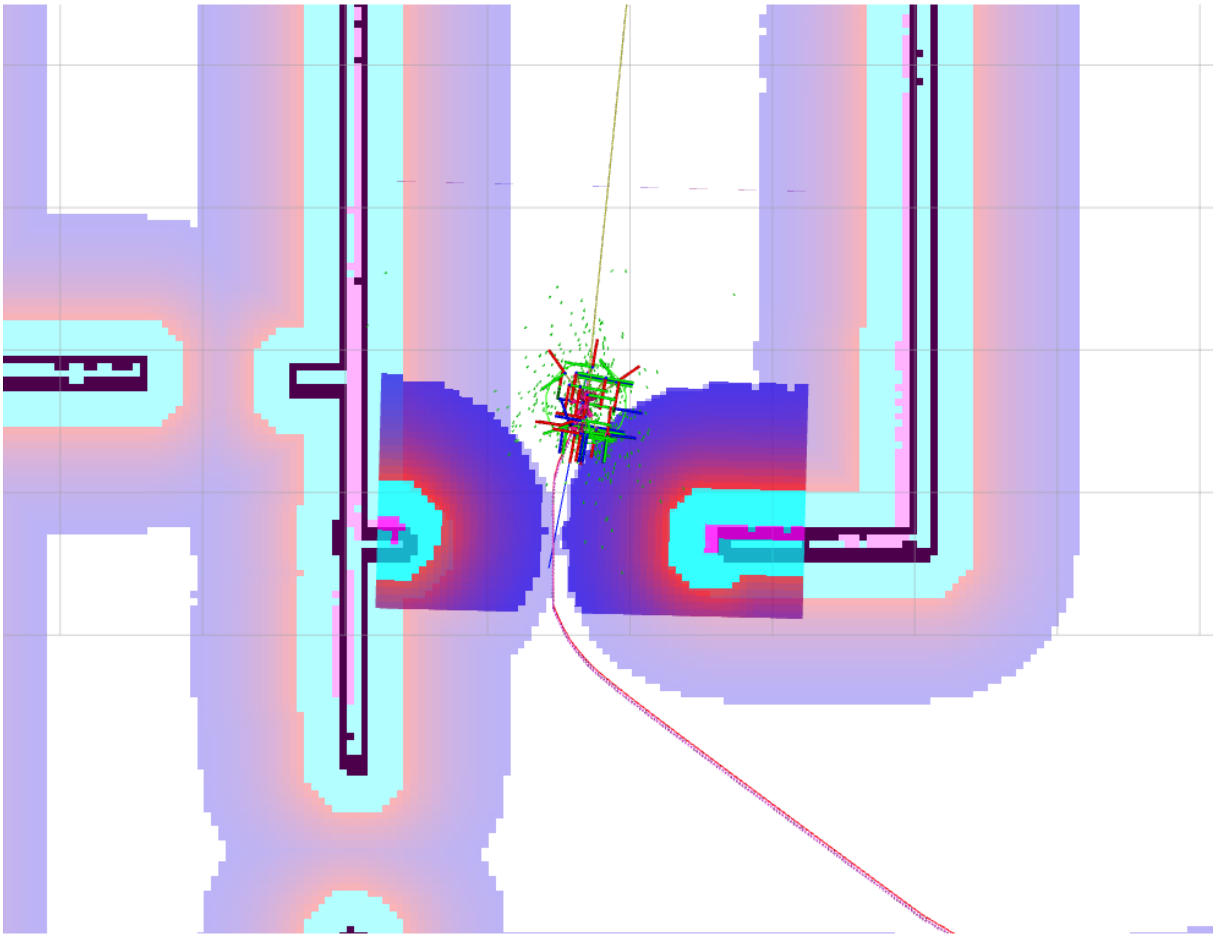
5.2. Mapy kosztów

5.2.1. Inflation radius

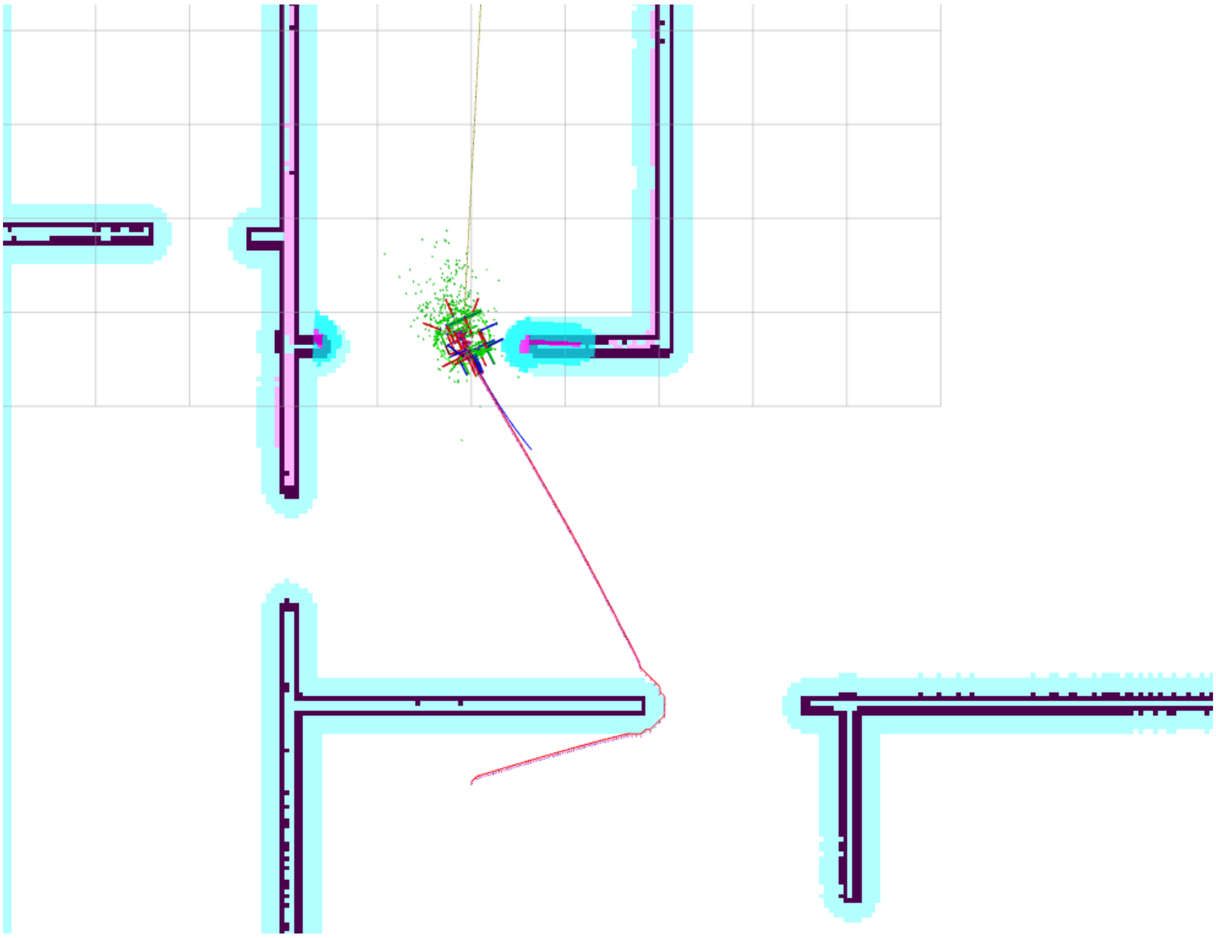
W celu zbadania wpływu parametru zmienione jego wartość lokalną i globalną z domyślnych 0,55 na 1, taka znaczące zwiększenie promienia sprawiło, że robot zaczął poruszać się środkiem szerszych drzwi, natomiast przez węższe nie był w stanie przejechać, z kolei po zmniejszeniu parametru na 0,2 Tiago na ostrzejszych zakrętach zahaczał o framugi. Dobierając ten parametr należy zadbać, aby nie był mniejszy niż promień bazy robota i innych jego elementów.



