ros_course_part2

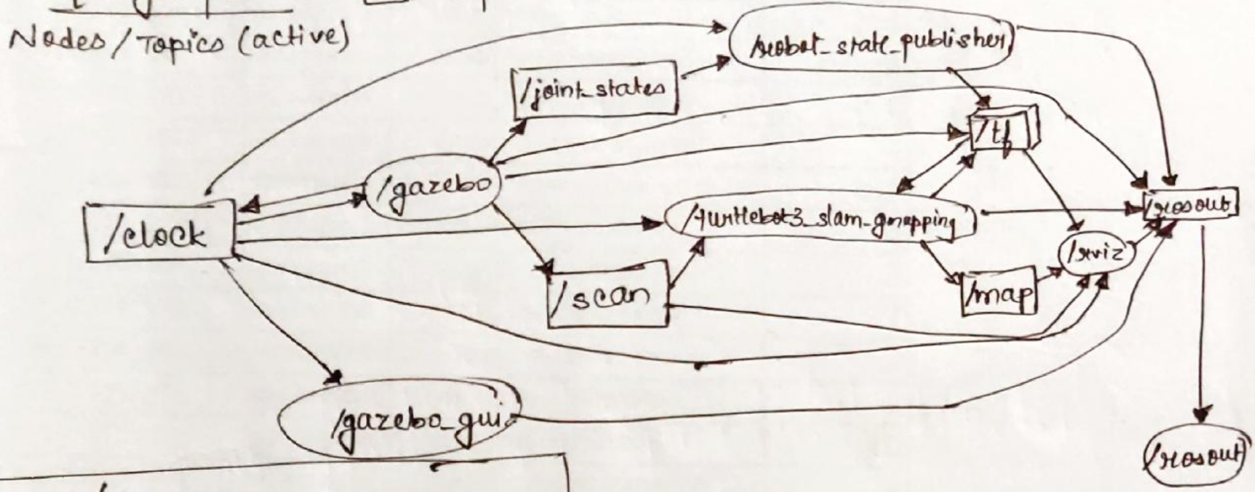
ANIS KOUBAA
(Udemy)

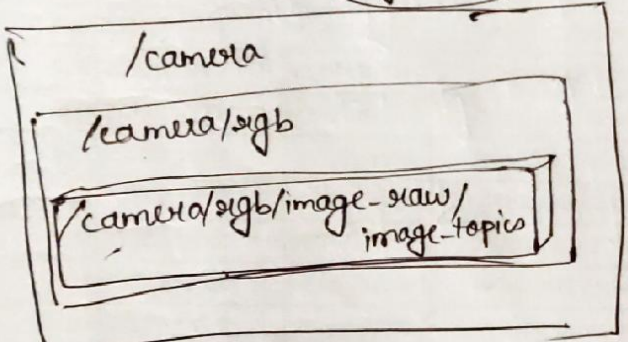
01) quaternion
02) tf_tutorials
03) map_navigation

ROS For Beginners II: Localization, Navigation, SLAM

rqt_graph   □ : Topic ○ : Node

Nodes / Topics (active)





- Find location of robot : rostopic echo /odom
  rostopic echo /amcl_pose

[map frame is the global frame that refers to the position of the robot]

- static obstacle-free path [we call global path planning]
- then executes the path using local path planner which also avoids dynamic obstacles ~~this course~~

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

Quarternion $s \langle v_1 | v_2 | v_3 \rangle$

$$q = s + v$$
$$q = s + v_1 i + v_2 j + v_3 k$$
$$i^2 = j^2 = k^2 = ijk = -1$$

$$\begin{cases} q_0 = q_w = s \\ q_1 = q_x = v_1 \\ q_2 = q_y = v_2 \\ q_3 = q_z = v_3 \end{cases}$$

In ROS: notation

(x, y, z, w)

$x \to q_x$, $y \to q_y$, $z \to q_z$, $w \to$ for s

- Euler Rotation Sequence
- Cardan Rotation Sequence
- Euler's Rotation Theorem
- Rodrigues Formula

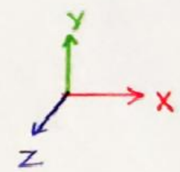
TF

- tf : It allows us to find the pose of any object in any frame using transformations.
- A robot is a collection of frames attached to its different joints.

[(red) : X-axis
(green) : Y-axis
(blue) : Z-axis]

- URDF: Language for the Description of Frames and Transformation in a Robot Model

- tf package nodes
  - view_frames
  - tf_monitor
  - tf_echo
  - roswtf
  - static_transform_publisher



- view_frames

  alias tf='cd /var/tmp && $\boxed{\text{rosrun tf view\_frames}}$ && evince frames.pdf &'

- Methods provided by tf package consider angle in radians

In code
- tf.transformations.quaternion_from_euler(roll, pitch, yaw)
- " . " . euler_from_quaternion (q)

