University of Applied Sciences
HOCHSCHULE EMDEN·LEER

Mobile Robotics

Prof. Dr.-Ing. Gavin Kane

University of Applied Sciences
HOCHSCHULE EMDEN·LEER

Robot Operating System (ROS)

Lecture Content

- Revision - Communications Concepts
- Revision - Packages
- Publisher / Subscriber
- Messages / Services
- Services / Actions
- Launch Scripts

Robot Operating System (ROS) - Communications Infrastructure

Terminology - ROS Master

- Manages the communication between nodes
- Every node registers at startup with the master

Start a master in MATLAB with:

```
>> rosinit
```

Prepare MATLAB to connect with a master at a given IP Address:

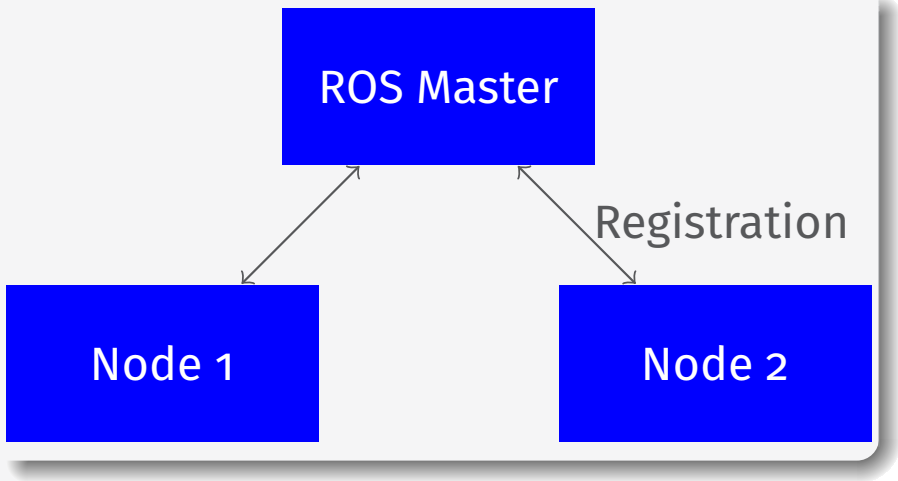
```
>> rosinit('192.168.154.131')
```

Start a master in Linux with:

```
gavin@comp:~$ roscore
```

Prepare Linux to connect with a master on another computer:

```
gavin@comp:~$ export ROS_MASTER_URI=http://min:11311/
```



Robot Operating System (ROS) - Communications Infrastructure

Terminology - ROS Nodes

- Single-purpose, executable program
- Individually compiled, executed, and managed
- Organized in packages

Run a node in MATLAB:

```
>> N = ros.Node(Name)
>>
```

Prepare to run a node in Linux:

```
gavin@comp:~$rosrun <package_name> <node_name>
```

Robot Operating System (ROS) - Communications Infrastructure

Getting information - ROS Nodes

To gain information about what nodes are running, you can use in a Linux Terminal or in Matlab:

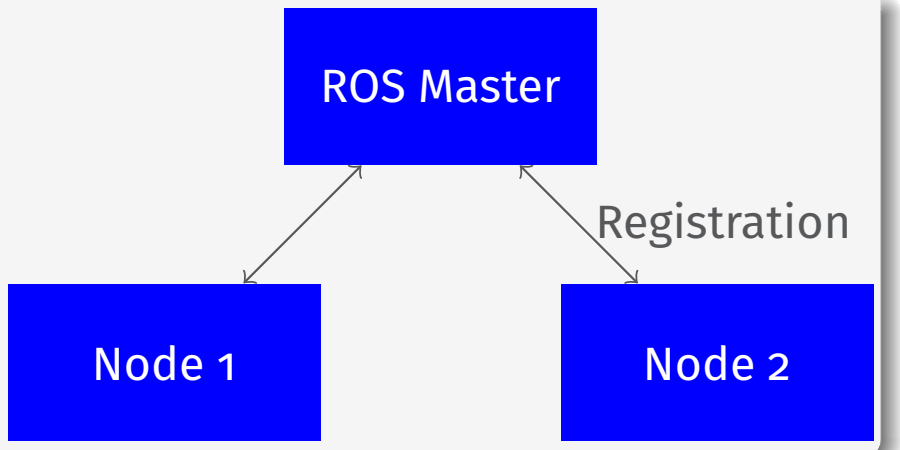
```
gavin@comp:~$ rosnode list
```

or to get a visual view of the nodes and how they are communicating:

```
gavin@comp:~$ rqt_graph
```

