Übungsblatt 1

Aufgabe 1

In dieser Übung möchten wir uns mit den Grundlagen von ROS befassen und unseren ersten kleinen Rosknoten entwickeln. Ihr solltet aktuellt eine bereits funktionsfähige ROS-Umgebung installiert haben. Entweder nativ, in einer VM oder über Docker.

a) Zunächst stellen wir sicher, dass wir einen richtigen ros kontext geladen habe. Ruft das programm `"ros2"` auf. Falls ihr dort eine Fehlermeldung bekommt, dann habt ihr die ros2 Umgebung noch nicht geladen, das geht via `"source␣/opt/ros/jazzy/setup.bash"`.

b) als nächstes möchten wir einen ros workspace erzeugen. Ein ros workspace (colcon workspake) ist letztlich nur eine Ordnerstruktur die einem bestimmten muster folgt. Dafür erzeugen wir zunächst ein neues Verzeichnis. Wir nennen es ”blatt1_ws”, dafur nutzt `"mkdir␣blatt1_ws"` in einem für euch geeigneten Pfad. Danach navigieren wir in den Ordner, `"cd␣blatt1\_ws"` und erzeugen einen Unterordner src, `"mkdir␣src"`.

c) Als nächstes möchten wir ein eigenes packages erzeugen. Dafür wechseln wir in den ordner src, `"cd␣src"` und führen dann folgenden Befehl aus: `"ros2␣pkg␣create␣my_cpp_pkg␣--build-type␣ament_cmake␣--dependencies␣rclcpp"`. Dies erzeugt ein ros package mit dem namen my_cpp_pkg als CMake Projekt mit einer Abhängigkeit zu rclcpp, die wir benötigen, wenn wir aus C++ heraus auf ros bibliotheken zugreifen möchten. Nun wurde ein Ordner namens my_cpp_pkg erstellt mit folgendem Inhalt: `"CMakeLists.txt␣␣include␣␣package.xml␣␣src"`

d) Nun gehen wir zurück zum workspace root und können `"colcon␣build"` ausführen. In den Ausgaben können wir sehen, dass unser package bereits gefunden und automatisch gebaut wird. Allerdings gibt es aktuell noch nicht sonderlich viel Inhalt. Zeit das zu ändern!

e) Wir legen eine Datei an und editieren diese: `"src/my_cpp_pkg/src/my_first_node.cpp"`. Wir geben ihr folgenden Inhalt (Tipp, neben diesem PDF liegt auch die Latex-Quelldatei, aus dieser Datei lässt es sich einfacher Inhalte kopieren!):

```
#include "rclcpp/rclcpp.hpp"
class MyCustomNode : public rclcpp::Node
{
  public:
    MyCustomNode() : Node("my_node_name"), counter_(0)
    {
      timer_ = this->create_wall_timer(std::chrono::seconds(1),
      std::bind(&MyCustomNode::print_hello, this));
    }

    void print_hello()
    {
      RCLCPP_INFO(this->get_logger(), "Hello %d", counter_);
```

```
14        counter_++;
15      }
16
17    private:
18      int counter_;
19      rclcpp::TimerBase::SharedPtr timer_;
20 };
21
22 int main(int argc, char **argv)
23 {
24   rclcpp::init(argc, argv);
25   auto node = std::make_shared<MyCustomNode>();
26   rclcpp::spin(node);
27   rclcpp::shutdown();
28   return 0;
29 }
```

f) Zudem müssen wir die CMakeLists.txt in unserem package anpassen. Der Inhalt der Datei sollte wie folgt aussehen.

```
1 cmake_minimum_required(VERSION 3.8)
2 project(my_cpp_pkg)
3
4 if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
5   add_compile_options(-Wall -Wextra -Wpedantic)
6 endif()
7
8 # find dependencies
9 find_package(ament_cmake REQUIRED)
10 find_package(rclcpp REQUIRED)
11
12 add_executable(test_node src/my_first_node.cpp)
13 ament_target_dependencies(test_node rclcpp)
14
15 install(TARGETS
16   test_node
17   DESTINATION lib/${PROJECT_NAME}/
18 )
19
20 if(BUILD_TESTING)
21   find_package(ament_lint_auto REQUIRED)
22   # the following line skips the linter which checks for copyrights
23   # comment the line when a copyright and license is added to all source files
24   set(ament_cmake_copyright_FOUND TRUE)
25   # the following line skips cpplint (only works in a git repo)
26   # comment the line when this package is in a git repo and when
27   # a copyright and license is added to all source files
28   set(ament_cmake_cpplint_FOUND TRUE)
29   ament_lint_auto_find_test_dependencies()
30 endif()
31
32 ament_package()
```

Hinzugefügt haben wir den aufruf zu add_executable und ament_target_dependencies und die install Anweisung.

g) Danach bauen wir wieder aus unserem root verzeichnis mit colcon build. Danach aktivieren wir unsere eigene ros Umgebung die auch unser package enthält, "`source␣./install/setup.bash`".

h) Jetzt können wir unseren code ausführen. "`ros2␣run␣my_cpp_pkg␣test_node`".

i) Wir sollten nun die Ausgaben von unserem Code sehen. Zusätzlich können wir unseren laufenen node mitels "`ros2␣node␣info␣/␣ros2␣node␣list`" insipzieren.