ROS

Robot Operating System

Introduction and quick start...

~ Robotics Club, IIT Kanpur ~

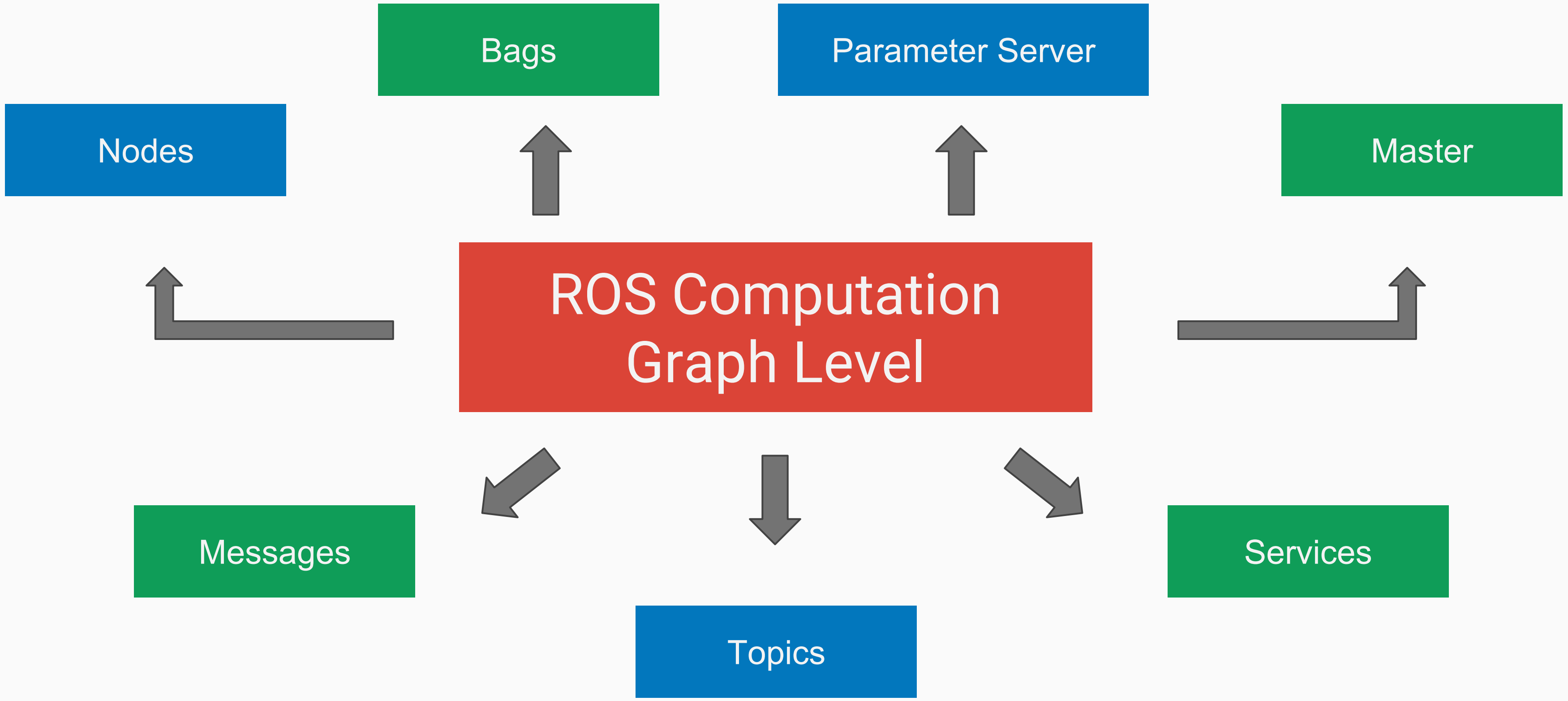
What is ROS??

ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. A ROS system is comprised of a number of independent nodes, each of which communicates with the other nodes using a publish/subscribe messaging model. For eg a sensor driver can be implemented as a node which publishes sensors data as messages. These messages can be used by number of other nodes which requires them.

Why ROS??

- ❖ Open-Source and freely available.
- ❖ Huge community and collaborative environment.
- ❖ Language Independent (Python, C++, Java, JS, more…)
- ❖ Thousands of available packages to use for robotics!!
- ❖ Seamless integration with most of the open-source projects and libraries. Eg: OpenCV, Gazebo, Matlab(using Robotics System Toolbox), etc.
- ❖ ROS is a distributed framework of processes (aka Nodes) that enables executables to be individually designed and loosely coupled at runtime.

