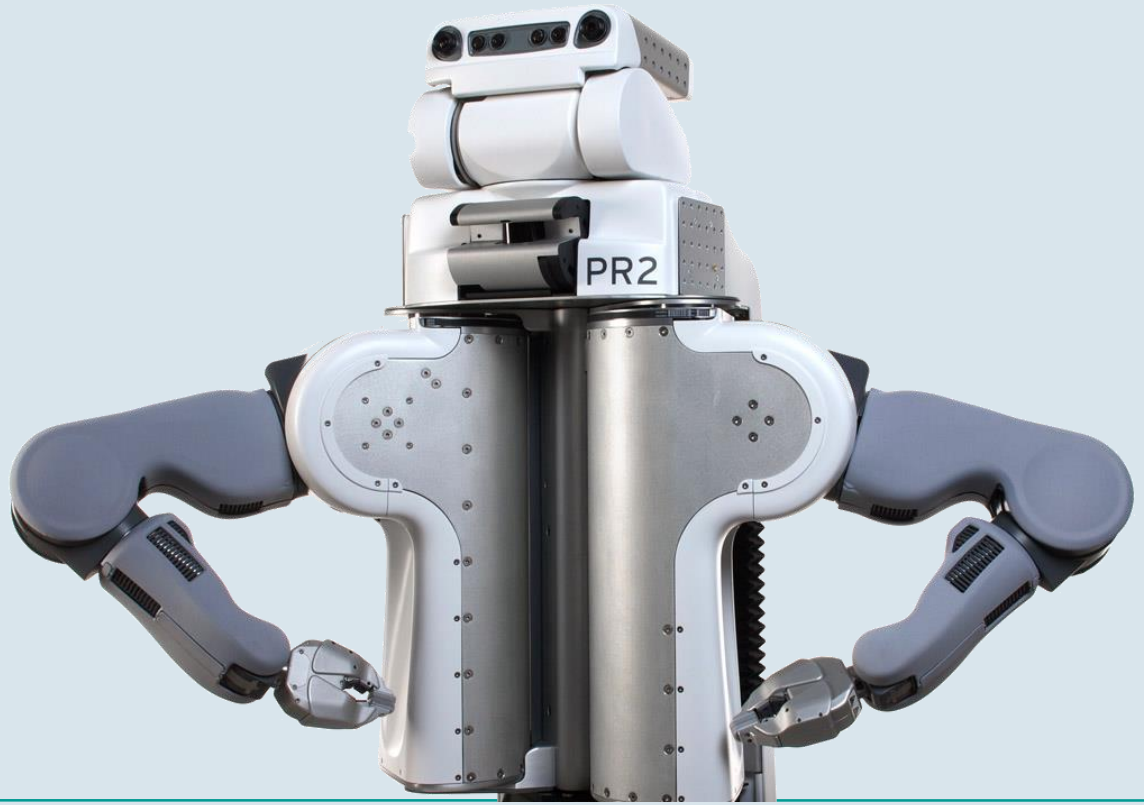


Mobile Roboter WS 2022/23

ROS Einführung

Robot Operating System (ROS)

Was ist ROS?

- ROS ist ein Middelware um Roboter zu Programmieren
- Meist genutzte Roboter Programmierumgebung
- Wird sowohl in der Industrie als auch im akademischen Umfeld weltweit eingesetzt

Übungen zur Programmierung Autonomer Roboter

- Kennen lernen von ROS
  - Philosophie, Architektur, Tools, …
- Implementierung eigener Algorithmen in ROS