Rys. 5.1. Inflation radius równny 1



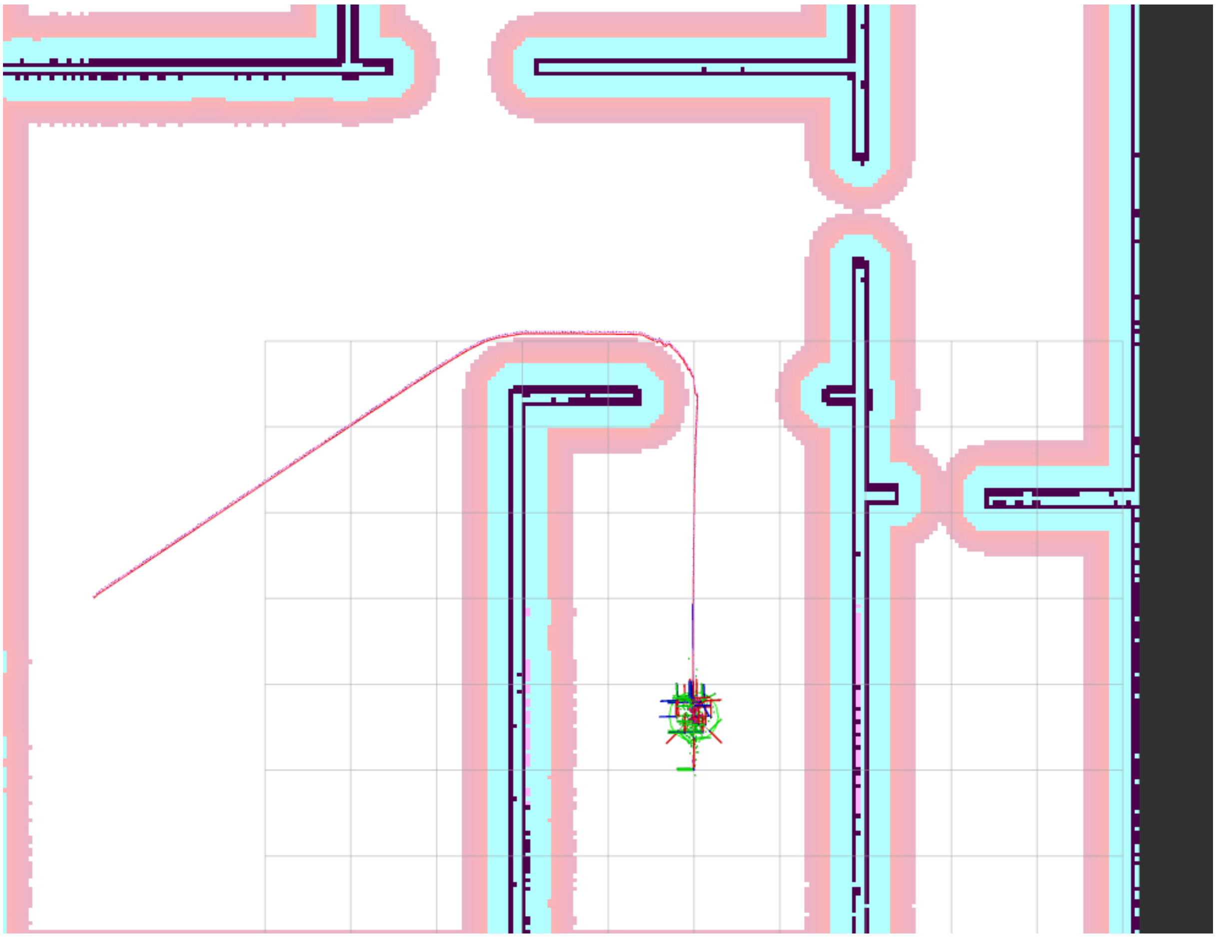
Rys. 5.2. Inflation radius równy 1



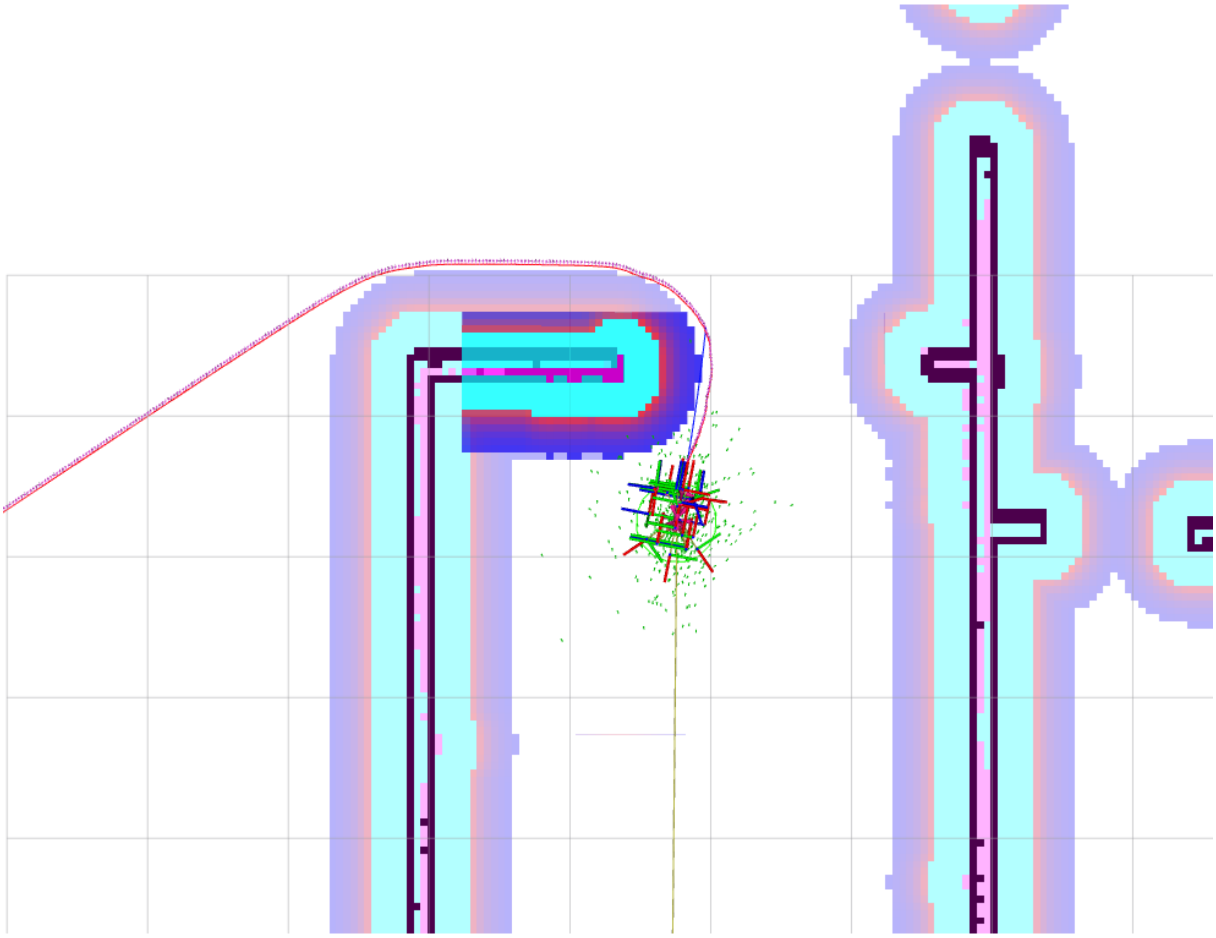
Rys. 5.3. Inflation radius równy 0,1

5.2.2. Cost scaling factor

Domyślnie ustawiony na 3 parametr cost scaling factor zmieniono na 1, zaobserwowano, że algorytm doboru trasy przypisuje mniejszy koszt obszarom przy ścinanie analogicznie po jego zwiększeniu na 10 ten koszt również jest zwiększany.



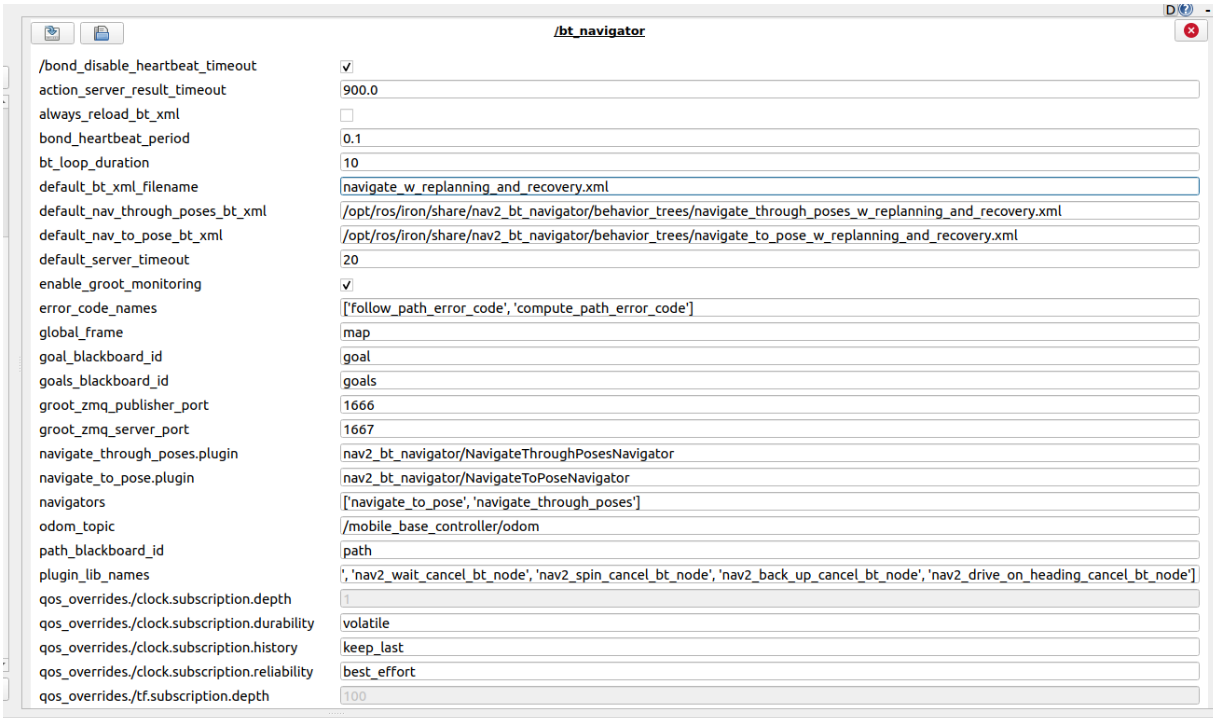
Rys. 5.4. Zmniejszony cost scalling factor do 1