The ROS System (general concept)



- **Nodes:** Nodes are processes that perform computation. Each ROS node is written using ROS client libraries such as roscpp, rospy, etc. In a robot system there will be numbers nodes to perform different tasks. The ROS communication methods helps them to communicate and exchange data. In general nodes contains the algorithms for different task to accomplish and all different nodes shares their result and datas using the ROS communication methods.

- **Master:** The ROS Master provides name registration and lookup to the rest of the Computation Graph, something like a DNS Server. It allows all other ROS Nodes to find and talk to each other.

- **Parameter Server:** The Parameter Server allows data to be stored by key in a central location. It is currently part of the Master. All nodes can access and modify these values.

- **Messages:** Nodes communicate with each other by passing messages. A message is simply a data structure, comprising typed fields. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types. Messages can include arbitrarily nested structures and arrays (much like C structs). We can also build messages of our own type using these standard message definitions.

- ❖ **Topics:** Messages in ROS are transported using topics. Topic is a name used identify the type of messages can be transported through it. Any no. of nodes can publish or subscribe messages to it irrespective what other node being doing i.e. nodes are unaware of who are subscribing and who are publishing the messages to the given topic they just know the info. about themselves, which ensures the independent working of each and every node.

- ❖ **Services:** The publish / subscribe model is a very flexible communication paradigm, but it's many-to-many, one-way transport is not appropriate for request / reply interactions, which are often required in a distributed system. Request / reply is done via services, which are defined by a pair of message structures: one for the request and one for the reply. A providing node offers a service under a name and a client uses the service by sending the request message and awaiting the reply.

- ❖ **Bags:** Bags are a format for saving and playing back ROS message data. Bags are an important mechanism for storing data, such as sensor data, that can be difficult to collect but is necessary for developing and testing algorithms.

Terminology

Now lets brief about some conceptual things i.e. how things work around. Nodes connect to other nodes directly; the Master only provides lookup information, much like a DNS server. Nodes that subscribe to a topic will request connections from nodes that publish that topic, and will establish that connection over an agreed upon connection protocol. You can think of Master as a Table of topics and the info about which nodes are subscribing to those topics. Each publisher's goes and find the topics which matches with their message type and send the messages directly to the nodes which are subscribed to those topics. These all works on callback mechanism i.e. whenever a publisher wants to publish something the master sends a callback to the subscriber to be ready to receive the message. Now come to service client module every services are registered with the master to receive query to deal with. The client sends a request to those service and wait for response when the service reply back with a response the query ends there. We can use this method to get certain data at certain time or to create a method to know whether the message which we ought to send to a node had been received by it or not.

Let's begin with how to create packages, nodes, and other ROS stuffs...

Catkin Package

Follow up this link to create catkin package. Now let’s discuss about different parts of package.

- /msg: This contains custom message files.
- /srv: This contains custom service files.
- /src: This contains the C++ executables.
- /package.xml: Contains information about the package, author, license, dependencies, compilation flags, and so on.
- /CMakeLists.txt: Contains information about the cmake tool to for building the package

Creating custom msg and srv files.

ROS allows you to create your own custom msg and srv files. It is similar to structs in C.

- ❖ Message: extension of file is “ .msg “ , saved inside msg folder. Eg.

```
Demo.msg
    int32 number
    string word
```

- ❖ Service: extension of file is “ .srv “, saved inside srv folder. Eg.

```
Demo.srv
    int32 req
    ---
    int32 res
```

Writing C++ executables, CMakeLists.txt and package.xml

- For creating C++ executables at first simply write your C++ code as you want and then follow up the procedure to convert it into a ROS supported code. Follow up this link
- Adding dependencies in package.xml. Follow this link
- You must know how to edit CMakeLists.txt and always use the codes in the sequential order as changing the sequence for CMakeLists will create errors while building your package. Follow up this link

Creating Arduino Node

First of all write the code according to the format given in this file. You can also check many other example codes inside the Arduino IDE under ros_lib section. After writing the code upload it to the Arduino using the IDE. Run the below command to open the node.

```
$ rosrun rosserial_python serial_node.py /dev/ttyACM0
```

Here /dev/ttyACM0 is the port to which your arduino is connected. Interested one can also check this package which was made by me to control your arduino board from your laptop during the runtime.

Thank You...