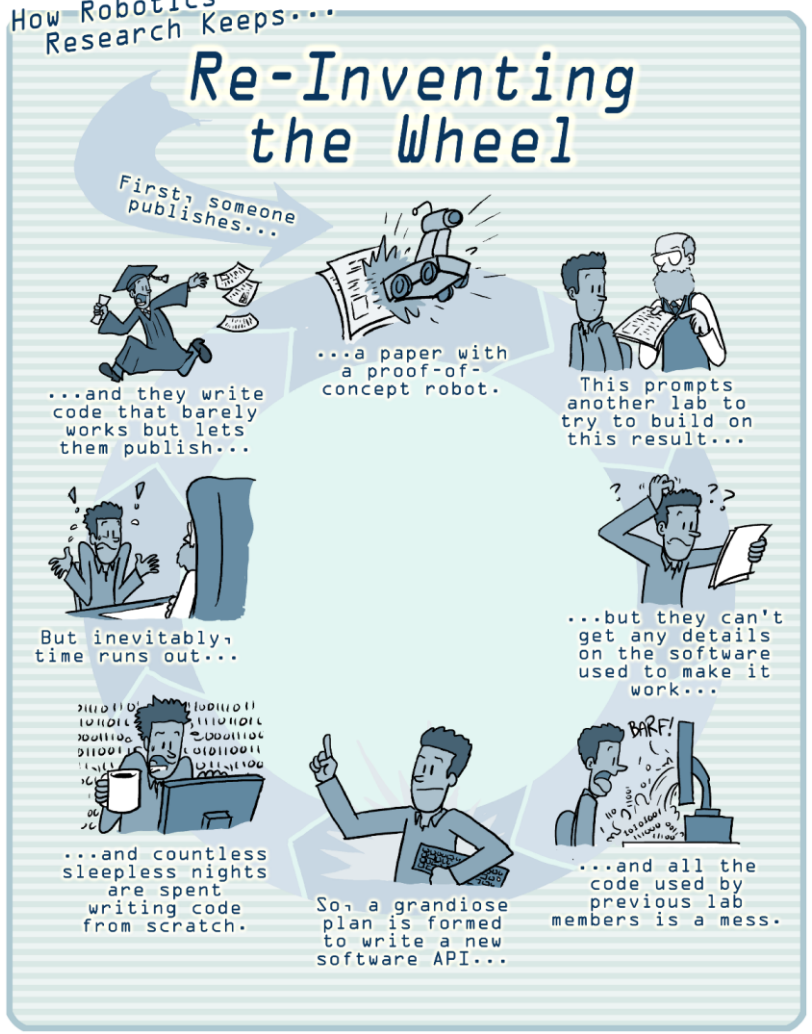
Voraussetzung für die Übungen mit ROS

- Grundlegende Programmierkenntnisse in C++ / Python
- Grundlegende Linux Kenntnisse (Konsole/Terminal)
- Grundlegende git Kenntnisse

Geschichte von ROS

Warum überhaupt ROS?

- In frühen Robotik-Projekten wurde oft die gesamte Software-Architektur neu entwickelt (Re-Inventing the Wheel)
- Keenan Wyrobek and Eric Berger Doktoranden an der Stanford University wollten dies ändern und ein Framework für Roboter entwickeln (Linux of Robotics)
- 2006 bekamen sie 50.000 $ um einen Roboter und das Framework dazu zu entwickeln. (Video PR1)
- Vielen Ideen und Konzepte aus diesem Projekt sind heute noch in ROS zu finden: ROS-comm libraries, RViz, rqt_tools oder der vereitele Ansatz
- 2007 Keenan Wyrobek and Eric Berger wechseln zu Willow Garage welches die Entwicklung von ROS übernimmt
- 2009 Veröffentlichung des ROS Papers auf der ICRA (ROS: An Open-Source Robot Operating System)

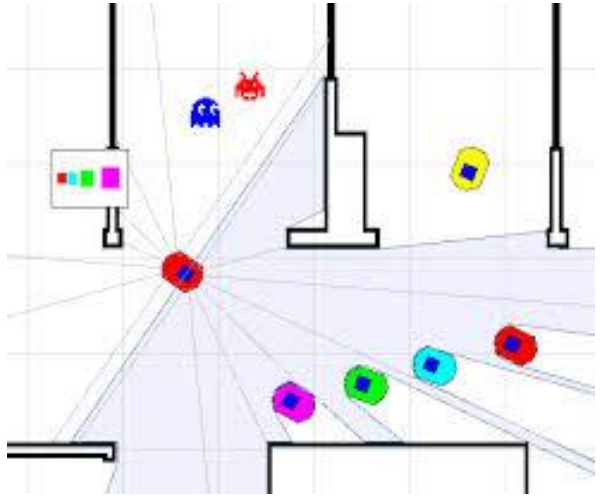


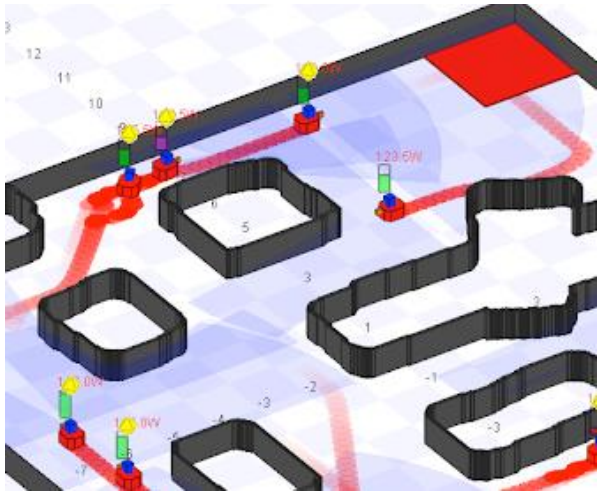
Quelle:http://www.willowgarage.com/pages/pr2/pr2-community

