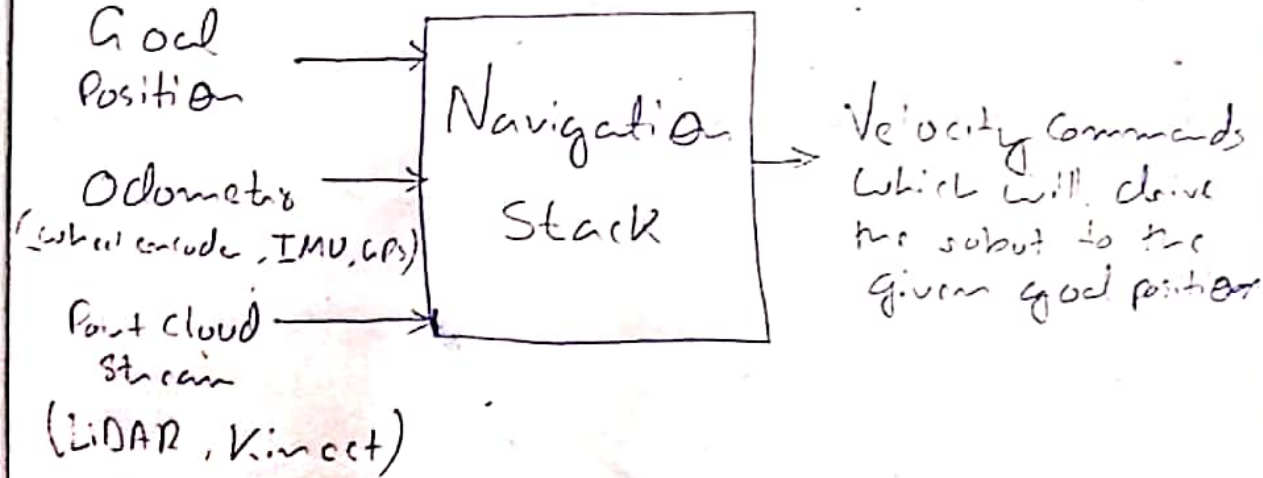
Navigation Stack

⇒ Mobile robot navigation.

- Simultaneous Localization and Mapping (SLAM)
- Adaptive Monte Carlo Localizatio (AMCL)

* Understanding ROS Navigation stack

⇒ The main aim of the ROS navigation package is to move a robot from the start position to the goal position, without making any collision with the environment.



Inputs to Navigation Stack

◉ Odometry Source

⇒ Odometry data of a robot gives the robot position with respect to its starting position.

⇒ Main odometry source(s) :-

- → Wheel encoders
- → IMU
- → 2D/3D Cameras. (Visual odometry)

⇒ The odom value should publish to the Navigation Stack, which has a message type of nav_msgs/odometry.

↓

{Can hold position & Velocity of robot}

◉ Sensor Source

⇒ We have to provide laser scan data or point cloud data to the Navigation stack for mapping the robot environment.

⇒ This data, along with odometry, combines to build global and local cost map.

⇒ The data should be of type sensor_msgs/LaserScan or sensor_msgs/PointCloud

⊙ Sensor transforms /tf

⇒ The robot should publish the relationship between the robot coordinate frame using ROS tf.

⊙ Base_Controller

⇒ The main function of the base Controller is to convert the output of the Navigation Stack, which is twist (geometry_msgs/Twist) message, and convert it into corresponding motor velocities of the robot.

