Introduction to ROS

– Robot Operating System –

Daniel Serrano
Department of Intelligent Systems, ASCAMM Technology Center
Parc Tecnològic del Vallès, Av. Universitat Autònoma, 23
08290 Cerdanyola del Vallès
SPAIN

dserrano@ascamm.com

ABSTRACT

This paper gives an introduction to the open-source Robot Operating System (ROS. ROS has been widely adopted by the research robotics community as it provides a powerful software framework to develop autonomous unmanned systems. In this paper, we discuss some of the components that are available on the ROS repository.

1.0 INTRODUCTION

The Robot Operating System (ROS) [1] is an open source robot operating system. It provides libraries and tools to help software developers create robot applications. It offers hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS is similar in some respects to 'robot frameworks,' such as Player, YARP, Orocos, CARMEN, Orca, MOOS, and Microsoft Robotics Studio.

The primary goal of ROS is to support code reuse in robotics research and development. It has been quickly adopted by many robotics research institutions and companies as their standard development framework. The list of robots using ROS is quite extent and it includes platforms from diverse domains, ranging from aerial and ground robots to humanoids and underwater vehicles. To name some, the humanoid robots PR-2 and REEM-C, ground robots such as the ClearPath platforms and aerial platforms such as the ones from Ascending Technologies are fully developed in ROS. Furthermore, a quite extent list of popular sensors for robots is already supported in ROS. To name some, sensors such as Inertial Measurement Units, GPS receivers, cameras and lasers can already be accessed from the drivers widely available on the ROS system.

ROS is licensed under an open source, BSD license. ROS has a repository in which developers can use code that has been already written for robot components like motion planners and vision systems. As ROS is Linux based software, it cannot guarantee the hard real-time properties of a system. However, ROS supports soft real-time applications. In addition, it can be integrated easily with other popular open-source software libraries such as OpenCV, Point Cloud Library, Gazebo simulator, etc.

The ROS runtime "graph" is a peer-to-peer network of processes (potentially distributed across machines) that are loosely coupled using the ROS communication infrastructure. ROS implements several different styles of communication: asynchronous, synchronous and data storage.

2.0 ROS DESIGN PRINCIPLES

ROS was designed to meet a specific set of challenges encountered when developing large-scale service robots but the resulting architecture goes beyond these applications.

ROS is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behaviors across a wide variety of robotic platforms. As a summary, it offers:

- hardware abstraction,
- low-level device control,
- implementation of commonly-used functionality,
- message-passing between processes,
- and package management.
- it also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.

ROS provides a structured communications layer above the host operating systems of a heterogeneous computer cluster. It was design with a philosophy of modular, tools-based software development. It aims at maximizing the reusability of available robot sensor visualizations, sensor fusion and control algorithms. It has gradually managed to gather around a lot of developers and it has become a sort of standard framework for robotic platforms (mostly in research).

ROS Indigo is the eighth ROS distribution release and was released July 22nd 2014. It is primarily targeted at the Ubuntu 14.04 LTS (Precise) release. However, previous versions such as Fuerte and Hydro had great success and are probably still in used in many institutions. Generally speaking, a system may have several versions installed, switch between them and even compatibility across nodes from different versions is possible in certain cases.

Every year ROS is gaining an increasing great representation in the international events and conference related to robotics such as ICRA and IROS. Beside the generic robotics events, ROS organized a yearly conference for ROS developers called ROSCON:

- Roscon 2012 was held in Minnesota, USA, in May 2012.
- Roscon 2013 was held in Stuttgart, Germany, in May 2013.
- Ros Kong 2014 June 6th at Hong Kong University, immediately following ICRA.

The library is natively supported for a Unix-like system (Ubuntu). Support for other operating systems is considered experimental. ROS follows the following design goals:

- **Thin**: designed to be as thin and easy to integrate with other robot software frameworks as possible
- **Language independence**: the ROS framework is easy to implement in any modern programming language, though the most used languages are Python, C++ and Lisp. Experimental libraries are available in Java.
- **Scaling**: ROS is appropriate for large runtime systems and for large development processes.

ROS is not a designed to be a real-time framework, though it is possible to integrate ROS with real-time systems.

3.0 ROS CONCEPTS

The ROS runtime "graph" is a peer-to-peer network of processes (potentially distributed across machines) that are loosely coupled using the ROS communication infrastructure. This means these components are processing data together through several mechanisms. The fundamental concepts of the ROS implementation are nodes, messages, topics, and services.

