Tadeusz Chmielik Michał Pióro

Laboratorium 4 STERO

Grudzień 2024

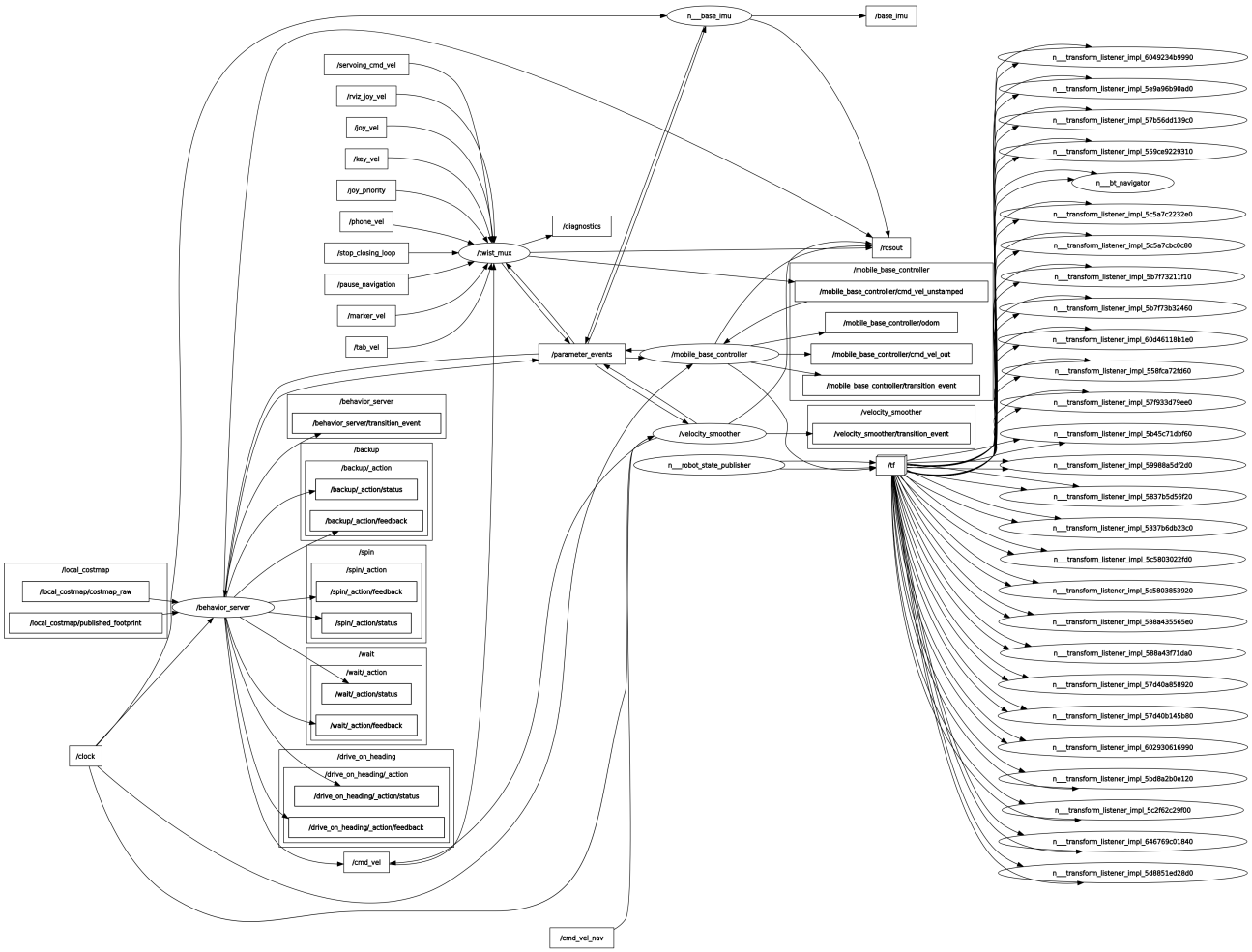
1. Analiza systemu robota TIAGo

System pozwalający na symulację systemu robota uruchomiono z sukcesem co pozwoliło na jego przeanalizowanie oraz stworzenie pierwszego węzła.