To gain information about a specific node:

```
gavin@comp:~$ rosnode info <node_name>
```



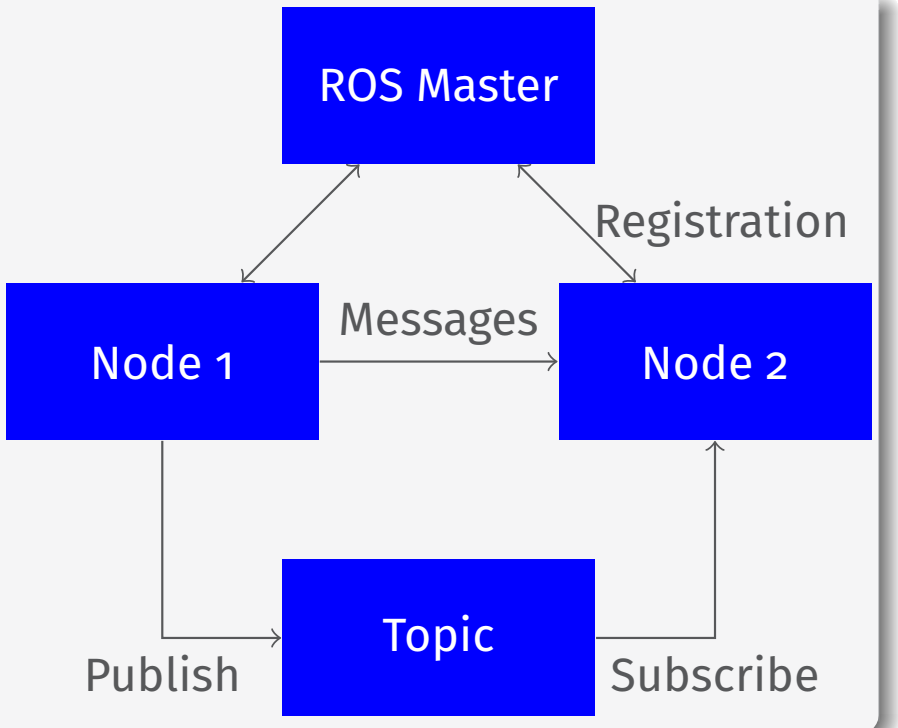
Robot Operating System (ROS) - Communications Infrastructure

Terminology - ROS Topics

- Nodes communicate over topics
- Nodes can publish or subscribe to a topic
- Typically, 1 publisher and n subscribers
- Topic is a name for a stream of messages

List active topics with:

```
>> rostopic list
oder
gavin@comp:~$ rostopic list
```