Geschichte von ROS

Weitere Robotik Frameworks zu dieser Zeit

- Player/Stage
  - 1999 - 2010
  - Open-source
  - ähnliche Konzept wie ROS
  - 2008 Hauptentwickler Brian Gerkey wechselt zu Willow





- URBI
  - 2003 – 2013 (Revision 3.0)
  - sehr gutes Framework mit vergleichbarer Funktionalität zu ROS
  - früher kostenpflichtig jetzt open-source
  - kleine Community

- Open-R
  - 1999 - 2006
  - Programmierumgebung von Sony für Aibo

- YARP, ARIA, …

Aibo bis 2006 (Open-R)

Aibo ab 2017 (ROS)

Geschichte von ROS

Wie hat sich ROS zum De facto Standard entwickelt?

- 2007 – 2013 *Willow Garage* übernimmt die Entwicklung von ROS
  - 2007 erst Veröffentlichung von ROS auf SourceForge
  - Entwicklung des PR2
- 2013 – 2016 weiterentwickelt durch die *Open Source Robotic Foundation* (OSRF)
  - Die Entwicklung von ROS wird durch weitere Fimen untesrstützt (BOSCH, Intel, …)
- 2016 Umbenennung der OSRF zu Open Robotics
- 2020 letzte ROS 1 Distribution (Noetic) wird veröffentlicht
- ab 2020 Entwicklung wird auf ROS 2 konzentriert

PR2 von Willow Garage

Robot Operating System (ROS)

ROS1 vs. ROS2

- ROS1 etabliert und erprobt
- ROS1 immer noch meist verbreitet
- Problemloser Umstieg von ROS1 auf ROS2

vs.

- ROS1 etabliert und erprobt
- ROS1 immer noch meist verbreitet

- ROS2 Released 12/17
- ROS2 besseres Design
- Nicht kompatible mit ROS1 (Bridge wird benötigt)
- Weniger Funktionalität
- Wenig Erfahrungen
- Kein ROS Master mehr dafür DDS

Robot Operating System 2 (ROS2)

ROS 2 Konzept

- ROS2 ist eine Middleware, die auf einem anonymen Publish/Subscribe-Mechanismus basiert, der den Nachrichtenaustausch zwischen verschiedenen ROS-Prozessen (Nodes) ermöglicht.
- Das Herzstück eines jeden ROS2 Systems ist der ROS-Graph. Dieser ist ein Netzwerk von Knoten (Nodes) und Kommunikationsverbindungen

Graphen Konzept

- **Nodes**: Ein Knoten ist eine Einheit, die ROS zur Kommunikation mit anderen Knoten verwendet.
- **Messages**: ROS-Datentyp, die verwendet werden, wenn ein Topic abonniert oder veröffentlicht wird.
- **Topics**: Knoten können Nachrichten in einem Topic veröffentlichen und ein Topic abonnieren, um Nachrichten zu empfangen.
- **Discovery**: Der automatische Prozess, durch den Knoten feststellen, wie sie miteinander kommunizieren können.

Robot Operating System 2 (ROS2)

