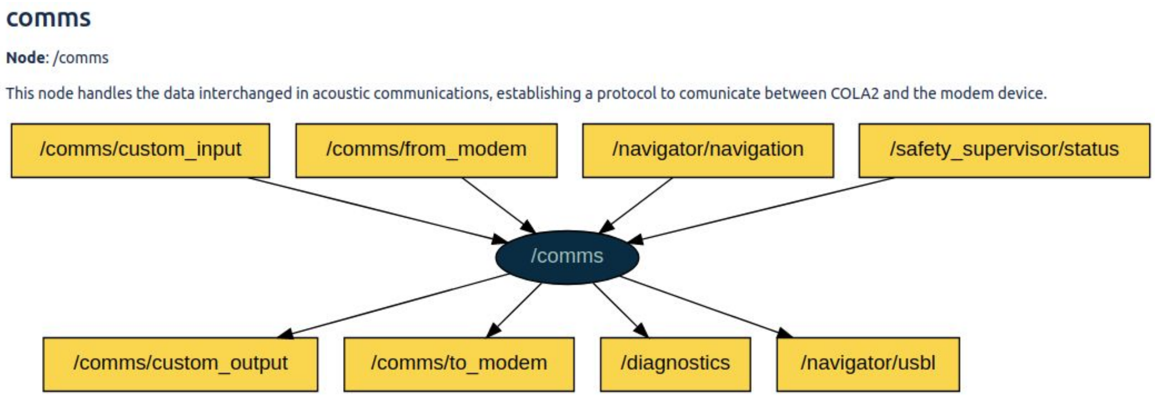
ROS1 Robots systems

1. Cola2

- End-to-end implemented with ROS1
- Uses standard ROS TF for frame and Rviz, Rqt (Ros based tools) for control and visualization
- Main modules of cola2
    - Communication
        - ROS package with nodes to perform acoustic communication

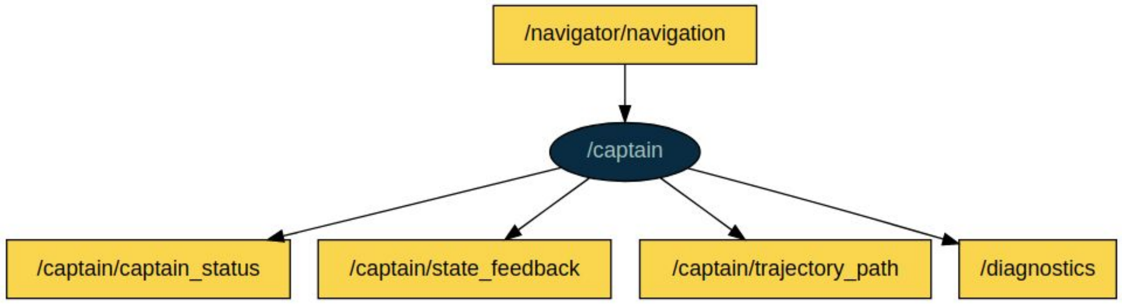


comms

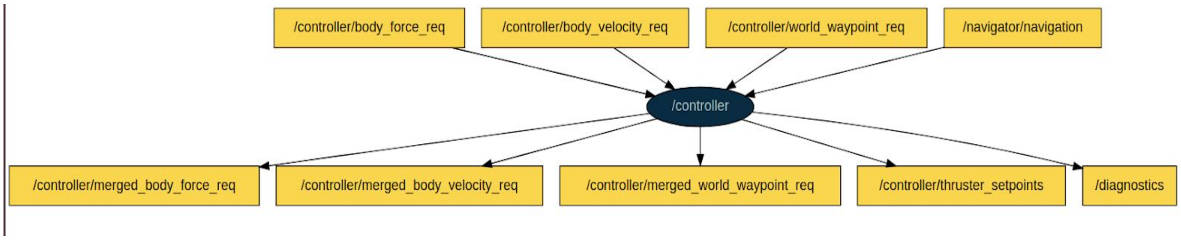
Node: /comms

This node handles the data interchanged in acoustic communications, establishing a protocol to comunicate between COLA2 and the modem device.

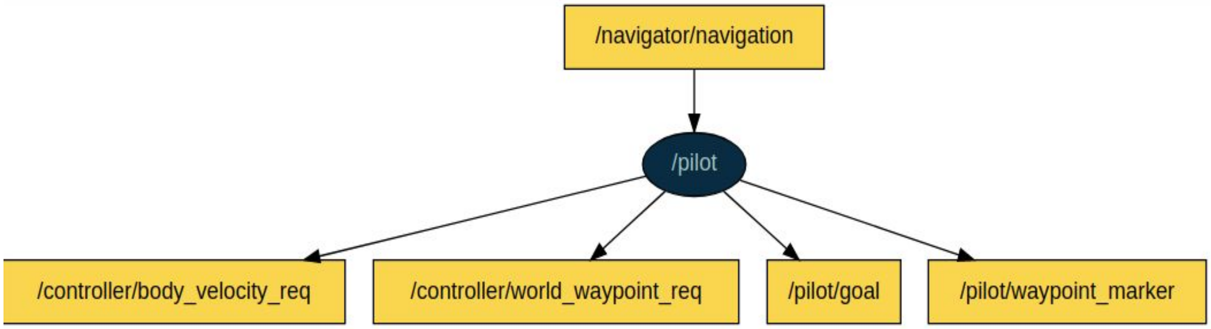
-
    - Control
        - ROS package with nodes to control the COLA2-based AUVs.
        - Captain :automatic control actions including: goto a waypoint, keep position