1.1. struktura sterowania

1.1.1. prędkość bazy

W celu zadania prędkości bazy należy skorzystać z jednego z dwóch topiców /cmd_vel, /cmd_nav (pierwszy najczęściej do ręczenego sterowania przez użytkownika, drugi do autopmatycznej nawigacji), które udostępniane są przez node moblie_base_controller. W celu zidentyfikowania tego skorzystano z rqt_graph-a.



Rys. 1.1. rqt_graph dla węzła moblie_base_controller

```
    student@gepard:~/stero$ ros2 topic info /cmd_vel
Type: geometry_msgs/msg/Twist
Publisher count: 5
Subscription count: 1
student@gepard:~/stero$ ros2 topic info /cmd_vel_nav
Type: geometry_msgs/msg/Twist
Publisher count: 1
Subscription count: 1
```

1.1.2. Odometria

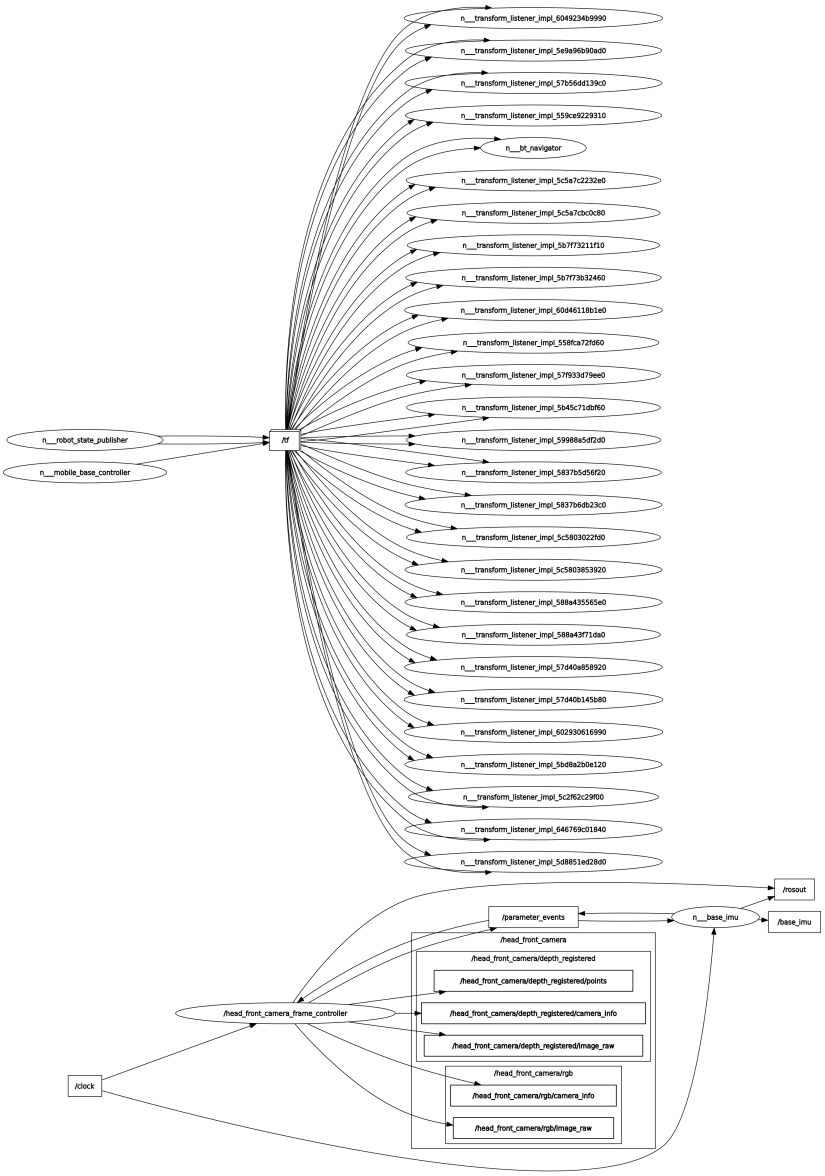
W celu odczytu odometrii robota można wykorzystać topic /mobile_base_controler/odom który można znaleźć na wyżej pokazanym grafie.

```
ros2 topic info /mobile_base_controller/odom
Type: nav_msgs/msg/Odometry
Publisher count: 1
Subscription count: 3
```