Rys. 5.5. Zwiększony cost scalling factor do 10

5.3. Behavior tree

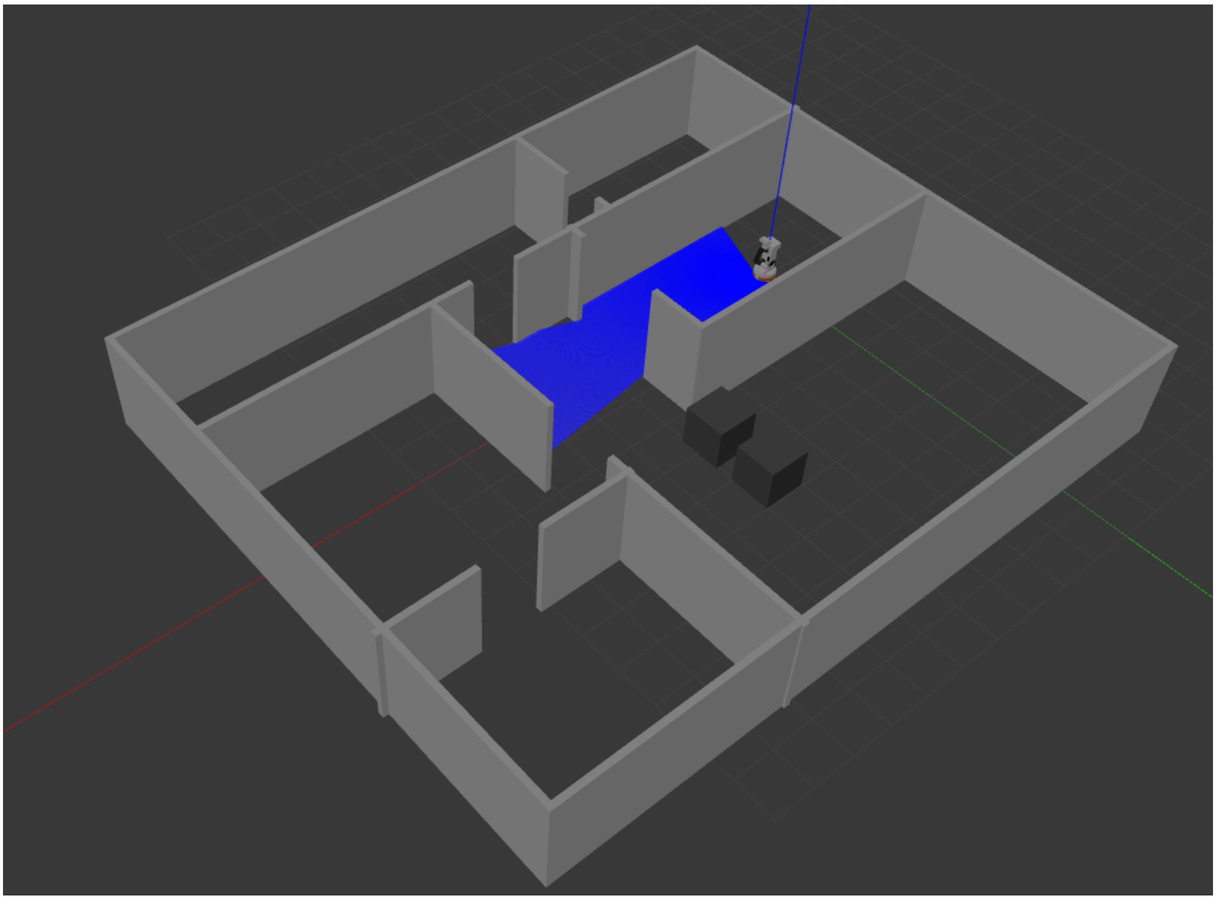
Domyślnie ustawione drzewo zachowań navigate_w_replanning_and_recovery.xml w razie zgubienia się w trakcie nawigacji możliwe jest wycofanie się robota, jego obrót w miejscu w celu zmapowania otoczenia oraz ponowne wygenerowanie ścieżki. Przetestowano drzewo zachowań navigate_to_pose.xml, które nie zapewnia ponownego planowania ścieżki, przez co po napotkaniu nieoczekiwanych danych z lidara robot się zatrzymywał i przerywał nawigację.



