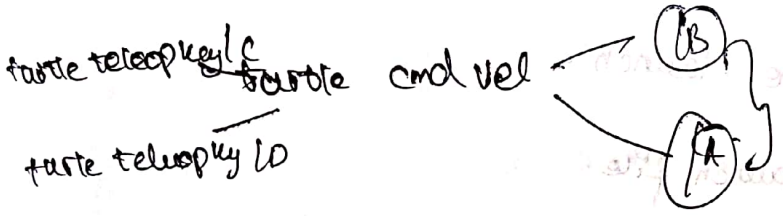
Ros

* Ros nodes communicate through msgs (in topics).

Ros master does not allow multiple nodes with same name.

rosrun turtlesim turtlesim_node __name:= A

" " " - node __name:= B



Every topic has a msg type associated with it

turtlesim/color → A msg type

↓ ↓

package type name.

geometry_msgs / twist

Create a catkin workspace.

and in a pkg create a pkg in src folder.

Create a scripts directory and create a node.

touch node_hello_ros.py

make it executable.

Command:

rosmsg show geometry_msgs / Twist

* Displays info about the message type

* Node names are different from name of the executable

executable is created within.

You can create a launch file to directly launch the node

create a launch file inside a pkg using.

touch filename : launch

And edit the launch file.

< launch >

<node pkg = "name of package" type = "name_of_executable

.py

name = "name of node

</launch/>

rosmsg show std_msgs/string

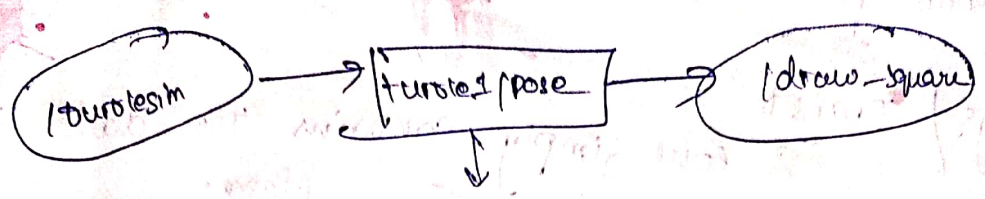
rostopic info /chatter

* shows info like publishers, subscribers.

rostopic echo /chatter

* msgs inside the topic

rostopic - A carrier of msgs from ros publisher to subscriber.



Orientation &
pose of turtlebot

rosmsg show geometry_msgs/Twist

geometry_msgs/Vector3 linear
- float64 x
- float64 y
- float64 z

field
constant

rostopic pub ~~topic~~ /turtle1/cmd_vel geometry_msgs/Twist

linear
  x: 0.0
  y: 0.0
  z: 0.0

angular:
  x: 0.0
  y: 0.0
  z: 0.5

Now the ~~rot~~ bot rotates for only one time bcoz pub is sent only once.

So use pub -r 10 Hertz.
[Publish the msg at 10 Hz].

rostopic list --help

~~rostopic echo~~ → [get to know further info about rostopic list]

list -p

are present for our custom msg

geometry_msgs / Twist → Twist simply contains the data fields.

We can also create a custom message file like. Twist

in our pkg-ros-basics.

? How to use that msg file ?

Use from geometry-msgs.msg import Twist

Eg: from pkg-ros-basics.msg import myMessage

Every msg file is stored in msg folder of a pkg.

where data types are present.

Turtlesim node for the turtlesimulation and I need to create a publisher node for this subscriber.

Turtlesim node will always be a subscriber of turtle cmd vel topic

publish / subscribe model is many to many one-way transport.

Use callback functions for subscriber nodes.
Use queue size for publisher nodes.

In ros services, client should wait for server to acknowledge the request msg. So we use Ros action
while server is processing the goal. So client can work on other things.

We use services for two-way communication

Actions for simple interface

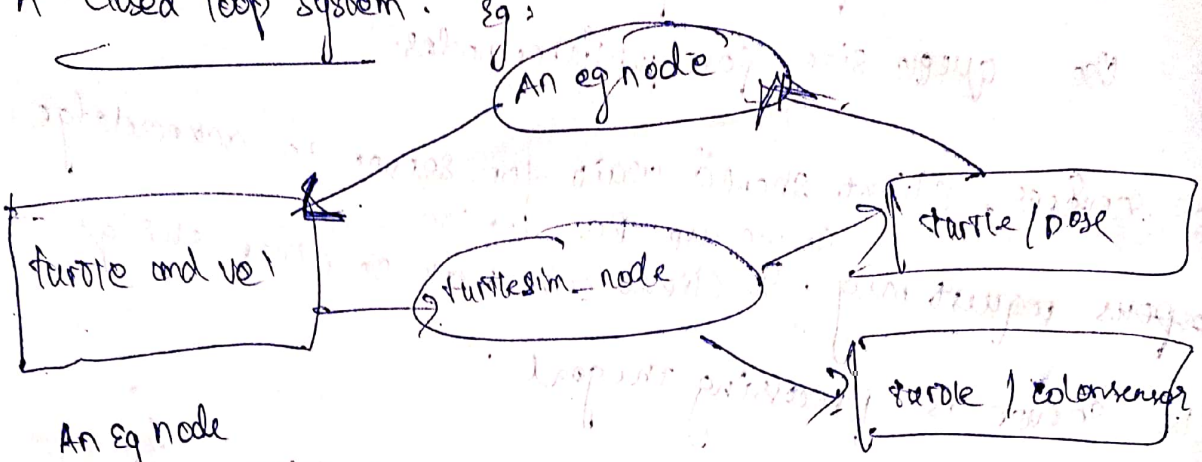
i) * Goal - Eg: a Pose stamped message about where the robot should move
ii) * Feedback - tells about the progress in goal
iii) * Result - Completion of goal

Action client will send a goal
Action server - sends Feedback while it is processing the goal
Action Server - send result after processing the goal.

Simple Action Server :- Can handle only one goal at a time.

Eg: turtlesim-node is a subscriber for turtle cmd vel topic
and publisher of for turtle pose & color topic

A closed loop system. Eg:



An eg node

Subscribes from pose topic and publishes it to to turtle cmd_vel.

So this eg node has python code for both subscriber and publisher.

Callback function is called when a subscriber receives a message from a topic.

rospy.spin is used in Subscriber ~~is used in~~ to process all the callbacks as well as to keep the node alive.

In ros services, the request and response are separated by three hyphen.

Eg: add_two_ints service :- int64 a
int64 b
---
int64 sum

rosservice list

rossrv show [just like msg]

rosservice call, do change the parameters in server.

Ros service - can be analagous to HTTP URL

Request from client - Response from server

So we create a service client and service server

To take input from keyboard:-

import sys

function [float(sys.argv[1]), float(sys.argv[2])]