#Topics covered
1. Opening up worlds and robots in Gazebo.
2. Generating Map using Cartographer by moving the robot manually using the teleop node. You can also publish to the /cmd_vel topic to move it.
3. Saving maps using nav2_map_server.
4. Analyzing what is inside the maps. e.g. resolution, origin, mode, free_thresh, occupied_thresh, map file.
5. Making the robot navigate using turtlebot3_navigation2 package.
6. Creating 2D Pose estimate to correct the starting point of the robot in the map.
7. Putting navigation goals to move in the map.
8. Navigate the robot using way point follower. The continuous way point follower is not very robust.

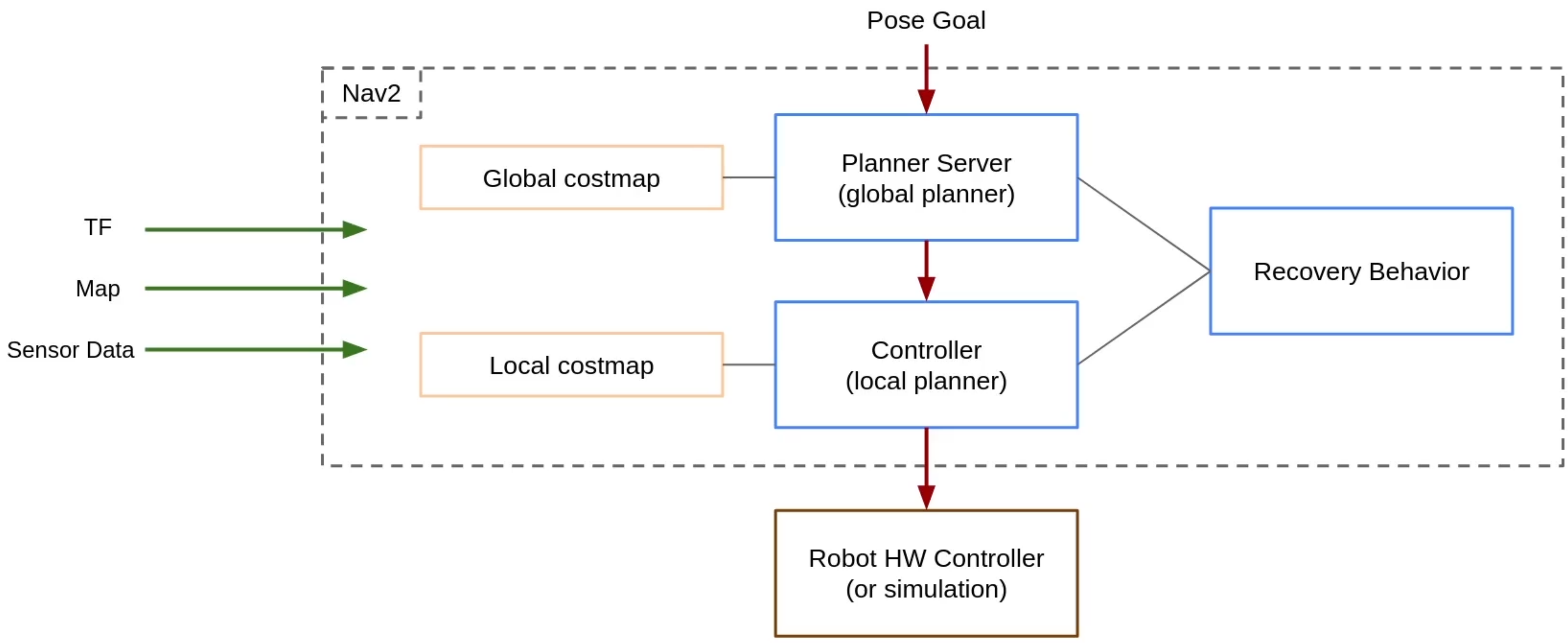
#More on Nav2
1. We have a Global Plannar and a Local Planner (Controller).
2. The Global Plannar computes the CostMap based on the entire map and the navigation goal. It also updates it slowly. Usually 1Hz or 2Hz. It will try to find the path with the least possible cumulative cost.
3. The CostMap is simply the cost for each pixel. The lower the cost, the better that pixel is. Obstacles always have the highest cost since you cannot bump into them.
4. The red pixels have high cost and the blue pixels have low cost. The wight pixels have the lowest cost.
5. The Controller plans locally and generates its own CostMap. It controlls the actual movement of the robot.

#Nav2 Parameters
1. To change parameters, open rqt.
2. For the global planner, reducing the inflation_radius lowers the cost around the obstacles. Adjust it to change the clearance around obstacles. The same method applies to Local Planner as well.
3. The lower the inflation_radius, the easier it is to compute the path, but you have a higher chance of bumping.
4/ The controller_server can be used to change the max velocity, acceleration, update rate etc.

#Recovery Behavior
1. When the plannar fails to generate a path, or the robot is unable to follow the path, then the recovery behavior is called.
2. It is usually a predefined motion. Like rotate a bit or move a bit etc.
3. The behavior server is called when there is an issue with the Global or the Local Plannar.