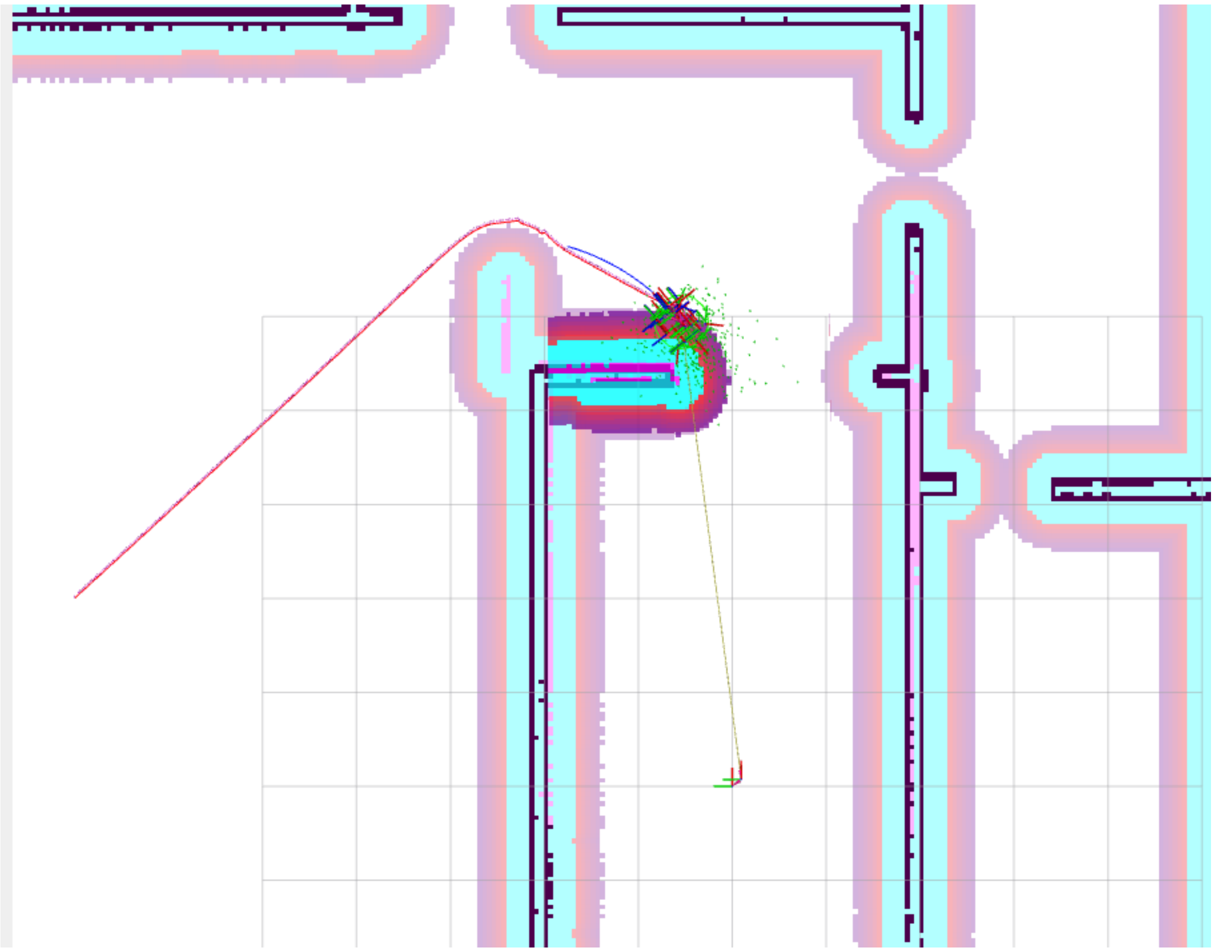
Rys. 5.6. Sekwncja zachowań robota w przypadku zgubenia trasy

5.4. Omijanie przeszkód

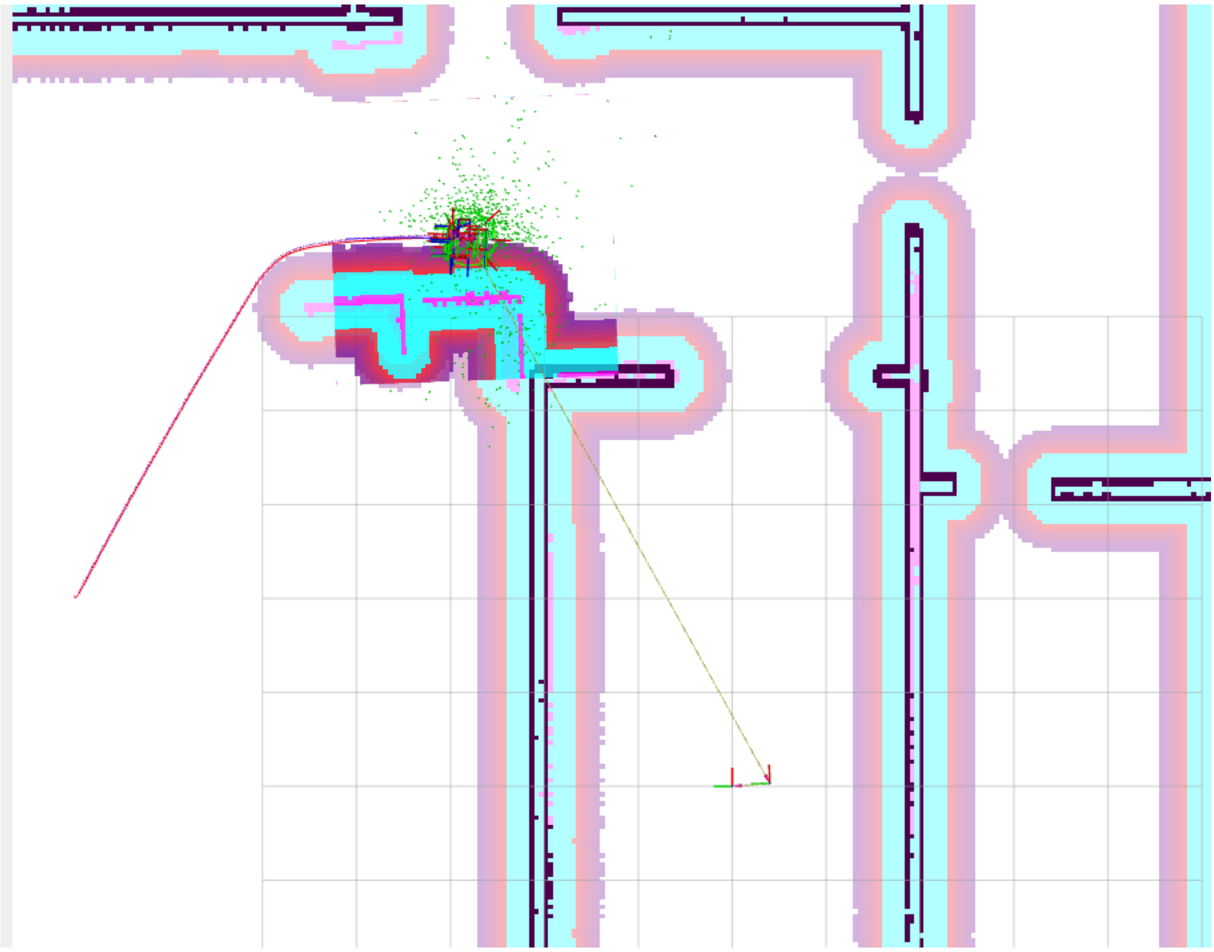
W środowisku Gazebo dodano przeszkody w postaci sześciennych bloków, robot w miarę ich wykrywania aktualizował swoją trasę poprzez wyznaczenie nowej ścieżki.