Nodes in ROS2

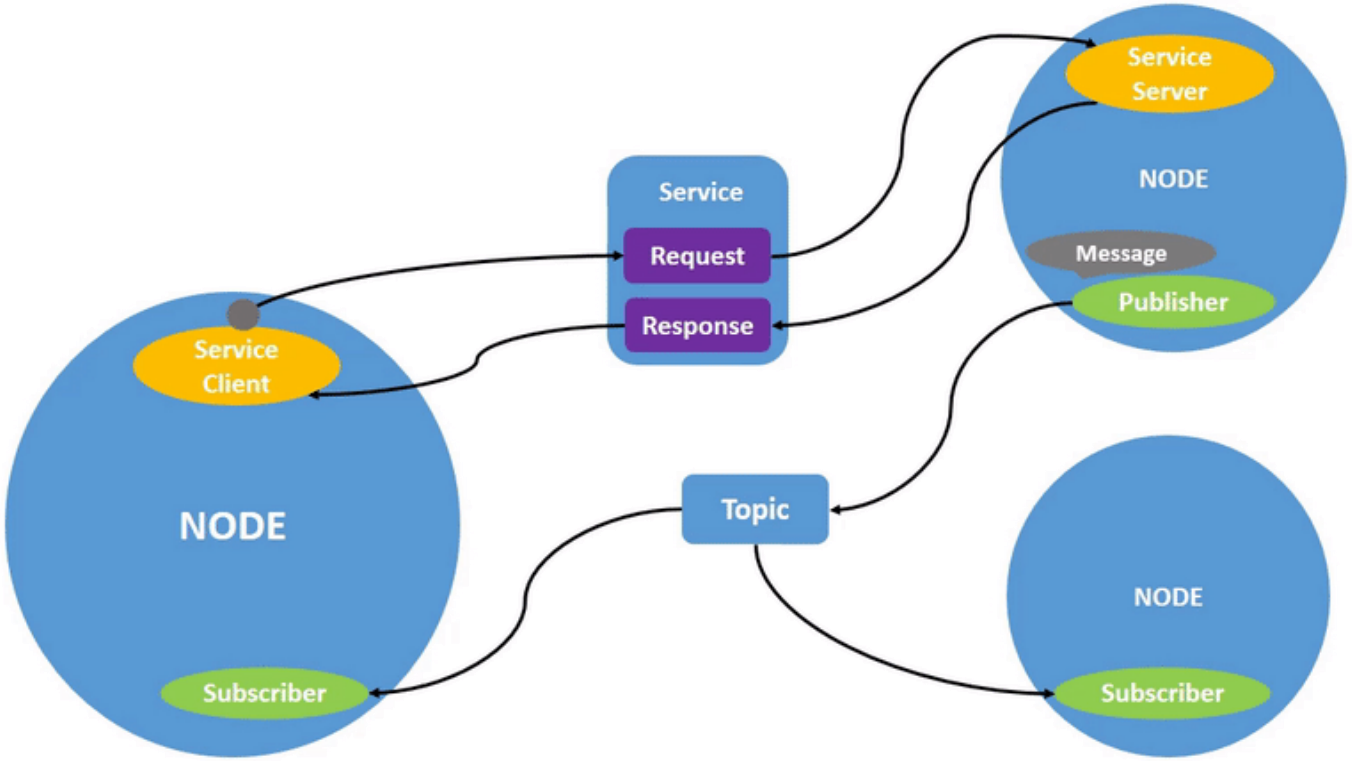
- Jeder Knoten in ROS sollte für einen einzigen, modularen Zweck zuständig sein (z. B. ein Knoten zur Steuerung von Radmotoren, ein Knoten zur Steuerung eines Arms, ein Knoten zur Steuerung eines Laserentfernungsmessers usw.).

- Jeder Knoten kann über Topics, Services, Actions oder Parameters Daten mit andere Knoten austauschen.

```
> ros2 node list
```

```
> ros2 node info <node_name>
```

```
> ros2 run <package_name> <node_name>
```



Robot Operating System 2 (ROS2)

Packages in ROS2

- In ROS2 ist alles in Packages organisiert.
- Meta-Packages Gruppieren Packages z.B. von einem Roboter (`my_robot`)

Besonderheit für die Übungen im Moro Docker-Container

- Sie legen ihre Packages im Unterordner moro/task… an

- Beispiel für Task3:

```
> ~/ros2_ws/src/moro/task3/my_package
```

```
> ~/ros2_ws/src/my_robot
```

```
|- my_robot
|- my_robot_base
|- my_robot_bringup
|- my_robot_description
|- my_robot_gazebo
|- my_robot_kinematics
|- my_robot_localization
|- my_robot_manipulation
|- my_robot_moveit_config
|- my_robot_msgs
|- my_robot_navigation
|- my_robot_teleop
|- my_robot_tests
|- my_robot_rviz_plugins
```

Quelle: https://automaticaddison.com/naming-and-organizing-packages-in-large-ros-2-projects/

ROS2 Package

ROS2 Python Package

- Für das Anlegen eines Package gibt es eine Hilfsfunktion
- Diese müssen in ihrem src Ordner der Workspace ausgeführt werden (z.B. ros2_ws/src)

```
> ros2 pkg create --build-type ament_python <package_name>
```