Nodes

A Node is a software processes that performs computation. It represents a software module or component. ROS is designed to be modular at a fine-grained scale: a system is typically comprised of many nodes. Typical examples are sensor and actuators drivers, algorithms to estimate the state of the platform, user interfaces, etc.

The use of nodes in ROS provides several benefits to the overall system. It reinforce the modularity of the system. Forcing specific functionality to be deployed as individual nodes increase the tolerance to faults. It makes easier the encapsulation of implementation details as opposed to monolithic systems. A minimal API is exposed to the rest of the system that may be based on different technologies as long as they can communicate through the ROS mechanisms.

A node is uniquely identified by its name. Every tool in the ROS framework will refer to a specific node by this identifier. Nodes have a node type (equivalent to their class in object oriented programming). A node type is composed by the name of the executable and the package where it belongs. This imposes some constraints when designing the packages and naming the types of your project.

rosnode is the ROS command line tool to display information about nodes.

Messages

A nodes can communicate to another node (peer-to-peer) by passing messages. A message is a strictly typed data structure.

Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types and constants. Messages can be composed of other messages, and arrays of other messages, nested arbitrarily deep.

The messages are defined in message files, a simple text file to specify the data structure of the message. An example of a message describing a point in the space (geometry_msgs/Point) is as follows:

```
# This contains the position of a point in free space
float64 x
float64 y
float64 z
```

Message types are named using ROS naming convention: package name + message name. ROS automatically generate source code to use this message in a ROS node, both in C++ and python.

rosmsg is the ROS command line tool to display information about messages.

Topics

The data exchange between nodes in ROS follows a publisher/subscriber pattern. A node sends out a message by publishing it to a topic. A topic is just a name that identifies this particular message flow. A node can subscribe to a topic in order to receive this message.

There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics. In general, publishers and subscribers are not aware of each others' existence, which decouples the production of information from its consumption.

A topic is typed by the ROS message type. When a node subscribes to a topic, a type matching check is performed and the communication only occurs if both the publisher and the subscriber of the topic are using the exact same message type.

The topics in ROS can be transmitted using TCP-IP (TCPROS) and UDP (UDPROS). TCP-IP is the default transport used in ROS.

rostopic is the ROS command line tool to display information about topics, including performance statistics such as:

- Period of messages by all publishers (average, maximum, standard deviation)
- Age of messages, based on header timestamp (average, maximum, standard deviation)
- Number of dropped messages
- Traffic volume (Bytes)

Services

Topics are intended for unidirectional, streaming communication. Nodes that need to perform remote procedure calls, i.e. receive a response to a request, should use services instead. Services are the mechanism for request/reply interactions. A providing node offers a service under a name and a client uses the service by sending the request message and awaiting the reply (like RPC).

A Service is defined by a pair of Messages, one for the request and another one for the reply. A ROS node offers a service under a service name. The call from the client is blocking. The node waits for the reply.

Similarly to messages, services are defined in service files, a simple text file that is compiled to automatically generate a stub implementation of the service.

rossrv and rosservice are the ROS command line tool to display information about services.

Master

Provides name registration. Without the master, nodes would not be able to find each other (like a DNS). The role of the Master is to enable individual ROS nodes to locate one another. Once these nodes have located each other they communicate with each other peer-to-peer.

It provides an XMLRPC-based API, which ROS nodes call to store and retrieve information. Primitives such as registerService, registerSubscriber and registerPublisher are available, but this is encapsulated under the client libraries roscpp and rospy. The user does not need to implement the interface at this level.

Parameter Server

It is a shared, multi-variate dictionary that is accessible via network APIs. Nodes use this server to store and retrieve parameters at runtime. As it is not designed for high-performance, it is best used for static, non-binary data such as configuration parameters. It is meant to be globally viewable so that tools can easily inspect the configuration state of the system and modify if necessary.

The Parameter Server is implemented using XMLRPC and runs inside of the ROS Master, which means that its API is accessible via normal XMLRPC libraries.

Bags

A format for saving and playing back ROS messages. Widely used for developing and testing algorithms.

rosbag is the ROS command line tool to display information about services.