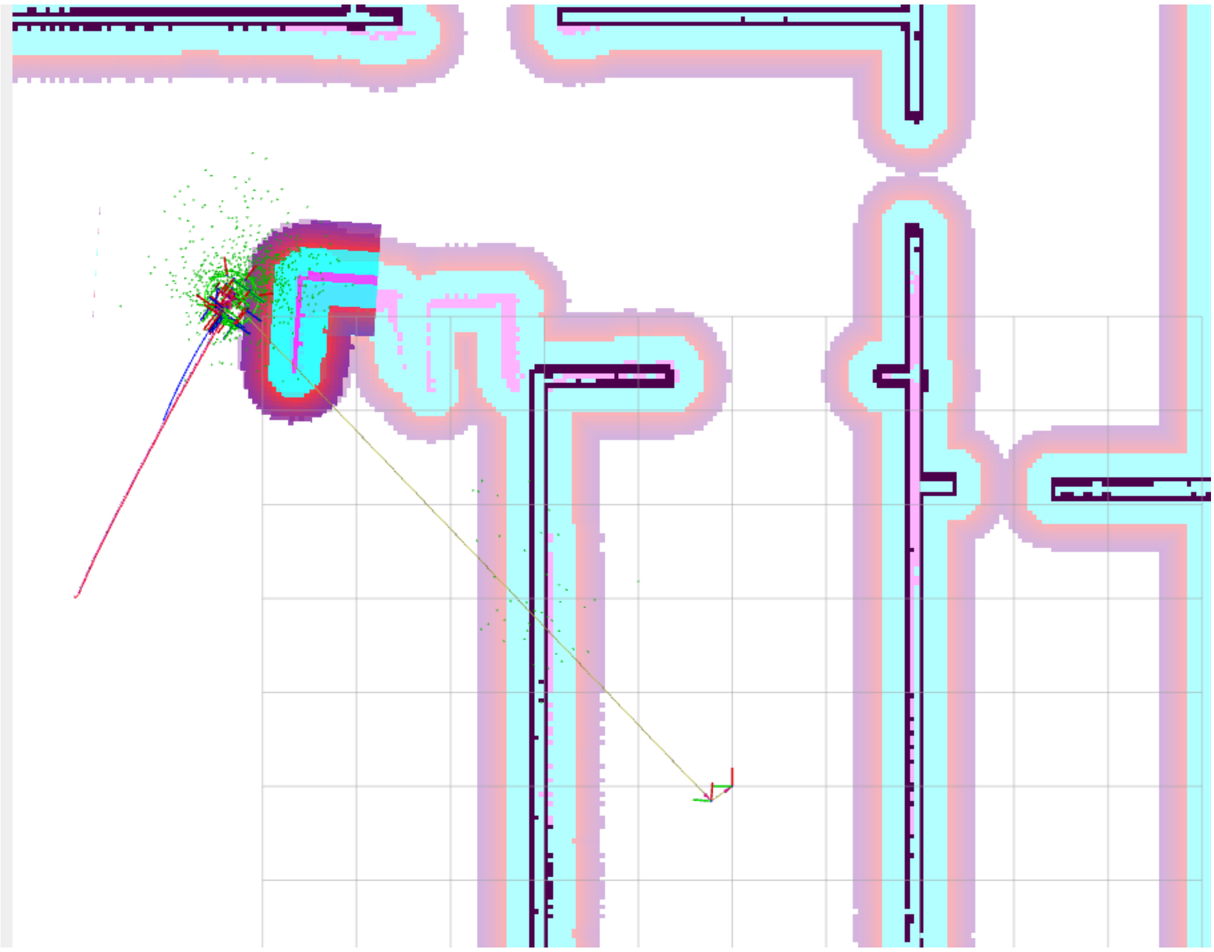
Rys. 5.7. Rozmieszczenie przeszkód w mieszkaniu



Rys. 5.8. Wykrycie przeszkody i zmiana trasy



Rys. 5.9. Wykrycie przeszkody i zmiana trasy



Rys. 5.10. Wykrycie przeszkody i zmiana trasy

6. Węzeł *folow_waypoint_nav_server*

W pakiecie *nav_msgs_stero* stworzono akcję *WaypintFollow*, która otrzymuje w postaci requesta listę punktów przez które ma przejechać robot, a zwraca feedback jako procent przejechanej trasy. Pozycja głowy była zadawana na temacie */head_controller/joint_trajectory*

```
std_msgs/Header header
geometry_msgs/Point[] waypoints
---
int32 status
---
float64 percentage_completed
```

