**3. ROS**

**3.1 Fundamentals**

Robot Operating System (ROS) framework has shown an increasing popularity at robotics applications since it was first released at 2009 by Willow Garage. Even though it has “operating system” in its name, ROS is not an actual operating system. It might be classified as a *framework* or a *middleware* that serves various useful tools.

The hardware on the present-day robots differs broadly. To prevent writing codes again for same or similar tasks on robots with different hardware, or in others words to prevent reinventing the steel, by providing an environment is the basic logic behind the ROS.

ROS is designed to be peer-to-peer, tool based, multi-lingual, thin, free and open-source [Quigley09]. Every ROS application consists of computational unit or programs named *nodes* and communications and relationships of them. The *nodes* can communicate over *topics* by passing certain data named *messages.*

ROS is based on four fundamentals; passing messages by publishing or subscribing to topics, passing messages using services, recording messages and playing-back when necessary and having a dynamically reconfigurable distributed parameter server. Using these core features, it is possible to perform many simple and complex tasks[rosCoreComponents].

**3.2 A Brief Review**

ROS project officially supports Ubuntu and pre-compiled ROS distributions are supplied officially. However, since it is an open-source project, its source files are available and it is possible to compile them on similar platforms. Hence there are experimental pre-compiled repositories available such as the one for the Raspbian operating system of Raspberry Pi development boards [rosRaspbian].

ROS file system is based on unit software organizations called *packages* and organization of related packages called *metapackages* [rosFileSystem]. There used be *stack* organization but they have replaced with *metapackages* in the newer ROS distributions. Every package can contain libraries, executables, source codes, launch files, scripts and so on.

ROS uses a build system called *catkin* which combines CMake build system with Python codes. A build system is the system that constitutes target files which might be executables, libraries, header files from the source code [rosCatkin]. ROS changed its build system to *catkin* on and after the Groovy distribution.

As mentioned in the previous section, one of the design goals of ROS was to be multi-lingual. ROS supports C++ and Python. So the nodes can be written in both C++ and Python and a node that has written in C++ can communicate with the one that has written in Python and vice versa.

ROS does not only provide features to make the nodes communicate in certain methods, but it also provides several tools to make diagnostics and debug. For example, there are tools to see the nodes and topics as a graph, to plot messages according to time or to find out the message passing frequency on a topic. Also there are simulation and visualization environments working with ROS such as Rviz and Gazebo.

ROS framework forms an environment for reuse the codes. There are many message type definitions for common messages and also it is easy to create custom messages. Many sensors and hardware have their ROS libraries and drivers which makes easier and faster to install various hardware. Finally, ROS has an involved and wide community. All these features make ROS preferable and usable.