1.1.3. Czujnik LiDAR

Za obsługę czujnika LiDAR odpowiada węzeł /base_laser. Ponadto do publikowania danych wykorzystywane są tematy /scan oraz /scan_raw. Dzięki serwisom: /base_laser, /describe_parameters, /base_laser, /get$_p$*arameter_types*, /*base_laser*, /*get*/$_p$*arameters*, /*base_laser*, /*get_type_descript*

1.1.4. kamera RGB-D



Rys. 1.2. rqt_graph dla węzła head_front_camera_controller

Jak można zauważyć do obsługi kamery oraz do wysyłania danych przez nią zebranych wykorzystano wiele węzłów udostępniających wiele danych na wiele sposobów.

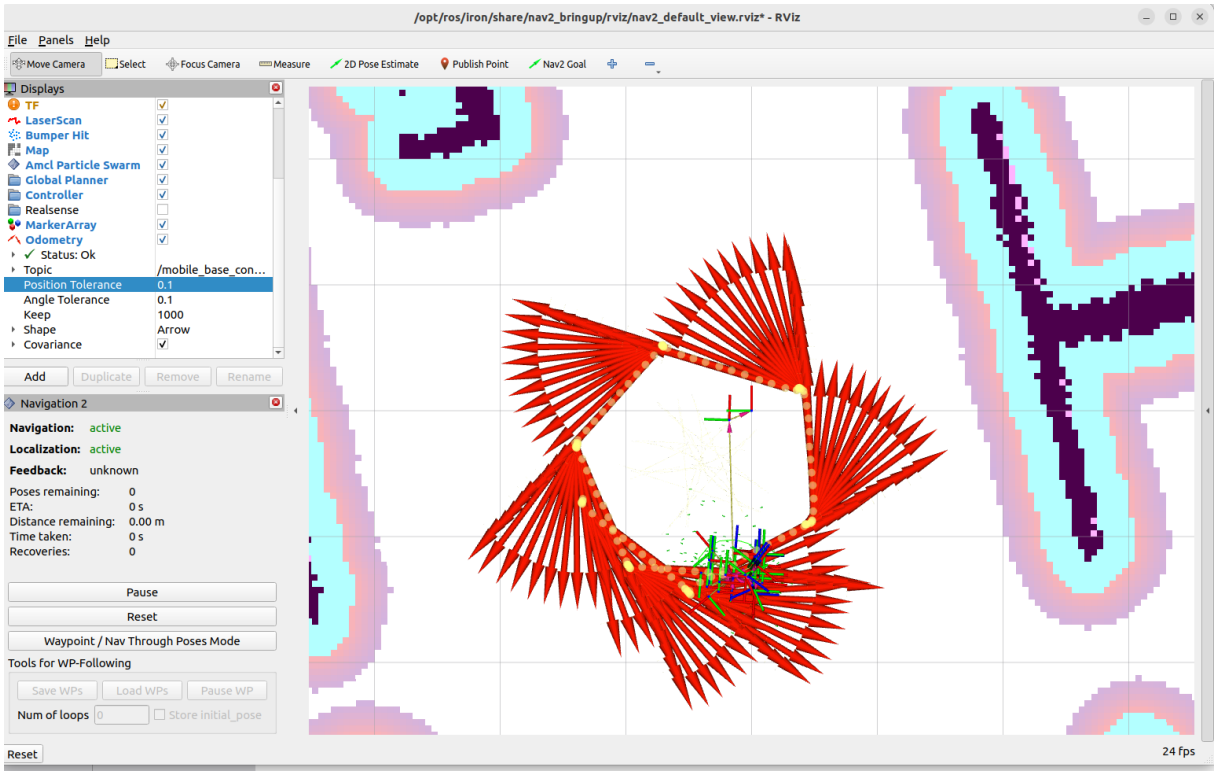
```
    ros2 node info /head_front_camera_frame_controller
/head_front_camera_frame_controller
  Subscribers:
    /clock: rosgraph_msgs/msg/Clock
    /parameter_events: rcl_interfaces/msg/ParameterEvent
  Publishers:
    /head_front_camera/depth_registered/camera_info: sensor_msgs/msg/CameraInfo
    /head_front_camera/depth_registered/image_raw: sensor_msgs/msg/Image
    /head_front_camera/depth_registered/points: sensor_msgs/msg/PointCloud2
    /head_front_camera/rgb/camera_info: sensor_msgs/msg/CameraInfo
```

```
    /head_front_camera/rgb/image_raw: sensor_msgs/msg/Image
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /rosout: rcl_interfaces/msg/Log
  Service Servers:
    /head_front_camera_frame_controller/describe_parameters: rcl_interfaces/srv/
    /head_front_camera_frame_controller/get_parameter_types: rcl_interfaces/srv/
    /head_front_camera_frame_controller/get_parameters: rcl_interfaces/srv/GetPa
    /head_front_camera_frame_controller/get_type_description: type_description_i
    /head_front_camera_frame_controller/list_parameters: rcl_interfaces/srv/Listl
    /head_front_camera_frame_controller/set_camera_info: sensor_msgs/srv/SetCame
    /head_front_camera_frame_controller/set_parameters: rcl_interfaces/srv/SetPa
    /head_front_camera_frame_controller/set_parameters_atomically: rcl_interface
  Service Clients:

  Action Servers:

  Action Clients:
```