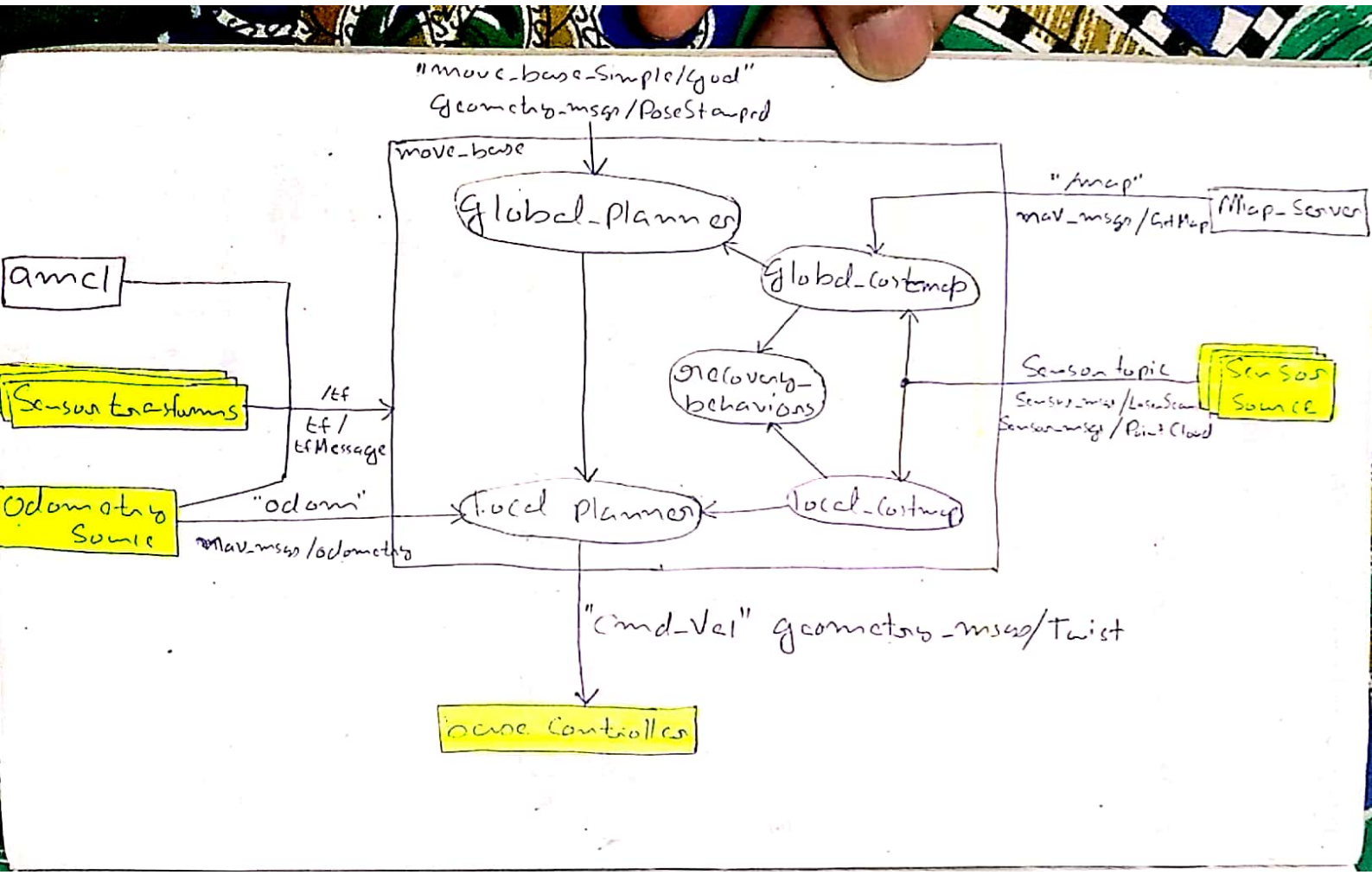
⊙ The optional nodes of the Navigation Stack are amcl and map server, which allow localization of the robot and help to save/load the robot map.

★ Working with Navigation package

★ Understanding the move_base node

(from package move_base)

⇒ The main function of this package is to move a robot from its current position to its goal position with the help of other Navigation nodes.