4.0 ROS MESSAGES TYPES

The following table shows the built-in primitive types (and relates then with C++ and Python primitive types):

Primitive Type	Serialization	C++	Python
bool	unsigned 8-bit int	uint8_t	bool
int8	signed 8-bit int	int8_t	int
uint8	unsigned 8-bit int	uint8_t	int (3)
int16	signed 16-bit int	int16_t	int
uint16	unsigned 16-bit int	uint16_t	int
int32	signed 32-bit int	int32_t	int
uint32	unsigned 32-bit int	uint32_t	Int
int64	signed 64-bit int	int64_t	long
uint64	unsigned 64-bit int	uint64_t	long
float32	32-bit IEEE float	float	float
float64	64-bit IEEE float	double	float
string	ascii string (4)	std::string	string
time	secs/nsecs unsigned 32-bit ints	ros::Time	rospy.Time
duration	secs/nsecs signed 32-bit ints	ros::Duration	rospy.Duration

Some other message types are defined and maintained by ROS to ensure interoperability across different ROS nodes. ROS defines standard message types for commonly used robot sensor data such as images, inertial measurements, GPS, odometry, etc. for communicating between nodes. Thus separate data structures do not need to be explicitly defined for integrating different components. However, these messages have been created on demand and they are continuously evolving as new needs are identified. These messages are grouped under the package ROS **common_msgs.**

For instance, geometry_msgs provides messages for common geometric primitives such as points, vectors, and poses. These primitives are designed to provide a common data type and facilitate interoperability throughout the system. The following message types are defined under geometry_msgs:

- Point
- Point32
- PointStamped
- Polygon
- PolygonStamped
- Pose
- Pose2D
- PoseArray
- PoseStamped
- PoseWithCovariance
- PoseWithCovarianceStamped
- Quaternion
- QuaternionStamped
- Transform
- TransformStamped
- Twist
- TwistStamped
- TwistWithCovariance
- TwistWithCovarianceStamped
- Vector3
- Vector3Stamped
- Wrench
- WrenchStamped

The sensor_msgs defines messages for commonly used sensors, including cameras and scanning laser rangefinders. The following sensor message types are defined:

- CameraInfo
- ChannelFloat32
- CompressedImage
- FluidPressure
- Illuminance
- Image
- Imu
- JointState
- Joy
- JoyFeedback
- JoyFeedbackArray
- LaserEcho
- LaserScan
- MagneticField
- MultiDOFJointState
- MultiEchoLaserScan
- NavSatFix
- NavSatStatus

- PointCloud
- PointCloud2
- PointField
- Range
- RegionOfInterest
- RelativeHumidity
- Temperature
- TimeReference

There are several research groups that their own repository of ROS developments and propose some more detailed definition of messages specific to their algorithms. These messages typically build upon the common_msgs already existing in ROS.

5.0 THE TF LIBRARY

ROS provide several very useful commands and tools to navigate and use the system. Of particular interest is the *tf* library, to represent transformations between coordinates frames over time. *tf* let the user transform points, vectors etc between any two coordinate frames at any desired point in time. Setting up the correct *tf* for sensors in a robot is a critical step for every new configuration before testing any software node.

The tf library [4] was designed to provide a standard way to keep track of coordinate frames and transform data within an entire system such that individual component users can be confident that the data is in the coordinate frame that they want without requiring knowledge of all the coordinate frames in the system.

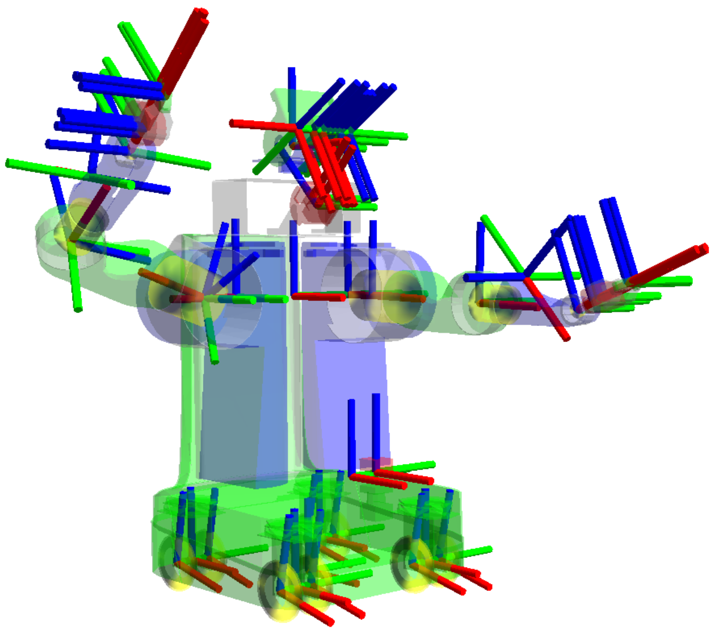


Figure 1: tf frames in the PR2 robot [4].

