Übungsblatt 3

Aufgabe 1

In dieser Übung schreiben wir einen simplen ROS Node der Nachrichten an eine topic sendet. Als basis nutzen wir den Quellcode im Verzeichnis ./code neben dieser Übung. Das Grundgerust hier baut aur der vergangenen Übung 2 auf.

a) Zunächst erzeugen wir einen neues ROS package, dafür gehen wir ins Verzeichnis ./code/src und führen dann folgenden Befehl aus:

```
ros2 pkg create --build-type ament_cmake my_robot_controller --dependencies rclcpp geometry_msgs
```

Dies erzeugt ein ROS Package in der die package.xml mit ein paar Abhängigkeiten vorausgefüllt ist, namentlich rclcpp, geometry_msgs und ament_cmake

b) Danach erzeugen wir die Datei src/my_robot_controller/src/velocity_publisher.cpp und füllen Sie mit folgendem Inhalt:

```cpp
#include <rclcpp/rclcpp.hpp>
#include <geometry_msgs/msg/twist.hpp>

class VelocityPublisher : public rclcpp::Node
{
public:
    VelocityPublisher() : Node("velocity_publisher")
    {
        publisher_ = this->create_publisher<geometry_msgs::msg::Twist>("/cmd_vel", 10);
        timer_ = this->create_wall_timer(
            std::chrono::milliseconds(500),
            std::bind(&VelocityPublisher::publish_velocity, this));
    }

private:
    void publish_velocity()
    {
        auto msg = geometry_msgs::msg::Twist();
        msg.linear.x = 0.5;
        msg.angular.z = 0.2;
        RCLCPP_INFO(this->get_logger(), "Publishing: linear.x = %f, angular.z = %f", msg.linear.x, msg.
            angular.z);
        publisher_->publish(msg);
    }

    rclcpp::Publisher<geometry_msgs::msg::Twist>::SharedPtr publisher_;
    rclcpp::TimerBase::SharedPtr timer_;
};

int main(int argc, char *argv[])
{
    rclcpp::init(argc, argv);
    rclcpp::spin(std::make_shared<VelocityPublisher>());
    rclcpp::shutdown();
    return 0;
}
```

Dieser code startet einen Ros node mit dem Namen velocity_publisher und er sendet alle 500ms eine Nachricht an die /cmd_vel topic. Dies steuert die simulierten Motoren von unserem Roboter an.

c) Danach fügen wir die neue Datei in die CMakeLists.txt von unserem Knoten hinzu. Dafür passen wir die Datei an (src/my_robot_controller/CMakeLists.txt)

```
cmake_minimum_required(VERSION 3.5)
project(my_robot_controller)

find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(geometry_msgs REQUIRED)

add_executable(velocity_publisher src/velocity_publisher.cpp)
ament_target_dependencies(velocity_publisher rclcpp geometry_msgs)

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

ament_package()
```

d) Jetzt erweitern wir noch unsere launch Datei, damit auch unser neu geschriebener Knoten gestartet wird! Ergänzen in der Datei src/my_robot_bringup/launch/-my_robot.launch.xml folgendes:

```
<launch>
  <node
    name="velocity_publisher"
    pkg="my_robot_controller"
    exec="velocity_publisher"
    output="screen" />
</launch>
```

e) Dann musst du einmal neu kompilieren und die erzeugte setup.bash sourcen. Du solltest inzwischen wissen wie das geht!

f) Jetzt können wir unsere launch datei starten:

```
ros2 launch my_robot_bringup my_robot.launch.xml
```