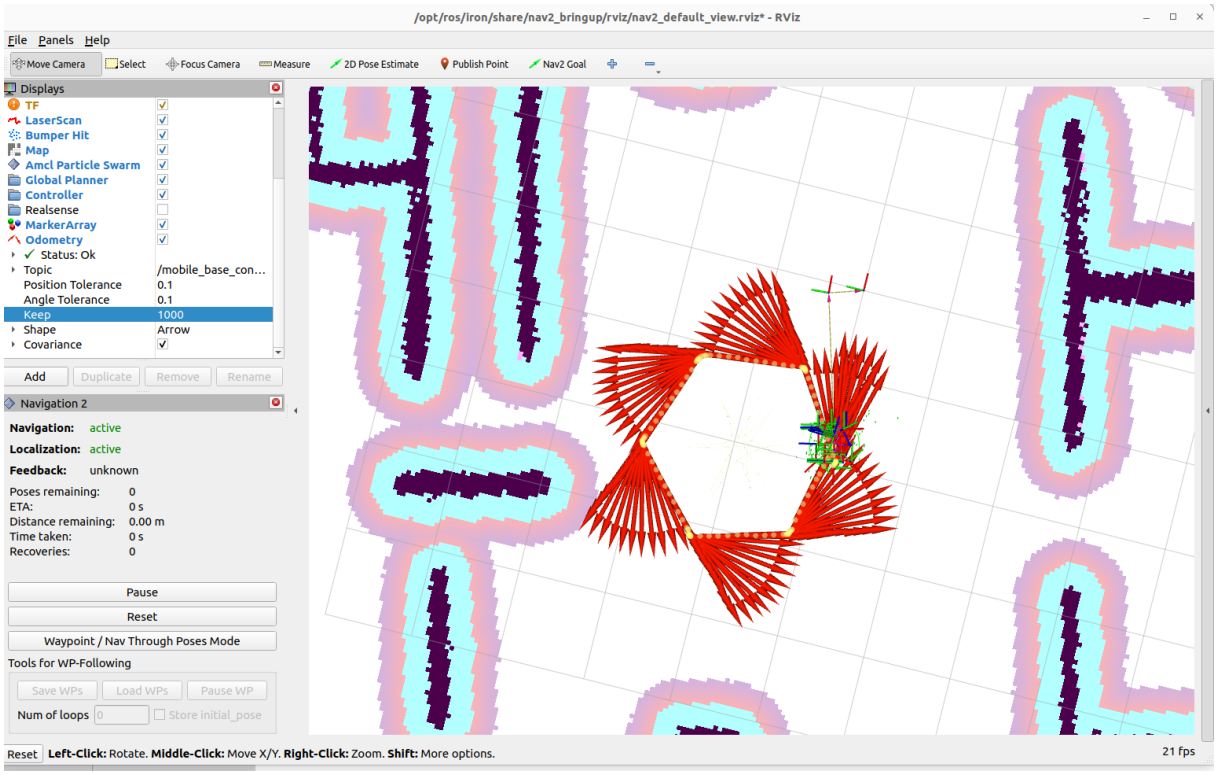
2. Węzeł test_nav

2.1. Opis wykonanego zadania

W ramach zadania napisano w języku C++ węzeł *test_nav*, którego zadaniem było sterowanie robotem Tiago w taki sposób, aby poruszał się w środpowisku symulacyjnym po trasie w kształcie sześciokąta. Aktualna prędkość liniowa i kątowa jest publikowana na temacie */cmd_vel*. Informacje o aktualnym położeniu i orientacji robota jest pobierana z tematu */odom*, na tej podstawie po przejechaniu zadanej odległości lub obrocie o kąt 120 stopni następowała zmiana publikowanych prędkości. Początkowo przy większej prędkości robot miał problem z dokładnym przejechaniem wyznaczonej trasy, może to być spowodowane opóźnieniami w komunikacji wewnątrz systemu, a także z bezwładności symulowanego robota.



Rys. 2.1. Wizaulizacja trasy przejazdu i kolejnych wektorów położenia robota przy zbyt dużej prędkości



Rys. 2.2. Wizaulizacja trasy przejazdu i kolejnych wektorów położenia robota dla optymalnych parametrów