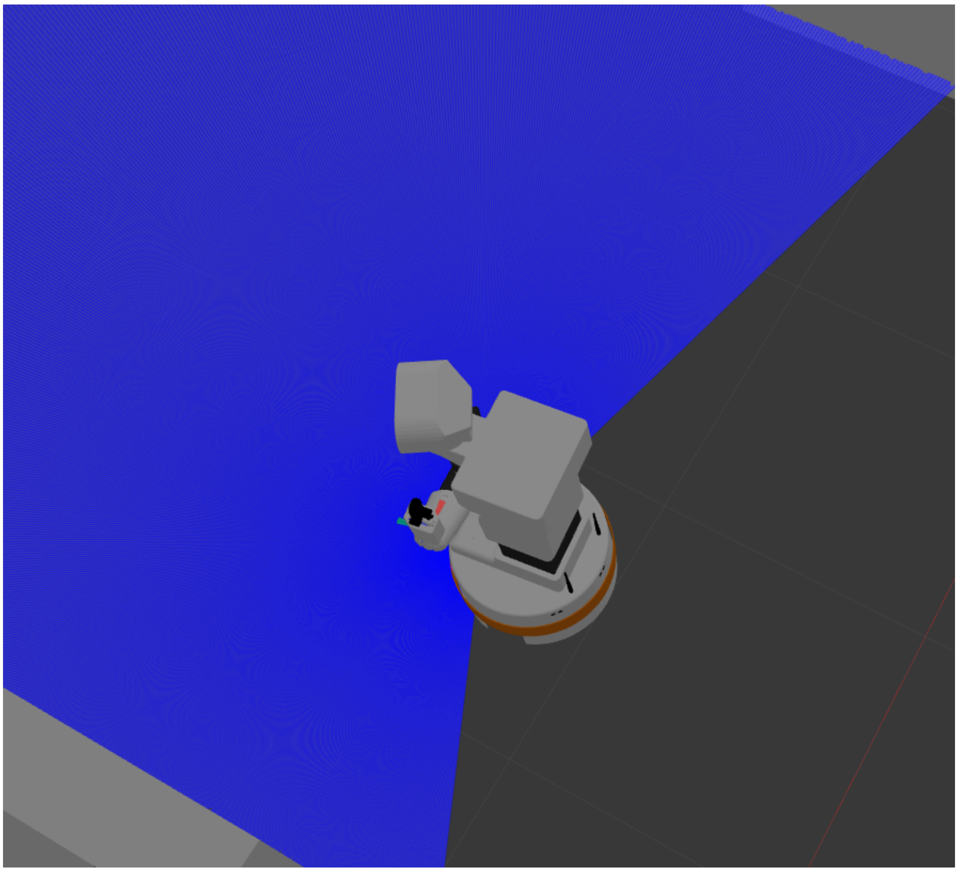
Następnie stworzono węzeł *follow_waypoint_nav_server* będący serwerem powyższej akcji. Do nawigacji przez listę punktów skorzystano z metody *followWaypoints()* Simple Comander API. Aby wyliczyć postęp wykrzystano pozycję robota pobieraną z tematu */amcl_pose*. Kolejną funkcjonalnością jest obrót głowy robota zgodny z kierunkiem obrotu bazy robota, zrealizowano go poprzez zadawanie kąta obrotu głowy odpowiadającego różnicy orientacji względem osi z w obecnej i poprzedniej chwili.

```
import rclpy
from rclpy.action import ActionServer
from rclpy.node import Node
from nav_msgs_stero.action import WaypointFollow
from geometry_msgs.msg import PoseStamped, PoseWithCovarianceStamped,
Point, Pose
from trajectory_msgs.msg import JointTrajectory, JointTrajectoryPoint
from nav2_simple_commander.robot_navigator import BasicNavigator
from nav_msgs.msg import Path
import math
from rclpy.callback_groups import ReentrantCallbackGroup
from rclpy.executors import MultiThreadedExecutor
from rclpy.qos import QoSProfile, ReliabilityPolicy, HistoryPolicy,
DurabilityPolicy

custom_qos = QoSProfile(
    reliability=ReliabilityPolicy.RELIABLE,
    durability=DurabilityPolicy.TRANSIENT_LOCAL,
    history=HistoryPolicy.KEEP_LAST,
    depth=10,
)

class WaypointFollower(Node):
    def __init__(self):
        super().__init__('waypoint_follower')

        self._nav = BasicNavigator()
```