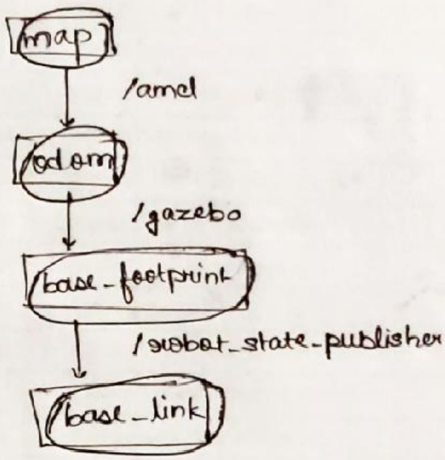
topic01_quaternion: scripts

✓ tf_rotation_conversions.py
(The 2 tf functions mentioned prev page)

✓ tf_orientation_tb3_robot.py
(Takes pose from /odom
↓
orientation
↓
quaternion → euler
↓
prints yaw)

In view_frames.pdf,

/robot_state_publisher : node that publishes transformations based on URDF description



- tf_echo

  $ rosrun tf tf_echo odom camera_rgb_frame 2

  echo rate ↑ (optional)

  Gives homogeneous

- tf_monitor

  $ rosrun tf tf_monitor odom base_footprint

  Information about transformation such as publisher node, the average delay and the maximum delay.

  Also the statistics of all broadcasters.

  $ rosrun tf tf_monitor

- static_transform_publisher

Y R → P ?

Tab 1 : $ roscore

Tab 2 : $ rosrun tf static_transform_publisher 1 2 3 0.1 0.2 0.3 frame_a frame_b 10

translation vector: 1 2 3

Y P R: 0.1 0.2 0.3

frame_a → parent

frame_b → child

10 → broadcast frequency (10 times/s)

topic02_tf_tutorials: ~~launch~~

- static_transform_publisher.launch
  - <node pkg="tf" type="static_transform_publisher" name="frame_a_to_frame_b" args="1 2 3 0.1 0.2 0.3 frame_a frame_b 10" />
- frame_a_to_frame_b_broadcaster.cpp
- ✓ " " " .py → tf.TransformBroadcaster() → .sendTransform(...)
- ✓ " " listener.py → tf.TransformListener()
  - .waitForTransform(..) → 4 seconds max {in code}
  - .lookupTransform(..)

Map-Based Navigation

- Map based navigation
- Reactive navigation

Map based navigation:

- Localization
- Mapping
- Motion/path planning
- SLAM
- Sensor Fusion

How does it work?

Three main packages of the navigation stack:-

- move_base : move to goal pose
- gmapping : creates maps ~~using laser scan data~~
- amcl : localization (using existing map)

- Occupancy Grid Map

Cell:
- Unknown (grey)
- Free (white)
- Occupied (black)

SLAM approaches in ROS :-

- gmapping
- cartographer
- hector_slam

- Quality of the sensors (odometry + laser scanner) affects the quality of the generated map

- map_server : package for publishing and manipulating maps
  map_saver : node inside map_server

tb3_map.pgm → greyscale image

tb3_map.yaml → occupied_thres, free_thres

- These 3 frames : base_frame, odom and map are mandatory for any robot navigation mission.

  "delta" → resolution of the map

  "0.05" means 0.05m or 5cm/px

- 2D Pose Estimate → 2D Nav Goal

  Motion Planners → Global Path Planner (static obstacle-free path)

  Motion Planners → Local Path Planner (avoid dynamic obstacles)

- Recovery Behaviour : initiated when the local planner finds obstacles while following the planned global path

  clearing and marking processes

- Navigation launch file include:
  1. turtlebot3_remote.launch → robot model, robot_state_publisher node
  2. map_server (takes yaml file as argument)
  3. amcl.launch [Adaptive Monte Carlo Localization]
     ↓
     uses particle filter to track pose of robot against a known map
  4. move_base.launch → Most important : represents the navigation stack
     ↓
     More on this?
  5. ~~[redacted]~~ rviz

Topic03_map_navigation : → navigate_goal.py, navigate_goal.cpp

Writing a ROS Node for Robot Navigation

Q.1] How to determine coordinates of goal location?

Graphical
(counting grid squares)

(i) $ rostopic echo initialpose
[In Rviz, 2D Pose Estimate at goal location]

(ii) $ rostopic echo amcl_pose

Method:
move_to_goal (x_goal, y_goal) } In the script "navigate.py"