Robot Operating System (ROS) - Communications Infrastructure

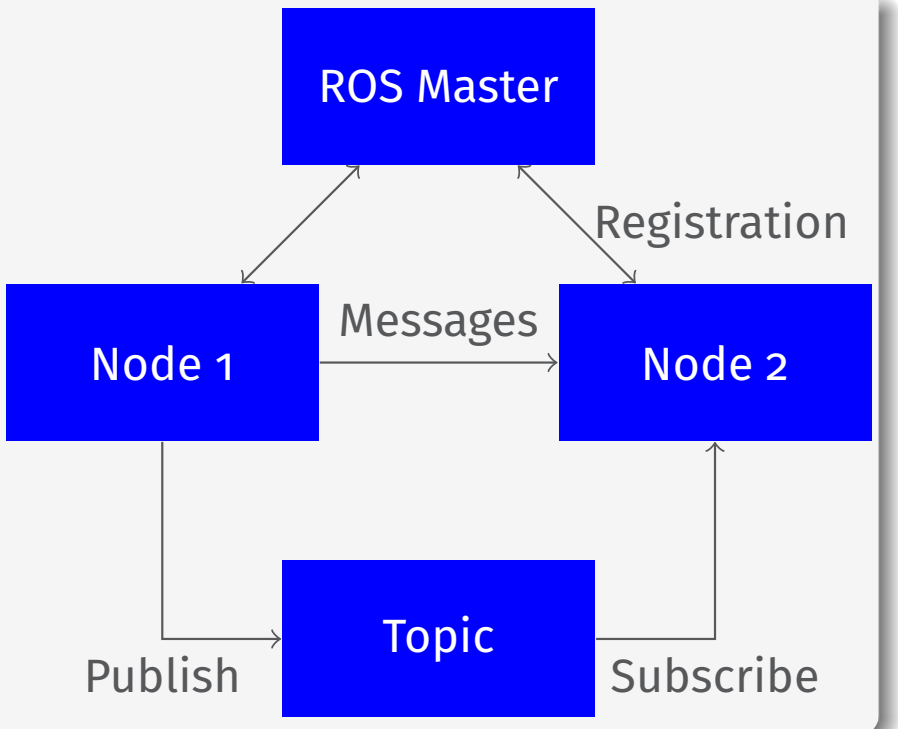
Terminology - ROS Topics

Subscribe and print the contents of a topic with

```
>> rostopic echo /topic
gavin@comp:~$ rostopic echo /topic
```

Show information about a topic with

```
>> rostopic info /topic
gavin@comp:~$ rostopic info /topic
```



Robot Operating System (ROS) - Communications Infrastructure

Terminology - ROS Messages

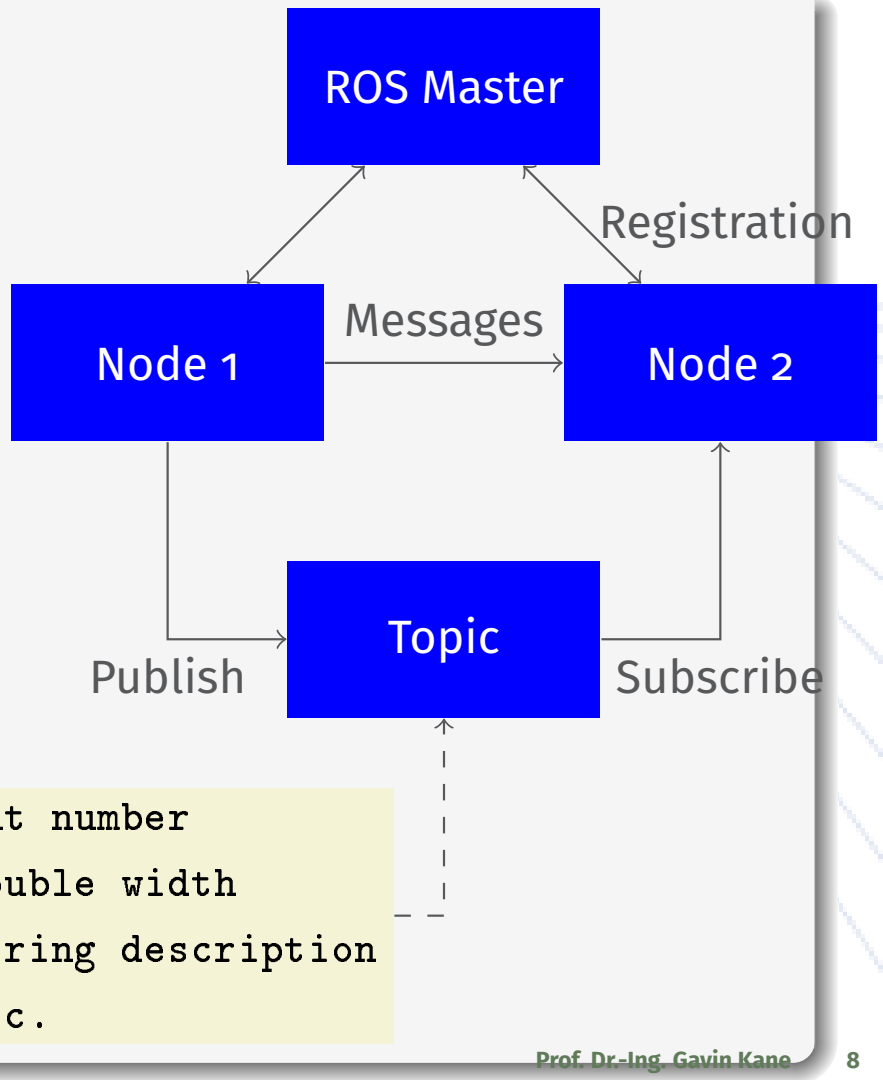
- Data structure defining the type of a topic
- Compromised of a nested structure of integers, floats, booleans, strings etc. and arrays of objects
- Defined in *.msg files