2.2. Kod

2.2.1. Węzeł C++

```
    #include <rclcpp/rclcpp.hpp>
#include <nav_msgs/msg/odometry.hpp>
#include <geometry_msgs/msg/twist.hpp>
#include <geometry_msgs/msg/pose.hpp>
#include <cmath>
#include <chrono>

using namespace std::chrono_literals;

// Definicja krawędzi sześciokąta i prędkości robota
const double SIDE_LENGTH = 1.0;  // Długość boku sześciokąta (w metrach)
const double LINEAR_VELOCITY = 0.2;  // Prędkość liniowa (w metrach na sekundę)
const double ANGULAR_VELOCITY = 3.14 / 18.0;  // Prędkość kątowa (60 stopni na s
const double ANGULAR_ROTATION = 3.14 / 3.0;

class OdomListener : public rclcpp::Node
{
public:
    OdomListener() : Node("odom_listener")
    {
        subscription_ = this->create_subscription<nav_msgs::msg::Odometry>(
            "/mobile_base_controller/odom", 10, std::bind(&OdomListener::odomCal
```

```
        cmd_vel_pub_ = this->create_publisher<geometry_msgs::msg::Twist>("/cmd_v

        current_side_ = 0;
        move_forward_ = true;

        prev_position_ = geometry_msgs::msg::Pose();
        prev_orientation_ = 0.0; // Inicjalizacja kąta obrotu
    }

private:
    rclcpp::Subscription<nav_msgs::msg::Odometry>::SharedPtr subscription_;
    rclcpp::Publisher<geometry_msgs::msg::Twist>::SharedPtr cmd_vel_pub_;

    int current_side_;  // Numer aktualnego boku sześciokąta
    bool move_forward_;  // Flaga wskazująca, czy robot jedzie do przodu, czy sk
    geometry_msgs::msg::Pose prev_position_;  // Poprzednia pozycja robota
    double prev_orientation_;  // Poprzednia orientacja robota (kąt)

    void odomCallback(const nav_msgs::msg::Odometry::SharedPtr msg)
    {
        double current_x = msg->pose.pose.position.x;
        double current_y = msg->pose.pose.position.y;

        double dx = current_x - prev_position_.position.x;
        double dy = current_y - prev_position_.position.y;
        double distance_travelled = std::sqrt(dx * dx + dy * dy);

        double current_orientation = getYaw(msg->pose.pose.orientation);

        double delta_orientation = current_orientation - prev_orientation_;

        if (delta_orientation > 3.14)
        {
            delta_orientation -= 2 * 3.14;
        }
        else if (delta_orientation < -3.14)
        {
            delta_orientation += 2 * 3.14;
        }

        if (move_forward_)
        {
            if (distance_travelled >= SIDE_LENGTH)
            {
                move_forward_ = false;
                prev_position_ = msg->pose.pose;
                prev_orientation_ = current_orientation;
            }
        }
        else
```

```
        {
            if (std::abs(delta_orientation) >= ANGULAR_ROTATION)
            {
                // Po obróceniu o zadany kąt, robot kontynuuje jazdę do przodu
                move_forward_ = true;
                prev_position_ = msg->pose.pose;
                prev_orientation_ = current_orientation;
                current_side_++;
            }
        }

        moveInHexagon();

        RCLCPP_INFO(this->get_logger(), "Current Angle: %f, Distance travelled: %
    }

    double getYaw(const geometry_msgs::msg::Quaternion& quat)
    {
        // Wzór na wyliczenie kąta yaw z kwaternionu
        double siny_cosp = 2.0 * (quat.w * quat.z + quat.x * quat.y);
        double cosy_cosp = 1.0 - 2.0 * (quat.y * quat.y + quat.z * quat.z);
        return std::atan2(siny_cosp, cosy_cosp);
    }

    void moveInHexagon()
    {
        auto msg = geometry_msgs::msg::Twist();

        if (move_forward_)
        {
            // Poruszamy się do przodu
            msg.linear.x = LINEAR_VELOCITY;
            msg.angular.z = 0.0;
        }
        else
        {
            // Skręcamy o 60 stopni
            msg.linear.x = 0.0;
            msg.angular.z = ANGULAR_VELOCITY;
        }

        // Publikacja prędkości
        cmd_vel_pub_->publish(msg);
    }
};

int main(int argc, char * argv[])
{
    rclcpp::init(argc, argv);
    rclcpp::spin(std::make_shared<OdomListener>());
    rclcpp::shutdown();
```

```
    return 0;
}
```

2.2.2. CMake

```
cmake_minimum_required(VERSION 3.8)
project(lab4)

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(nav_msgs REQUIRED)

add_executable(test_nav src/test_nav.cpp)

target_include_directories(test_nav PUBLIC
  $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
  $<INSTALL_INTERFACE:include>)

target_compile_features(test_nav PUBLIC c_std_99 cxx_std_17)

ament_target_dependencies(test_nav rclcpp nav_msgs)

install(TARGETS test_nav
  DESTINATION lib/${PROJECT_NAME})

if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  set(ament_cmake_copyright_FOUND TRUE)
  set(ament_cmake_cpplint_FOUND TRUE)
  ament_lint_auto_find_test_dependencies()
endif()

ament_package()
```