The complexity of this task made it a common place for bugs when developers improperly applied transforms to data. The problem is also a challenge due to the often distributed sources of information about transformations between different sets of coordinate frames.

6.0 RVIZ

RVIZ is the 3D visualization tool in ROS. The following data types are supported by default:

- Axes
- Effort
- Camera
- Grid
- Grid Cells
- Image
- InteractiveMarker
- Laser Scan
- Map
- Markers
- Path
- Point
- Pose
- Pose Array
- Point Cloud(2)
- Polygon
- Odometry
- Range
- RobotModel
- TF
- Wrench
- Oculus

RVIZ lets the user select the adequate configuration for each specific robot, allowing fairly advance functionality as shown in the following image:

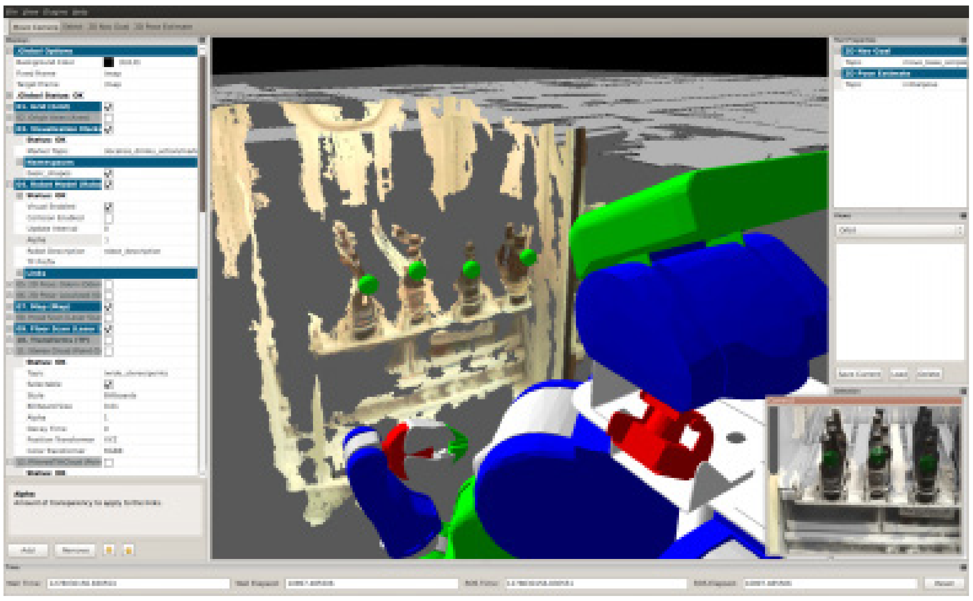


Figure 2: Robonaut2 robot in RVIZ.

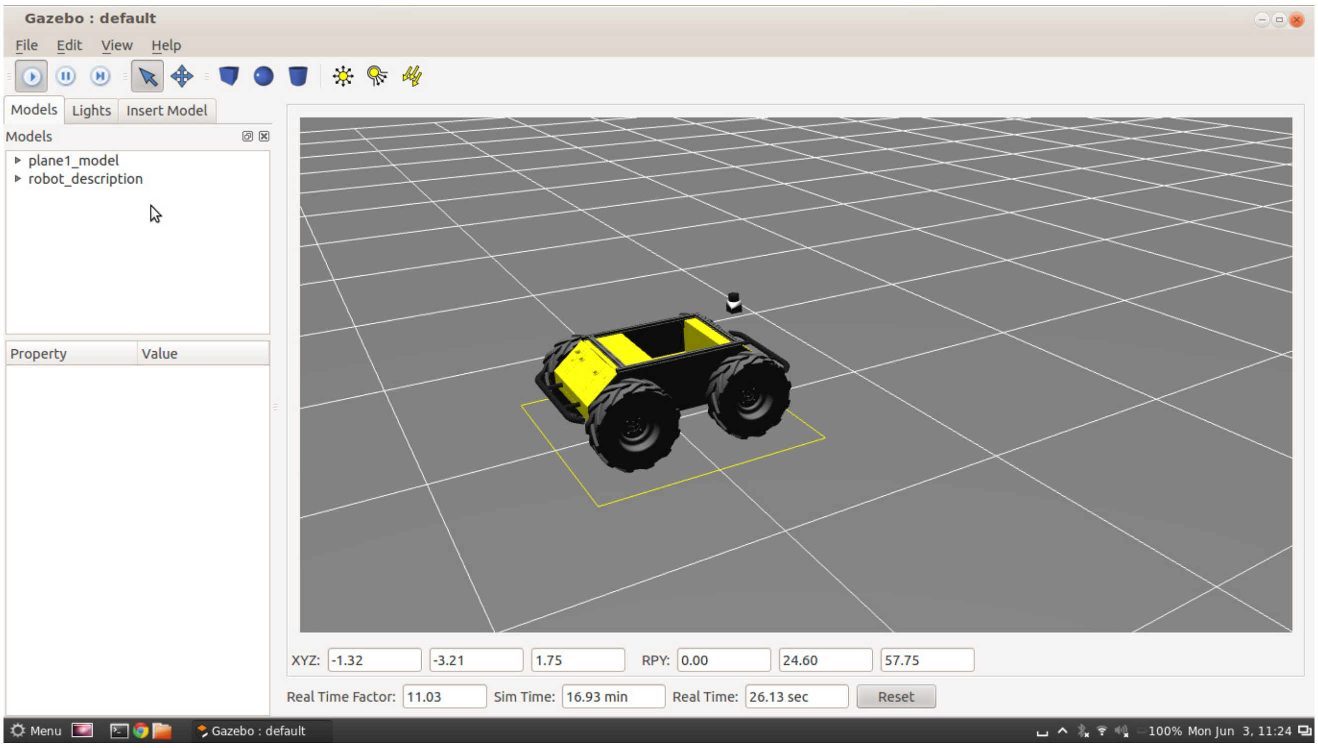
Furthermore, the user can customize the RVIZ interface by developing application-specific plugins.