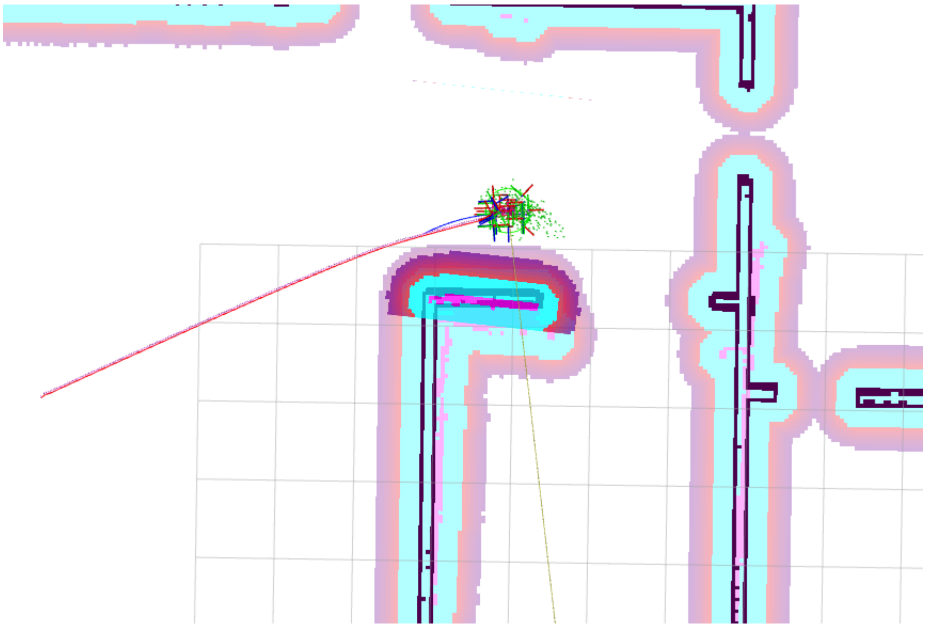
```
        self._waypoints = None
        self._waypoints_poses = None
        self._current_pose = None
        self._previous_orientation = None
        self._head_yaw = 0
        self._current_path = None
        self._current_path_length = None
        self._total_path_length = None
        self._paths = []
        self._paths_length = []

        self._action_server = ActionServer(
            self,
            WaypointFollow,
            '/waypoint_follow',
            self.execute_callback,
            callback_group=ReentrantCallbackGroup(),
        )

        self._path_subscriber = self.create_subscription(
            Path, "/plan", self._get_current_path_callback, 100,
            callback_group=ReentrantCallbackGroup())

        self._amcl_pose_subscriber = self.create_subscription(
            PoseWithCovarianceStamped, "/amcl_pose",
            self._get_current_pose_callback, qos_profile=custom_qos,
            callback_group=ReentrantCallbackGroup())

        self._head_publisher = self.create_publisher(
            JointTrajectory, "/head_controller/joint_trajectory",
            qos_profile=custom_qos)

    def execute_callback(self, goal_handle):
        feedback = WaypointFollow.Feedback()

        while self._current_pose is None:
            pass

        initial_pose = self._current_pose
        self._previous_orientation = initial_pose.pose.orientation.z
        self._nav.setInitialPose(initial_pose)

        self._waypoints = goal_handle.request.waypoints
        self.get_logger().info("Received goal with waypoints.")

        self._nav.waitUntilNav2Active()
        self._waypoints_poses = self._convert_points_to_poses(self._waypoints)
        self._waypoints_poses.insert(0, initial_pose)

        self.setup_paths_length()
```

```
        self._nav.followWaypoints(self._waypoints_poses)

        while self._current_path_length is None:
            pass
        self.get_logger().info("Path set.")
        self.get_logger().info(
            f"Total path: {self._total_path_length:.2f}")

        rate = self.create_rate(0.1)
        while not self._nav.isTaskComplete():
            self.get_logger().info(
                f"current path: {self._current_path_length:.2f}")
            waypoint = self._nav.getFeedback().current_waypoint
            driven_path = sum(self._paths_length[:waypoint]) -
            self._current_path_length

            percentage_complete = driven_path / self._total_path_length * 100
            percentage_complete = max(percentage_complete, 0)
            feedback.percentage_completed = float(percentage_complete)
            self.get_logger().info(
                f"Path completed: {percentage_complete:.2f}%")

            self.control_head_movement()
            rate.sleep()

        goal_handle.succeed()
        result = WaypointFollow.Result()
        result.status = 0
        return result

    def control_head_movement(self):
        if self._current_pose is not None and self._previous_orientation is not

            current_orientation = self._current_pose.pose.orientation.z