#TFs
1. To check the TF tree : ros2 run tf2_tools view_frames. It will generate a nice pdf.
2. map >> odom >> base_footprint.
3. map keeps track of the robot position for long time. e.g GPS or LIDAR
4. odom usually comes from IMU sensor or rotation of wheel etc.

#Nav2 Summary



#Load turtlebot3 in gazebo
ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py

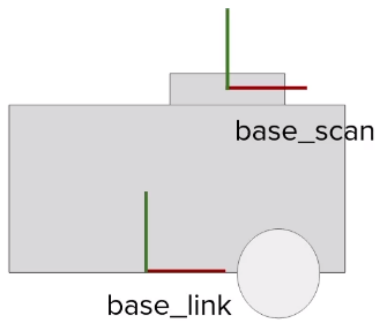
#Start the cartographer to generate the map
ros2 launch turtlebot3_cartographer cartographer.launch.py use_sim_time:=True

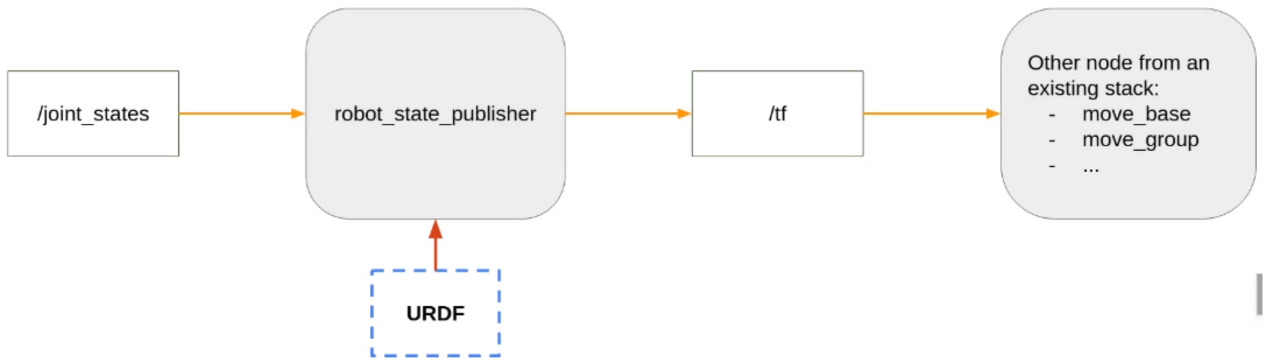
#save the map
ros2 run nav2_map_server map_saver_cli -f maps/my_map

#Now, do the navigation
ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py
ros2 launch turtlebot3_navigation2 navigation2.launch.py use_sim_time:=True map:=maps/my_map.yaml

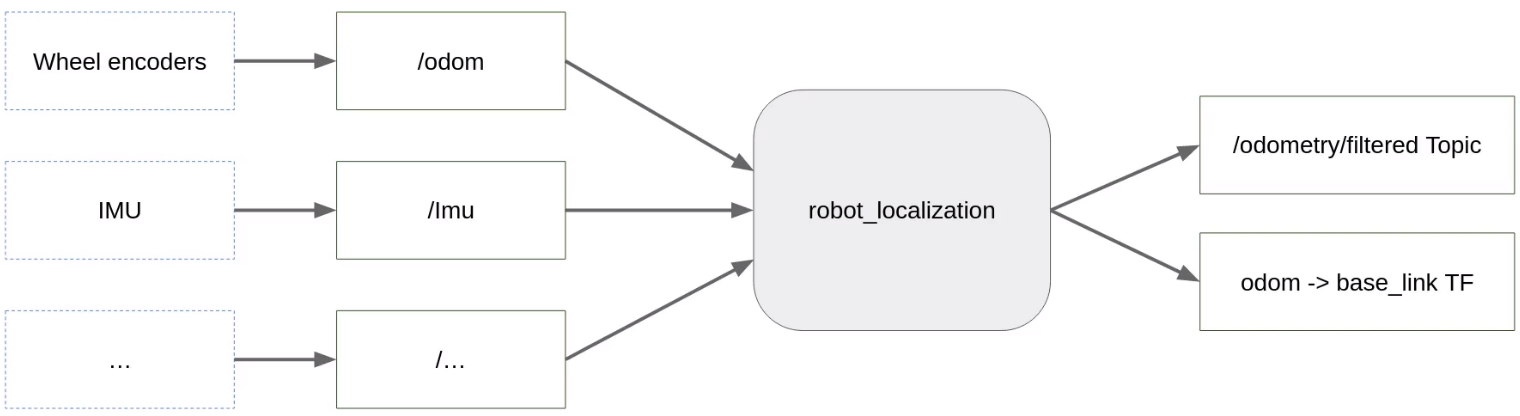
#Buiding custom worlds in Gazebo
1. You can use the Building editor to create walls, doors, windows, and stairs etc.
2. You can also import .png files and set the scale to create the world.
3. The Building editors saves the models in .sdf and .config files.
4. To save the entire world, make sure to put .world extension for the file name.
5. gazebo my_world.world (run to load the world)
6. Downloaded git repo for making our own simulation.
7. Maps can be improved with Gimp.