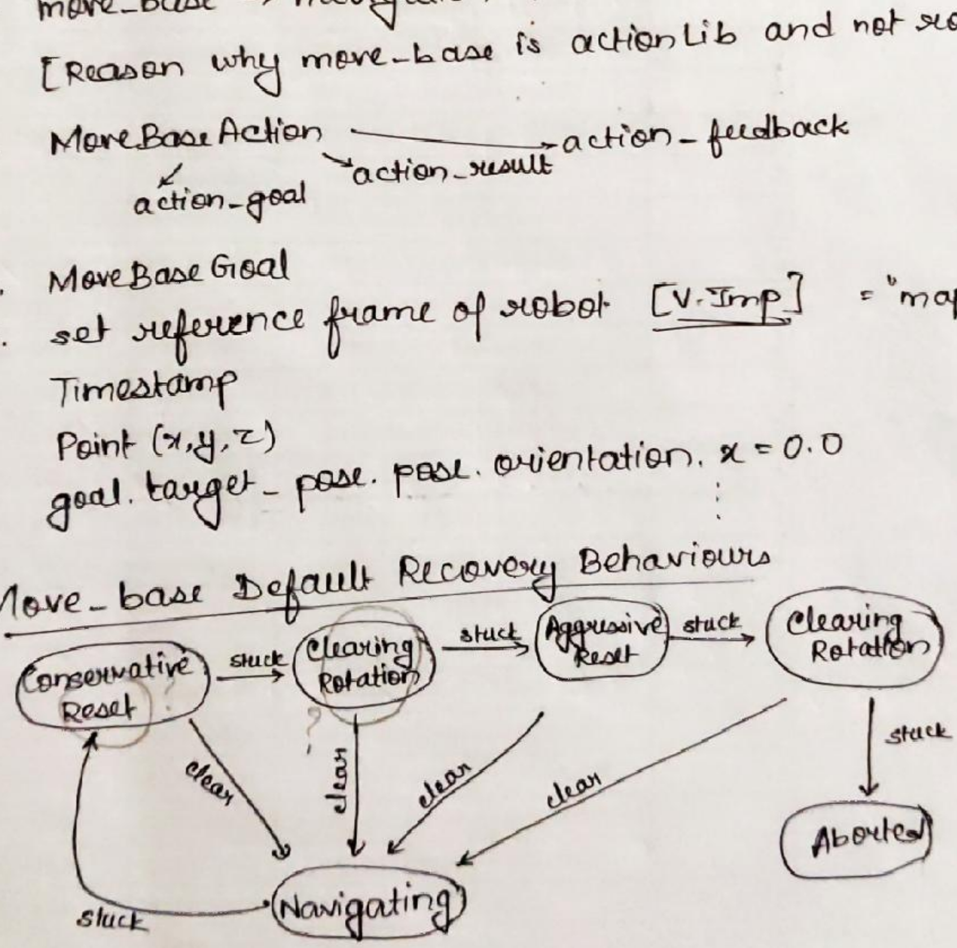
1. actionlib client
   ↳ defines a client - server application where tasks are pre-emptable
   ↳ means they can be interrupted
   ↳ communication is fully asynchronous

2. "move_base" → navigation stack server
   [Reason why move_base is actionLib and not rosservice]

3. MoveBaseAction → action_feedback, action_result, action_goal

4. MoveBaseGoal
5. set reference frame of robot [V.Imp] = "map"
   Timestamp
   Point (x, y, z)
   goal.target_pose.pose.orientation.x = 0.0
   ⋮

• Move_base Default Recovery Behaviours



- Robot setup to support the ROS Navigation Stack

(Would be more relevant later)

Provide node
- global_planner
- global_costmap
- recovery behaviours
- local_planner
- local_costmap

Optional provided node
- amcl
- map-server

Platform specific node
- sensor transforms
- odometry source
- sensor sources

{Parts that need to be set up}

LAST SECTION

Configuration & Tuning of the Navigation Stack Parameters

{(Reference): ROS Navigation Tuning Guide ~ Kaiyu Zheng}

- Tuning Max/Min Velocities & Accelerations
  - $ roscd turtlebot3_navigation
  - $ cd param
  - $ more dwa_local_planner_params_waffle

- Global Planner Parameter Tuning

  nav core :: BaseGlobalPlanner
  - initialize
  - makePlan
  - makePlan (with cost)
  - destructor

  [Another thread here: "Writing a Global Path Planner as a Plugin in ROS"]

  3 built-in global planners
  1. carrot_planner
  2. navfn
     - Djikstra's
  3. global_planner
     - can use grid path
     - support of A*

  ?

- Local Path Planner Overview

  nav core :: BaseLocalPlanner
  - initialize
  - isGoalReached
  - computeVelocityCommands
  - setPlan
  - destructor

- DWA Algorithm
[Dynamic Window Approach] ~ Dieter Fox

· $\begin{bmatrix} v \\ \omega \end{bmatrix}$ · highest-scoring trajectory

(random values
↓
then forward simulations)

- Tuning the Simulation Time of the DWA Algorithm
time allowed for the robot
to move with the sampled velocities

- DWA Trajectory Scoring

* cost = path_distance_bias × (distance to path from the endpoint of the trajectory)

{local trajectories closer to global path}

+ goal_distance_bias × (distance to local goal from the endpoint of the trajectory)

{trajectory closer to the goal, may be far from global path}

+ occdist_scale × (maximum obstacle cost along the trajectory in obstacle cost (0-254))

(Occlusiondistance)
{help selecting trajectories far from obstacles}

- Tuning the DWA Trajectory Scores ⟶ in the yaml file
$ rosrun rqt_reconfigure rqt_reconfigure *
↓
allows to change parameter values dynamically without changing configuration files

1) init_node : rospy.init_node('node_name', anonymous=True)
2) Subscriber — 4) callback (msg)
3) spin : rospy.spin()

rospy.Subscriber('topic_name', {MessageType}, callbackFunction)