4 Modular Research Platform for Robot-Assisted Minimally-Invasive Surgery

4.1 The Robot Operating System (ROS)

The Robot Operating System (ROS) [229] is an open source middleware and a collection of associated software frameworks for modular (distributed) robot software. A focus of the middleware is exchange of streaming data\(^1\) under soft real-time conditions while maintaining a high runtime flexibility. ROS is often used in a research context where heterogeneous software components, often developed in isolation by different people, must work together across multiple computers. Interoperability is ensured by common network message formats, native client libraries for several programming languages (i.a. C++, Python, Matlab, Android, Lisp) and protocol gateways, e.g. JSON over WebSockets. A large number of ROS-wrappers for other large software frameworks such as OpenCV\(^2\), PCL\(^3\) and OMPL\(^4\) are available. There are also drivers for many common sensors, actuators, devices and whole robots available. Furthermore, advanced functionality such as simultaneous localization and mapping (SLAM)\(^5\) and collision free path planning\(^6\) are well integrated with ROS. One big advantage of

\(^1\)The most common use cases assume a fast local network, like Gigabit Ethernet, and message frequencies of up to 1 kHz.
\(^2\)http://opencv.org
\(^3\)http://pointclouds.org
\(^4\)http://ompl.kavrakilab.org
\(^5\)See the navigation stack http://wiki.ros.org/navigation.
\(^6\)See the MoveIt! http://moveit.ros.org/.
ROS compared with other middleware frameworks are the support tools such as 2D and 3D visualization\(^7\) and introspection capabilities. The Gazebo robotics simulator\(^8\) although usable independently of ROS is well integrated with ROS through plugins.

Processes that use the ROS middleware are called *nodes*. There are two basic communication mechanism implemented in ROS:

- **Services**: Synchronous request\&response remote procedure calls (RPCs).
- **Topics**: Asynchronous unidirectional data streams modeled after the publish-subscribe pattern.

Both services and topics are strongly typed. The transported data types are called *messages* and defined in a special interface description language (IDL) from which native types for each programming language are generated. The *roscore*\(^9\) acts as a well-known entry point with naming service and registry. If a node wants to call a service for the first time or subscribe to a topic, it contacts the roscore to receive the IP address and port of the other node (Fig. 4.1). The topic mechanism allows multiple publishers and multiple subscribers on the same topic. This is realized by means of multiple point to point connections.\(^{10}\)

*Actions* built on top of services and topics, provide a communication pattern for long-running requests with intermediate feedback and preemption. A valuable in-depth article on the semantics of communication patterns in ROS and other robotic middleware systems was published by Schlegel et al. [230]. Apart from the means of communication, three further ROS components require a short introduction:

---

\(^7\) Especially the feature-rich rviz tool [http://wiki.ros.org/rviz](http://wiki.ros.org/rviz).

\(^8\) [http://gazebosim.org](http://gazebosim.org)

\(^9\) Actually roscore is a collection of several nodes that act as central registry (ROS master), provide the parameter server and a central logging infrastructure (rosout).

\(^{10}\) By default ROS uses TCP for all connections.