See the type of a topic

```
>> rostopic type /topic
oder
gavin@comp:~$ rostopic type /topic
```

or in matlab

```
>> msg = rosmessage(msgtype);
```



Robot Operating System (ROS)

ROS Messages - Pose Stamped Example

geometry_msgs/PoseStamped.msg

```
std_msgs/Header header
 uint32 seq
 time stamp
 string frame_id
geometry_msgs/Pose pose
 geometry_msgs/Point position
 float64 x
 float64 y
 float64 z
 geometry_msgs/Quaternion orientation
 float64 x
 float64 y
 float64 z
 float64 w
```

geometry_msgs/Point.msg

```
float64 x
float64 y
float64 z
```

Robot Operating System (ROS)

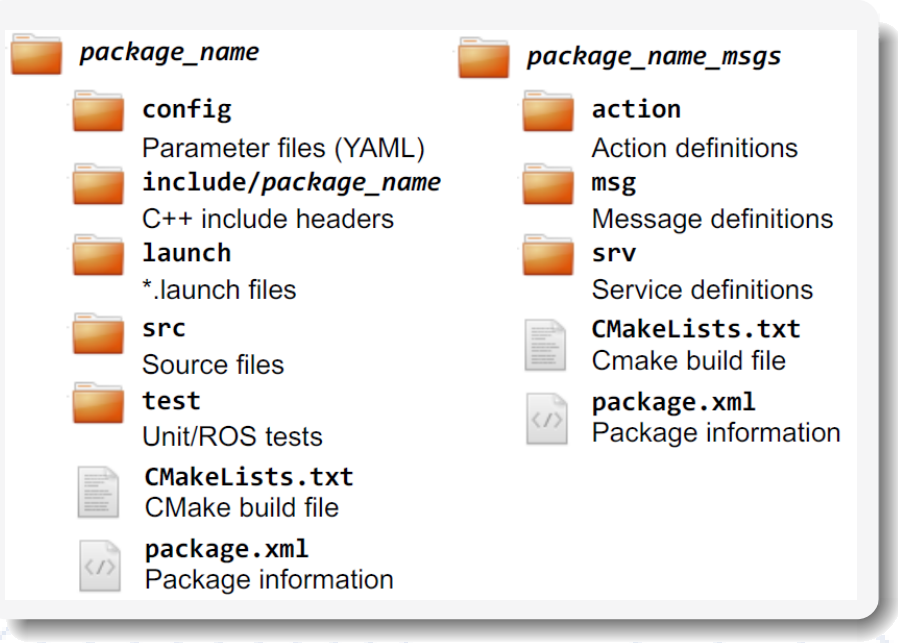
Revision - Workspace Directory Structure

```
/catkin_ws ..... Workspace Base Directory
 ├── devel ..... Location of setup.bash File
 ├── build
 └── src ..... Location where Packages are placed
```

Robot Operating System (ROS)

Revision - ROS Packages

- ROS software is organized into packages, which can contain source code, launch files, configuration files, message definitions, data, and documentation
- A package that builds up on/requires other packages (e.g. message definitions), declares these as dependencies



Robot Operating System (ROS)

ROS Packages

package.xml

- The package.xml file defines the properties of the package
  - Package name
  - Version number
  - Authors
  - Dependencies on other packages

package.xml

```
<?xml version="1.0"?>
 <package format="2">
 <name>ros\_package\_template</name>
 <version>0.1.0</version>
 <description>A template for ROS
    packages.</description>
 <maintainer email="somebody@e...">
    Somebody</maintainer>
 <license>BSD</license>
 <url type="website">https://github.com/...
    </url>
 <author email="Somebody@somewhere.com">
    Somebody</author>
 <buildtool\_depend>catkin
    </buildtool\_depend>
 <depend>roscpp</depend>
 <depend>sensor\_msgs</depend>
</package>
```

Robotic Operating System (ROS)

