ROS

ROS - Robot operating system. [in Linux only]

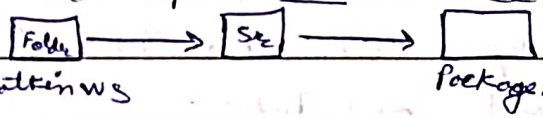
→ In Ros we subscribe and publish data on topic through different sensor and script and use or manipulate the data according to our ~~this~~ wish as usc.

For ex: Taking data from LIDAR and using it to build a enviornment for the car.

* Tasks in Ros:

1) creating package and make file executable:

(i) create a workspace folder and src folder in it and open cmd there.



(ii) now use ~~catkin_init_workspace~~ ~~catkin_create_package~~

$ catkin_create_package <package_name> dependend module.
(ex rospy, roscpp)

(iii) now go to the workspace folder and type
$ catkin_make.

(iv) To use this package we have to source the workspace.
$ source ~/<workspace folder>/devel/setup.bash.

* Some Important point for using ros:

1) we use ros through command prompt (cmd) and all action is done on it only.
2) we have install the ros first to use it all the resources are on 'wiki.ros.org' or google 'ros tutorials'.
3) ~~shortcut~~ shortcut for cmd is ctrl+alt+T.
4) The '$' sign represent the cmd prompt. which is reflected automatically and we don't need to right it.
5) The things written in <> should be replaced and '<>' sign should be removed.

→ we need to make all python or c++ files to executable to run them for that we need to open cmd in the folder where code is present and type.

$ chmod +x <file name> [chmod → change mod]

$ chmod +x * [For all files in folder]

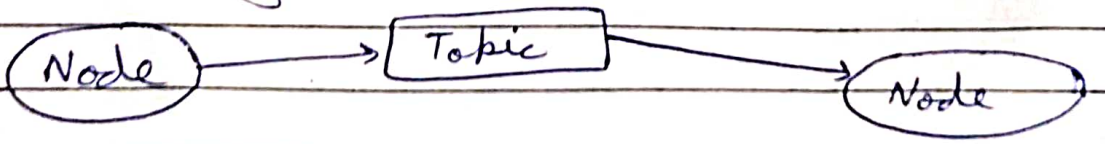
→ Here +x is file permission and + represent the add permission.

r → read

w → write

x → executable.

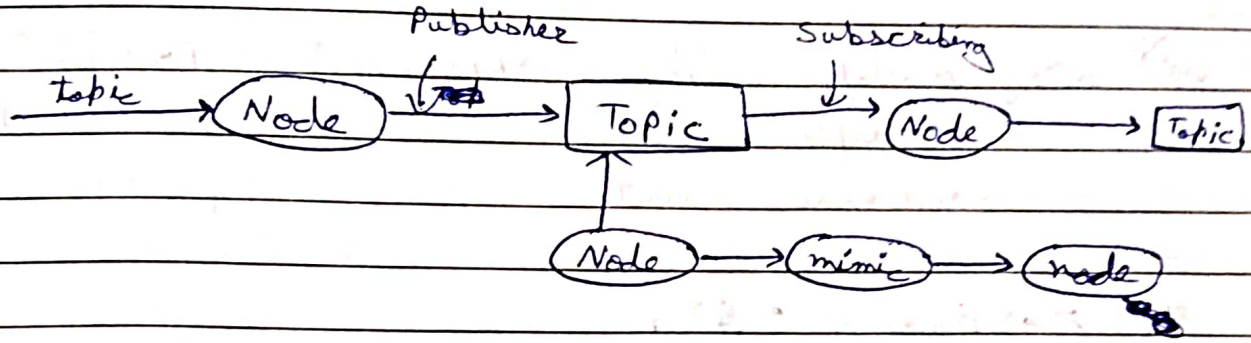
* Ros working:



Ros master:

→ To use ros we first have to launch ros master which handel all the topics and nodes. for that we need to type 'roscore' in seprate terminal.

it can be like this also.



↦ Node: they are the scripts which subscribe or publisher to topic. ~~For every topic code we write we need to~~.

→ For every subscriber or Publisher we need to initiate node and give a unique name to it.