#Adapting a Custom Robot for Nav2
1. Important transforms needed:
   a. map --> odom
   b. odom --> base_link
   c. base_link --> base_scan
2. The robot_state_publisher node converts the URDF to /tf while also
   using the /joint_states.





#Odometry, Sensors, and Controller
1. Odometry: Localizing the location of the robot from its starting position using its own motion.
2. For example, wheel encoders, IMU etc.
3. Read the encoder velocity and integrate to get position. Then publish is on /odom topic.
4. Also you have to publish the TF from odom --> base_link using the /tf topic.
5. You can also use diff_drive_controller of ros2_control framework. This framework has several repos.
6. If you have odometry data from multiple sensors, then use robot_localization package.



7. For the LIDAR scanner, publish the data on /scan topic with interface sensor_msgs/msg/LaserScan.
8. For a 2D camera, publish on /camera/raw_image with interface sensor_msgs/msg/Image.
9. Make sure to add the camera_link in the URDF file.

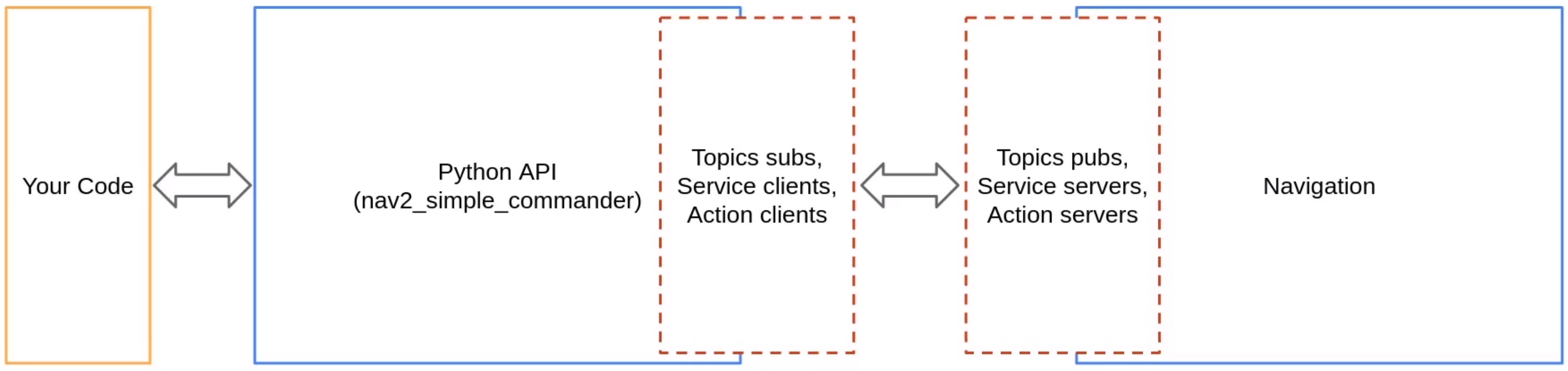
#Robot Hardware Controller
1. Custom. Convert /cmd_vel to actual motor commands etc.
2. The robot controller also does the odometry by reading the encoders and IMU etc.
3. diff_drive_controller can also be used instead of developing your custom controller.

#slam_toolbox
1. Start the real prototype of the robot or a simulation in gazebo.
2. Use ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py. (or implment your own version)
3. This should publish the important topics: camera/camera_info, /camera/image_raw,/clock
/cmd_vel, /imu, /joint_states, /odom, /parameter_events, /performance_metrics, /robot_description,
/rosout, /scan, /tf, /tf_static
4. Now start the nav2 generic functionality irrsepective of the robot model.
5. This can be done using: ros2 launch nav2_bringup navigation.launch.py use_sim_time:=True
6. Now start slam_tool box: ros2 launch slam_toolbox online_async_launch.py use_sime_time:=True
7. Start rviz2 to visualize etc.
8. Maps can be saved with ros2 run nav2_map_server map_saver_cli -f <filepath>

9. Now you have saved the map. Next step is to navigate the robot.
10. ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py (ROBOT SIMULATION)
11. ros2 launch nav2_bringup bringup_launch.py use_sim_time:=True map:=<filepath>
(DIFFERENT THAN WHEN BUILDING MAP, NOW WE ALREADY HAVE MAP)
12. Launch rviz2, configure, and start navigation.

#How to put everything into a single LAUNCH FILE & Adapt Parameters for the ROBOT
1. Important params: base_frame_id, global_frame_id, odom_frame, robot_model_type, max_vel_x etc.

#Interact programmatically with Nav
1. nav2_simple_commander
2. map frame is the anchor frame for everything. (0,0,0)
3. initial pose just uses a topic while navigation goal uses an action with request, response, and feedback.



4. Way point following can also be done using nav2_simple_commander

What to learn next?
- ros2_control
- Moveit 2 for Robotic Arms
- Life Cycle nodes, actions, components etc.