Topics / Messages / Publishers / Subscribers

- ROS Nodes communicate with Topics
- "Many to Many" one way communications
- No acknowledgment
- No Synchronisation
- Useful for Monitoring

Services

- Request / Response Communication
- Needs to be defined by two message types
- Request Message Type / Response Message Type
- Useful for Event-based execution

Robotic Operating System (ROS)

Message Files

- Placed in the <ros_package_name>/msg folder
- have .msg file type
- are compiled through catkin b when package.xml includes:

```
<build_depend>message\_generation</build\_depend>
  <exec_depend>message\_runtime</exec\_depend>
```

- and CMakeLists.txt includes the generate_messages package in it's find_packages list, and the service message is included in it's "add_message_files" list

MessageTemplate.msg

```
# Example message file
Header header           # Commentary
string child_frame_id  # More commentary
geometry_msgs/PoseWithCovariance pose   #
geometry_msgs/TwistWithCovariance twist #
```

Robotic Operating System (ROS)

Message Definitions

In .msg Files

- **Point** Message Type
  float x
  float y
  float z

Support for Derived Message Types:

- **Pose** Message Type
  - **Position** Message Type
    float x
    float y
    float z
  - **Orientation** Message Type
    float x
    float y
    float z
    float w

Robotic Operating System (ROS)

Message Definitions

Information for message types can be found through the command:

```
$ rosmsg info pose
```

or through the ROS Wiki and Documentation. For example:
`http://docs.ros.org/melodic/api/sensor_msgs/html/msg/LaserScan.html`

Robotic Operating System (ROS)

Service Files

- Placed in the <ros_package_name>/srv folder
- have .srv file type
- are compiled through catkin b when package.xml includes:

```
<build_depend>message\_generation</build\_depend>
  <exec_depend>message\_runtime</exec\_depend>
```

- and CMakeLists.txt includes the generate_messages package in it's find_packages list, and the service message is included in it's "add_service_files" list

ServiceTemplate.srv

```
# Example service file
int64 a  # Request Part 1
int64 b  # Request Part 2
---
int64 sum  # Response Part 1
```

Robotic Operating System (ROS)

Services

- Service Calls are blocking Calls
- Useful for developing sequential programming
- Desirable to have quickly executing computations in service callback

Robotic Operating System (ROS)

Actions

- No waiting until an execution is completed
  - Non-blocking!
  - Waiting is an option if required
- A Generalised Request-Response System
- Actions are defined by three message types:
  - Request (Goal)
  - Response (Result)
  - Feedback

ActionTemplate.action

```
# Example action file
int64 a  # Goal (Request) Part 1
int64 b  # Goal (Request) Part 2
---
int64 sum  # Response Part 1
---
string message # Feedback Part 1
```

Robotic Operating System (ROS)

Action Files

- Placed in the <ros_package_name>/action folder
- have .action file type
- are compiled through catkin b when package.xml includes:

```
<build_depend>message\_generation</build\_depend>
  <exec_depend>message\_runtime</exec\_depend>
```

- package.xml needs to include actionlib and actionlib_msgs as dependencies. CMakeLists needs to include actionlib_msgs in generated messages and in catkin_package
- and CMakeLists.txt includes the generate_messages package in it's find_packages list, and the action message is included in it's "add_action_files" list

Robotic Operating System (ROS)

Launch Scripts

- Normal ROS Nodes are started with rosrun
- roscore needs to be running
- new terminal needed for each rosrun and roscore command
  - roslaunch allows grouping of ros nodes together in one file
  - Code is still in seperate files, just their startup is simplyfied
  - is in xml format
  - started with:

    ```
    $roslaunch <package_name> <launch_file.launch>
    ```

  - accepts arguments and namespaces at execution
  - configures parameters
  - launch packages can be cascaded
  - placed in the <package_name>/launch directory
  - naming convention: <package_name>_<file_name>.launch

Robotic Operating System (ROS)

Example Launch File

```
<?xml version="1.0"?>
<launch>
  <!-- Argument to Launch File -->
  <arg name="argName" default="1.0"/>

  <!-- Start a service Node -->
  <node name="myNode" pkg="packagename"
                      type="nodeScriptName" output="screen">
    <!-- Set a parameter to value given by input argument -->
    <param name="paramName" type="double"
                                    value="$(arg argName)"/>
  <\node>

</launch>
```