↦ Topic: Different nodes interact with each other using topic. The messages are printed on topic which can later used by subscriber.

* Ros service:

→ To debug the code or better understand the functionic more properly we can use services provided by ros.

<Ros service> list	print ~~information~~ list of active service
<ros service> call	call service with provided args.
<ros service> type	print service type
<ros service> find	find service by service type
<ros service> uri	print service RosRPC uri

(rosservice) info — print the information about service.

Imp ex:→ rostopic info \topic name } Print sub/Pub etc about topic
ros topic list } print all topic (active)
ros node list } print all active nodes.

* **rosparam:**
→ it is use to do operation on the parameter file to edit constant values.

ros param	set	
"	get	
"	load	load parameter from file
"	dump	dump parameter to file.
"	delete	
"	list	

* **rostopic:**
→ it is tool to use or understand topic in ros.
it have diffrent function like.

	rostopic	bw	display bandwidth of topic
Imp	"	echo	print messages publishing on topic
	"	hz	display publishing frequency
Imp	"	list	print list of all active topic
	"	pub	publish data to topic
	"	type	print topic type
Imp	"	info	print all information of topic

* Launch file in ros :

→ We can create a launch file in ros which on launching ; launches multiple nodes at same time .

→ In other words, instead of using rosrun for each node or file we can create a launch file which automatically launches all the specified nodes in it .

* Steps to create launch file :-

① create a folder named 'launch' in package .

② create a file named '<filename>.launch' in launch folder . [folder name can be anything for launch files] .

Structure of launch file :-

```
<launch>
  <node name = '<name to give node>' pkg = '<package name in which node is present>'
    type = '<name of node to be launched>' />
    :
    :
  <!-- to add comment -->
  <!-- to change the topic to which the
       node is mapped we use remap tag -->
```

we can close the tag by puting / at the end of tag in single line .

→ after the node name use remap tag to map the node to another topic .

```
<remap from = "<previous topic name>' to = '<new topic>' />
```

→ To include a file in launch file.

<include file = '<file location>' />

→ we can also give parameters with while including a file by rosparam.

<rosparam file = '<file location>' command = '<>' />.

→ at the end
</launch>

---

* writing a subscriber or Publisher :-

↦ ~~Subscriber~~ Publisher → it Publish messages on a topic
↦ subscribers → it reads or subscribes messages of a topic.

Note: ① A script or code / ~~node~~ node have multiple subscribers and Publishers but a script can have single node.
② A topic can have multiple publishers as well as multiple subscriber.

● structure of Publisher :

meaning of code.

```
#!/usr/bin/env python3
import rospy
from std_msgs.msg import String
def talker():
    pub = rospy.Publis
```

{ tell about the interpreter of code
{ import necessary objects
{ import the data type to be Published. we can create our own data type to be published.
→ it is similar to the variable declaration in class for objects.

def func():
created an Publisher object ← pub = rospy.Publisher('<topic name>', String, queue_size = 10)
(String → Datatype to be published; queue_size → enter data size.)
initiate the node for script ← rospy.init_node('<node name>', anonymous = True)
(anonymous = True → make sure that node name is unique by adding random sequence)
⋮ code

rospy.loginfo('...') → print message on screen, the printed message will not be published by this.

pub.publish('...') → publish message on topic.

rospy.spin() → execute function till the node is not kill or closed.

func()

- Structure of subscriber:

```
#!/usr/bin/env python3
import rospy
from std_msgs.msg import String

def funct():
    rospy.init_node('node name', anonymous=True)
    rospy.subscriber('<topic name>', String, call back)

    rospy.spin()

def call back(data):

    rospy.loginfo(data.data)

funct()
```

(String → data type printed on topic.)
(call back → function to be called for subscriber action.)
(data.data → we can use data published on topic by data.data.)

Note: To use the script of Publisher and subscriber. we have to do following steps:

1. create folder named 'scripts' or 'src' in package.
2. now make subscriber or publisher in the folder.
3. Now, open cmd in folder and make file to executable by 'chmod'.
4. Now, use rosrun or roslaunch to execute publisher or subscriber.