7.0 OTHER USEFUL LIBRARIES

An advantage is that it natively integrates a set of tools and libraries commonly used in Robotics. For instance:

Gazebo simulator [5]

Gazebo is a 3D indoor and outdoor multi-robot simulator, complete with dynamic and kinematic physics, and a pluggable physics engine. Integration between ROS and Gazebo is provided by a set of Gazebo plugins that support many existing robots and sensors. Because the plugins present the same message interface as the rest of the ROS ecosystem, you can write ROS nodes that are compatible with simulation, logged data, and hardware. You can develop your application in simulation and then deploy to the physical robot with little or no changes in your code. The following image shows the Husky ground robot by Clearpath Robots simulated in Gazebo and automatically accesible from ROS:



XYZ: -1.32 -3.21 1.75 RPY: 0.00 24.60 57.75

Real Time Factor: 11.03 Sim Time: 16.93 min Real Time: 26.13 sec

Figure 3: Clearpath Husky simulation in Gazebo.

Point Cloud Library (PCL) [6]

PCL, the Point Cloud Library, is a perception library focused on the manipulation and processing of three-dimensional data and depth images. PCL provides many point cloud algorithms, including filtering, feature detection, registration, kd-trees, octrees, sample consensus, and more. If you are working with a three-dimensional sensor like the Microsoft Kinect or a scanning laser, then PCL and ROS will help you collect, transform, process, visualize, and act upon that rich 3D data.