- Wenn sie schon wissen welche Nodes sie implementieren wollen können sie dies auch mit angeben

```
> ros2 pkg create --build-type ament_python --node-name <node_name> <package_name>
```

- Ein konkretes Beispiel ist dann:

```
> ros2 pkg create --build-type ament_python --node-name my_node my_python_pkg
```

- Wenn sie diese Befehle nutzen wird ein ROS2 Package mit entsprechender Ordnerstruktur angelegt
- Es werden auch die für ein Package benötigten Dateien mit angelegt
- Im Verlauf eines Projekts kommen dann oft noch weitere Dateien und Ordner dazu

ROS2 Package

ROS2 Python Minimal Package - Ordner

- `my_python_pkg`
  - Enthält eine `__init__.py` Datei und alle Nodes und sonstigen Code
  - In diesem Beispiel `my_node.py`
- `Resource`
  - Enthält eine leere Datei mit dem Package Name
  - Wird benötigt damit ROS das Package später findet
- `test`
  - Enthält erst mal nur drei Test
  - Ein Test zum prüfen ob in jeder Quelldatei ein Copyright enthalten ist
  - Zwei Linter Tests die den Python Code Style prüfen (https://peps.python.org/pep-0008/)

```
my_python_pkg/
├── my_python_pkg
│   ├── __init__.py
│   ├── my_node.py
│   └── ...
├── package.xml
├── resource
│   └── my_python_pkg
├── setup.cfg
├── setup.py
└── test
    ├── test_copyright.py
    ├── test_flake8.py
    └── test_pep257.py
```

ROS2 Package

ROS2 Python Minimal Package - Dateien

- `package.xml`
  - Package Informationen, wie z.B. Beschreibung, Autor, Version, Lizenz
  - Abhängigkeiten z.B. zu `rospy`

- `setup.cfg`
  - In dieser Datei wird festgelegt, wo die Skripte installiert werden sollen.
  - Hier muss erst mal nichts geändert werden

```
[develop]
script-dir=$base/lib/my_python_pkg
[install]
install-scripts=$base/lib/my_python_pkg
```

```
my_python_pkg/
├── my_python_pkg
│   ├── __init__.py
│   ├── my_node.py
│   └── ...
├── package.xml
├── resource
│   └── my_python_pkg
├── setup.cfg
├── setup.py
└── test
    ├── test_copyright.py
    ├── test_flake8.py
    └── test_pep257.py
```

ROS2 Package

ROS2 Python Minimal Package - Dateien

- `package.xml`
  - Package Informationen, wie z.B. Beschreibung, Autor, Version, Lizenz
  - Abhängigkeiten z.B. zu `rospy`
- `setup.cfg`
  - In dieser Datei wird festgelegt, wo die Skripte installiert werden sollen.
  - Hier muss erst mal nichts geändert werden
- `setup.py`
  - Enthält Informationen die für die Installation, das Kompilieren und das Ausführen der Nodes notwendig sind
  - Enthält auch wieder Informationen zum Package wie die package.xml

```
my_python_pkg/
├── my_python_pkg
│   ├── __init__.py
│   ├── my_node.py
│   └── ...
├── package.xml
├── resource
│   └── my_python_pkg
├── setup.cfg
├── setup.py
└── test
    ├── test_copyright.py
    ├── test_flake8.py
    └── test_pep257.py
```

ROS2 Package

Package.xml

Package Informationen

- `version`
- `description`
- `maintainer`
- `license`

Weiter Abhängigkeiten eintragen

- `python3-pytest`
- rclpy

```
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd"
schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>my_python_pkg</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="your@email.com">Name</maintainer>
  <license>TODO: License declaration</license>

  <buildtool_depend>ament_python</buildtool_depend>

  <test_depend>ament_copyright</test_depend>
  <test_depend>ament_flake8</test_depend>
  <test_depend>ament_pep257</test_depend>
  <test_depend>python3-pytest</test_depend>

  <exec_depend>rclpy</exec_depend>

  <export>
    <build_type>ament_python</build_type>
  </export>
</package>
```

ROS2 Package

setup.py

Package Informationen

- `version`
- `description`
- `maintainer`
- `license`

Entry_points für Ausfühung der Nodes

- `entry_points`

```
from setuptools import setup
package_name = 'my_python_pkg'
setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='Name',
    maintainer_email='your@email.com',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
        ],
    },
)
```