Ergänzung zur Vorlesung

Leistungskurs C++

Martin Knopp

8. November 2016

1 Voreinstellungen

Die hier genannten Punkte sind in der virtuellen Maschine bereits erledigt. Falls Ihr ROS auf eurem eigenen Computer einrichten wollt, oder euch die Hintergründe interessieren, findet ihr auf den verlinkten Seiten Anleitungen, was für den jeweiligen Punkt zu tun ist.

- catkin (CMake[1] basiertes Buildsystem für ROS) Workspace erzeugen und einrichten: `http://wiki.ros.org/catkin/Tutorials/create_a_workspace`
- QtCreator anpassen. QtCreator unterstützt von Haus aus CMake, deswegen sind nur wenige Dinge nötig. Für die von uns eingesetzte ROS-Version (Indigo) ist Punkt 2, catkin_make, relevant: `http://wiki.ros.org/IDEs#QtCreator`
- Ihr solltet jetzt die oberste CMakeLists.txt als Projekt im QtCreator öffnen können, in der virtuellen Maschine ist das bereits als „Recent Project“ geschehen und ihr könnt den Code aus dem nächsten Kapitel ausprobieren.

[1] `http://cmake.org`

2 „Hello World“ in ROS

```cpp
#include <ros/ros.h>

int main(int argc, char *argv[]) {
    ros::init(argc, argv, "hello");

    ros::NodeHandle nh;
    ros::Rate loop_rate(10);

    int count = 0;
    while (ros::ok()) {
        ROS_INFO_STREAM("hello world!" << count);

        ros::spinOnce();
        loop_rate.sleep();
        ++count;
    }

    return 0;
}
```

Listing 1: catkin_ws/src/tutorial_1/hello.cpp

Die interessanten Teile im Einzelnen:

- `#include <ros/ros.h>` — Bindet roscpp[2] ein, zwingend notwendig.
- `ros::init(argc, argv, "hello");` — Initialisiert den ROS-Knoten.

  Idealerweise kommt die Zeile gleich als erster Aufruf in `main()`, da ROS dynamisch die verbundenen Knoten ändern kann (→ erste Vorlesung: einfaches Testen). Zu diesem Zweck schreibt ROS ggf. die übergebenen Argumente um, der nachfolgende Code sollte daher die hier angepassten Variablen nutzen. Als dritter Paramenter wird der Knoten benannt, Namespaces (/…/) sind hier nicht erlaubt. Der Name muss innerhalb des ROS-Pakets eindeutig sein, doppelte Namen sind nicht erlaubt.
- `ros::NodeHandle nh;` — Die Hauptschnittstelle in ROS, Interface zu Topics, Services, Parametern, usw.

  Der Konstruktor des ersten NodeHandles initialisiert den Knoten vollständig, der Destruktor des letzten NodeHandles beendet den Knoten.
- `ros::Rate loop_rate(10);` — Hilfsklasse um Schleifen mit einer definierten Frequenz ablaufen zu lassen. Die Rate wird in Hz angegeben.

  Dazu gehört die Hilfsmethode `ros::Rate::sleep()`, sie wartet für die restliche Zeit des Zyklus. Ausgangspunkt des Zeitgebers ist der letzte Aufruf von `sleep()`, `reset()` oder des Konstruktors.
- `ros::ok()` — Abfrage, ob der Knoten weiterarbeiten soll.

  Das ist nicht der Fall, wenn

[2] http://docs.ros.org/api/roscpp/html/

  - `SIGINT` an den Prozess geschickt wird ($\rightarrow$ Strg+C gedrückt wurde).
  - Wir von einem Knoten mit dem gleichen Namen aus dem Netz geworfen wurden.
  - `ros::shutdown()` innerhalb der Anwendung aufgerufen wurde.
  - Alle `ros::NodeHandles` zerstört wurden
- `ROS_INFO_STREAM(…);` eine der fünf klassischen Logausgaben (DEBUG, INFO, WARN, ERROR und FATAL), hier in der C++-Stream-Variante. Es gibt sie auch ohne `_STREAM`, dann ist ihre Syntax wie `printf()`.
- `ros::SpinOnce();` Erlaubt ROS die Callbackfunktionen für eingehende Nachrichten aufzurufen.

2.1 Ausprobieren

Wie ihr in der `tutorial_1/CMakeLists.txt` und der dazugehörigen `package.xml` sehen könnt, wird aus der `hello.cpp` das Programm `hello` im Paket `tutorial_1`.

Startet also, wie in der Vorlesung gezeigt, in einem Terminal (Strg+Alt+t) `roscore` und in einem zweiten euren Knoten (`rosrun tutorial_1 hello`). Das funktioniert auf Anhieb, weil die `source […]`-Befehle passend in eurer `~/.bashrc` stehen. Falls ihr das auf eurem eigenen Rechner macht, müsstet ihr die dort auch anfügen, oder vorher im Terminal ausführen.

Wenn ihr fleißig mit TAB ergänzt, ist euch vielleich schon aufgefallen, dass es auch einen Knoten `tutorial_1_node` gibt, der genau das gleiche macht, den könnt ihr als Vergleich verwenden, ansonsten zeigt er schön, was passiert, wenn man in der CMakeLists.txt den Namen des erzeugten Knoten ändert: Der Alte wird nicht automatisch gelöscht.