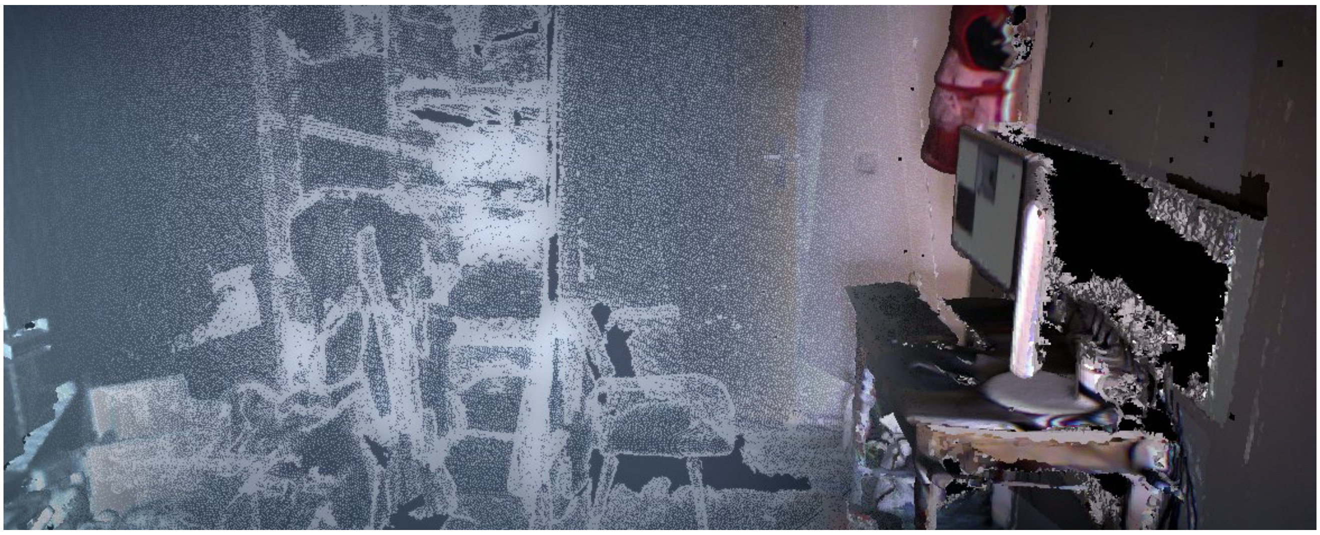


Figure 4: PCL visualizer integrated in ROS.

OpenCV [7]

OpenCV is the premier computer vision library, used in academia and in products around the world. OpenCV provides many common computer vision algorithms and utilities that you can use and build upon. ROS provides tight integration with OpenCV, allowing users to easily feed data published by cameras of various types into OpenCV algorithms, such as segmentation and tracking. ROS builds on OpenCV to provide libraries such as image_pipeline, which can be used for camera calibration, monocular image processing, stereo image processing, and depth image processing. If your robot has cameras connected through USB, Firewire, or Ethernet, ROS and OpenCV will make your life easier.

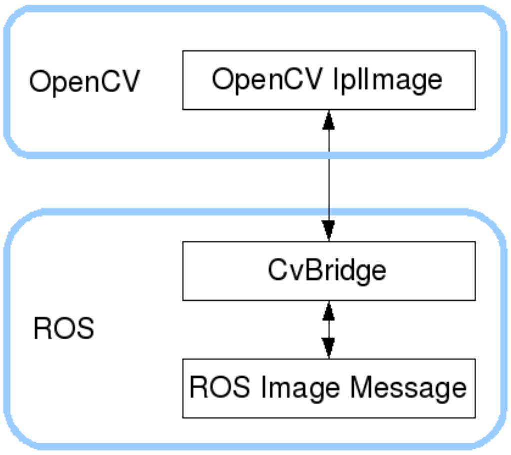


Figure 5: OpenCV integration in ROS via CvBridge.

MoveIt! [8]

MoveIt! is a motion planning library that offers efficient, well-tested implementations of state of the art planning algorithms that have been used on a wide variety of robots, from simple wheeled platforms to walking humanoids. Whether you want to apply an existing algorithm or develop your own approach, MoveIt! has what you need for motion planning. Through its integration with ROS, MoveIt! can be used with any ROS-supported robot. Planning data can be visualized with rviz and rqt plugins, and plans can be executed via the ROS control system.

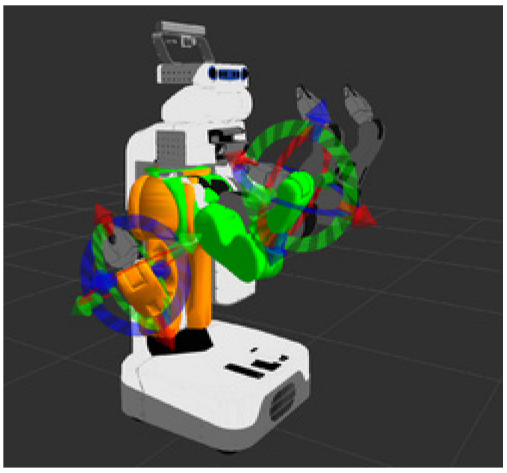


Figure 6: MoveIt! integrated in ROS.

REFERENCES

[1] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. "ROS: an open-source robot operating system". In ICRA Workshop on Open Source Software, 2009.

[2] ROS website *http://wiki.ros.org/*

[3] A. Martinez, E. Fernández. "Learning ROS for Robotics Programming". Packt Publishing. 2013.

[4] T. Foote. "tf: The Transform Library", IEEE International Conference on Technologies for Practical Robot Applications (TePRA), 2013

[5] Gazebo Simulator *http://gazebosim.org/*

[6] Point Cloud Library *http://pointclouds.org*/

[7] OpenCV library *http://opencv.org/*

[8] MoveIt! library *http://moveit.ros.org*/