            delta_orientation = current_orientation - self._previous_orientation
            self.get_logger().info(f"delta orientation: {delta_orientation}")
            self._previous_orientation = current_orientation

            trajectory_msg = JointTrajectory()
            trajectory_msg.joint_names = ["head_1_joint", "head_2_joint"]
            self._head_yaw += delta_orientation

            point = JointTrajectoryPoint()
            point.positions = [self._head_yaw, 0.0]
            point.time_from_start.sec = 1
            trajectory_msg.points = [point]

            self._head_publisher.publish(trajectory_msg)
```

```
    def _convert_points_to_poses(self, points: list) -> list[PoseStamped]:
        poses = []
        for point in points:
            msg = PoseStamped()
            msg.pose.position.x = point.x
            msg.pose.position.y = point.y
            msg.pose.orientation.z = point.z
            msg.pose.orientation.w = 1.0
            msg.header.stamp = self.get_clock().now().to_msg()
            msg.header.frame_id = ”map”
            poses.append(msg)
        return poses

    def _get_current_pose_callback(self, msg: PoseWithCovarianceStamped):
        self._current_pose = PoseStamped()
        self._current_pose.pose = msg.pose.pose
        self._current_pose.header = msg.header

    def _get_current_path_callback(self, msg: Path):
        self._current_path = msg.poses
        self._current_path_length = self.calculate_path_length(self._current_path

    def calculate_path_length(self, path: list[PoseStamped]) -> float:
        length = 0
        for i in range(1, len(path)):
            dx = path[i].pose.position.x - path[i-1].pose.position.x
            dy = path[i].pose.position.y - path[i-1].pose.position.y
            length += math.sqrt(pow(dx, 2) + pow(dy, 2))
        return length

    def setup_paths_length(self):
        self._total_path_length = 0.0
        for i in range(1, len(self._waypoints_poses)):
            path = self._nav.getPath(self._waypoints_poses[i-1],
            self._waypoints_poses[i], use_start=True)
            path_length = self.calculate_path_length(path.poses)
            self._total_path_length += path_length
            self._paths_length.append(path_length)

def main(args=None):
    rclpy.init(args=args)

    waypoint_follower = WaypointFollower()
    executor = MultiThreadedExecutor()

    rclpy.spin(waypoint_follower, executor)
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```



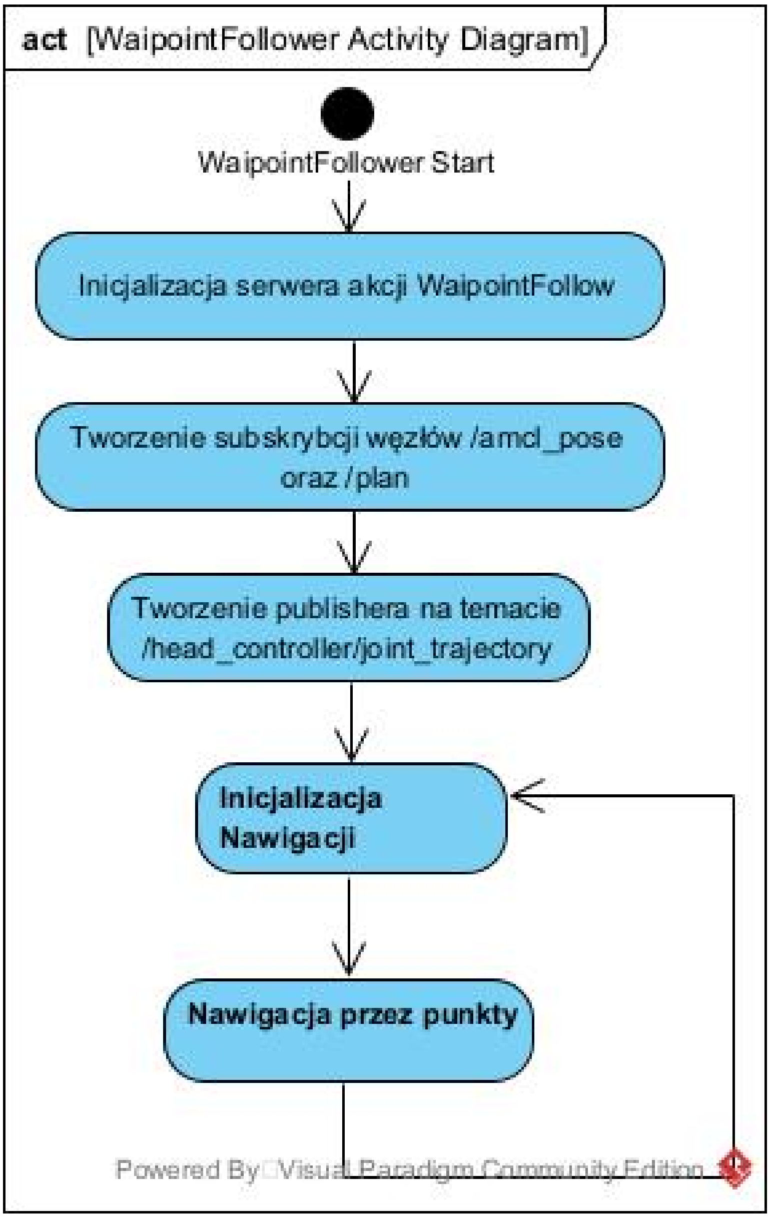
Rys. 6.1. Ruch głową robota zgodny z kierunkiem obrotu bazy mobilinej



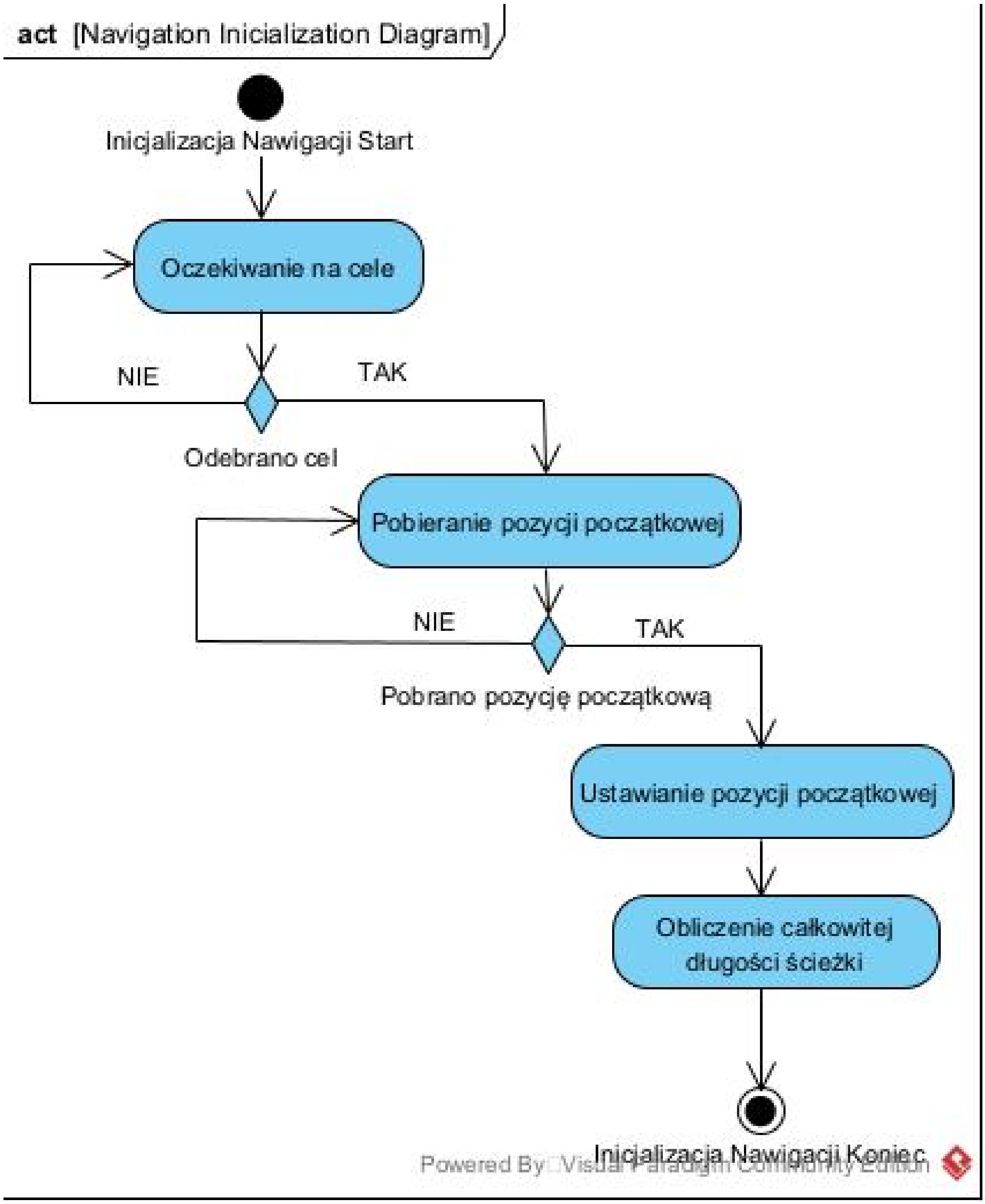
Rys. 6.2. Wizualizacja położenia oraz trasy robota

7. Diagramy akcji

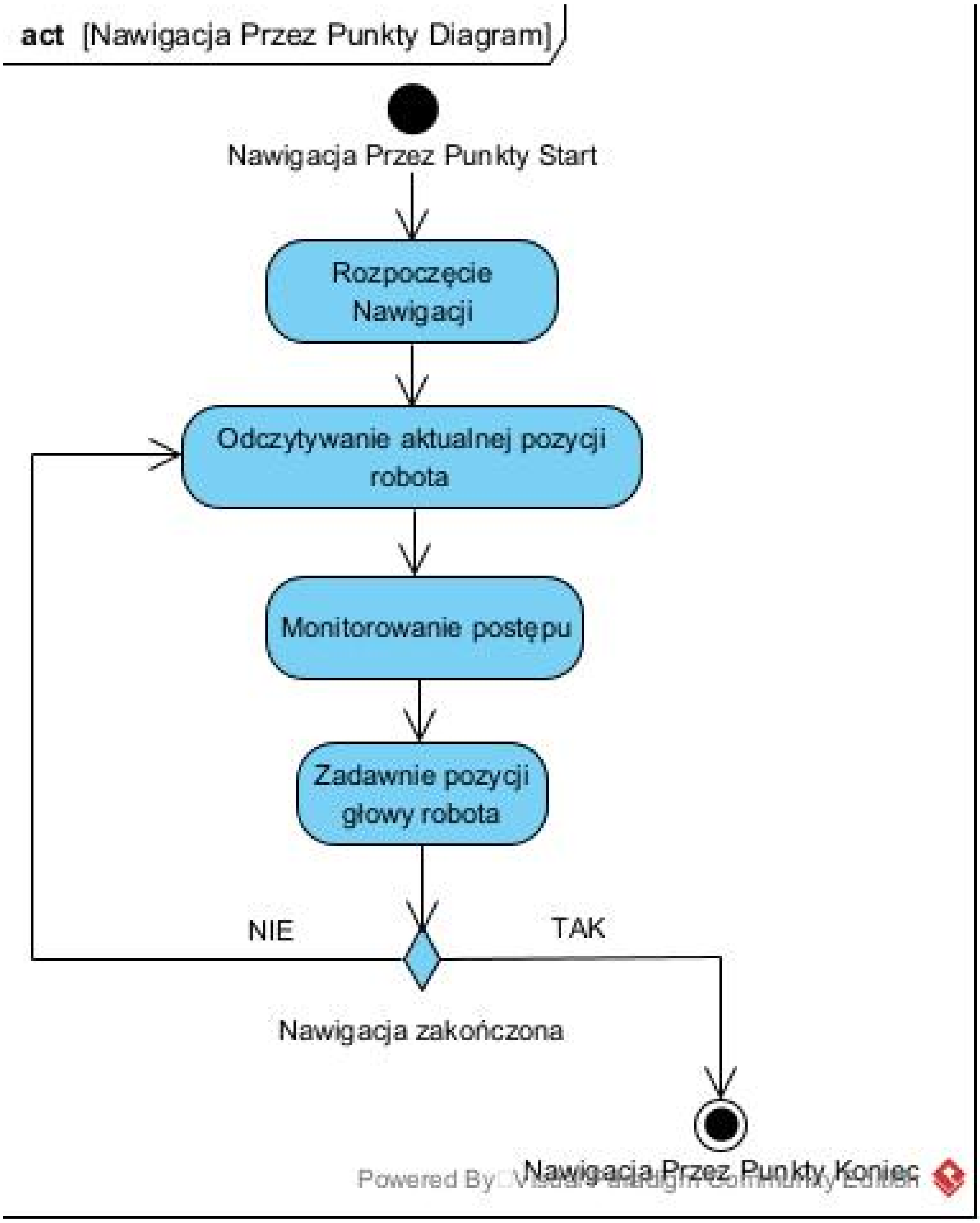
Na diagramie 7.1 przedstawiono ogólny schemat działania wezła WaipointFollower z wydzieleniem akcji dziejących się podczas uruchamiania nawigacji 7.2 oraz podczas nawigowania robota przez punkty 7.3



Rys. 7.1. WaipointFollower Action Diagram



Rys. 7.2. Nawigacja Inicjalizacja Diagram



Rys. 7.3. Nawigacja przez punkty Diagram