⇒ The move_base node basically is an implementation of SimpleActionServer which takes a goal pose with the message type (geometry_msgs/PoseStamped)

↳ move_base node subscribes the goal from a topic called move_base_simple/goal.

"When move_base node receives a goal pose

↓

It links to Components such as
- global_planner
- local_planner
- recovery_behavior
- global_costmap
- local_costmap

↓

generates the output which is the Command Velocity (geometry_msgs/Twist)

↓

and Send to the base Controller for moving the robot to achive the goal Pose"

=> Following is the list of all the packages which are linked by the move-base node:

global-planner

=> This package provides libraries and nodes for planning the optimum path from the current position of the robot to the goal position with respect to the robot map.

=> This package has implementation of path finding algorithms such as A*, Dijkstra, and so on for finding the shortest path from the current robot position to the goal position.

local-planner

=> It takes the odometry and sensor reading, and send an appropriate velocity command to the robot controller for completing a segment of the global path plan.

=> This package is the implementation of the trajectory rollout and dynamic window algorithms.

rotate_recovery

⇒ This package helps the robot to recover from a local obstacle by performing a 360° rotation.

Clear-Costmap-recovery

⇒ Also for recovering from a local obstacle by clearing the costmap by reverting the current costmap used by the Navigation stack to the static map.

Costmap-2D

⇒ The main use of this package is to map the robot environment.

↳ Robot can only plan a path with respect to a map.

⇒ In ROS, we create 2D or 3D occupancy grid maps, which is a representation of the environment in grid of cells.

↓

Each cell has a probability value which indicate whether the cell is occupied or not.

⇒ There are global cost maps for global navigation and local cost map for local navigation.

⇒ Other packages which are interfaced to the move_base node:

⊙ map_server:

⇒ Map Server package allows us to save and load the map generated by the Cost map 2D package.

⊙ AMCL

↳ Method to localize the robot in map

⇒ this approach uses particle filter to track the pose of the robot with respect to the map.

↳ With the help of probability theory.

⇒ In the ROS System, AMCL can only work with maps which were built using laser scans.

⊙ gmapping

⇒ Implementation of an algorithm called Fast SLAM which takes the laser scan data and Odometry to build a 2D Occupancy grid map.

★ Over all Working of Navigation Stack

1) Localizing on the map

→ The first step the robot is going to perform is localizing itself on the map.

(AMCL package)

2) Sending a goal and path planning

→ After getting the current position of the robot, we can send a goal position to the move_base node.

→ Move_base node will send this goal position to a global planner

→ Global planner will plan the path from the current robot position to the goal position.

→ global planner sends this path to the local planner, which executes each segment of the global plan.

→ Local planner gets the odometry and the sensor value from the move_base node and find a collision free local plan for the robot.

(local cost map)

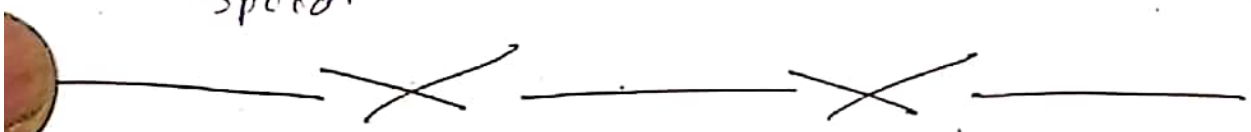
3) Collision recovery behavior

→ If the robot is stuck somewhere, the Navigation package will trigger the recovery behaviours modes.

4) Sending Command Velocity

→ The local planner generates Command Velocity in the form of a twist message, to the robot base Controller.

→ The robot base Controller Converts the twist message to the equivalent motor Speed.



★ Building a map using SLAM

⇒ slam_gmapping, is the implementation of SLAM which helps to create a 2D occupancy grid map from the laser scan data and the mobile robot pose.

Creating a launch file for gmapping

⇒ The main task while creating a launch file for the gmapping process is to set the parameters for the slam_gmapping node and the move_base node.