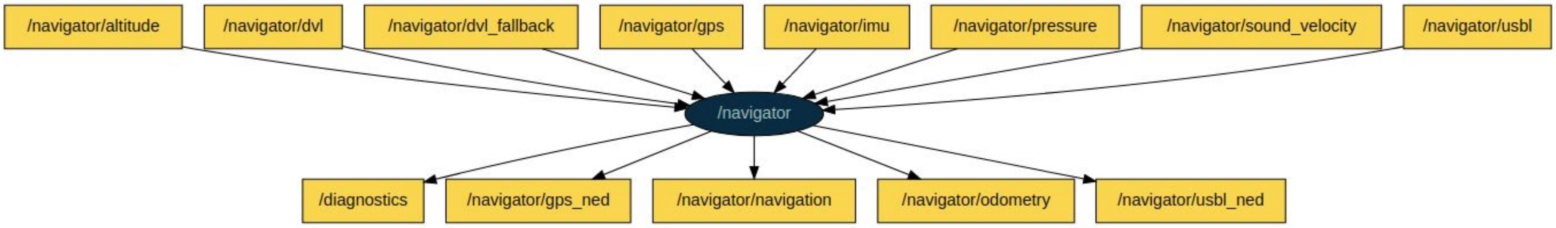
-
    -
        -
        - Controller: pose and velocity low level controllers and the thruster allocator.


/controller/body_force_req
/controller/body_velocity_req
/controller/world_waypoint_req
/navigator/navigation
/controller
/controller/merged_body_force_req
/controller/merged_body_velocity_req
/controller/merged_world_waypoint_req
/controller/thruster_setpoints
/diagnostics

    - Joystick/Keyboard teleoperation
    - Pilot: Provides two action libs to be used by the captain node and upon its requests publishes position and velocity setpoints to the position and velocity controller



- Log : nodes related to the logging/storing cola2 parameters and data, includes rosbags
- Nav: nodes to estimate the position of the AUV
    - Navigator: merges data from different navigation sensors of an AUV to estimate the robot position and velocity using an **Extended Kalman Filter**



- Safety: ROS package with nodes to perform safety checkings to the COLA2 architecture.
    - recovery_actions

        - safe_depth_altitude
        - safety_supervisor
        - set_zero_velocity
        - vehicle_status_parser
        - virtual_cage
        - Watchdog
    - Sim : for simulating instead of running it on real hardware

2. Husky

- Ground Vehicle
- ROS1 based
- Uses Extended and Unscented Kalman filter based robot localization ros package (sensors data acquired: IMU, GPS, Odom)
- Besides, it uses navsat_transform ros package to convert lat-long data into robot’s odometry coordinates
- It uses laser scan for SLAM creating a map
- Husky uses move_base (from ROS1 Navigation stack) to navigate to the goals while avoiding obstacles
- From navigation stack it uses AMCL (adaptive monte carlo localization); using laser scan as input along with move_base for autonomous planning

3. Kingfisher

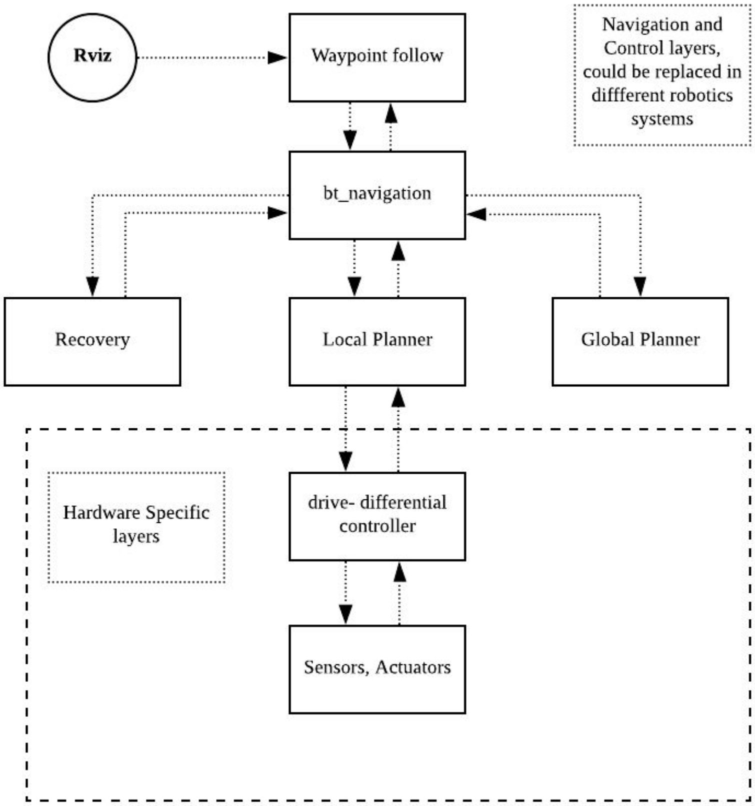
- ROS1 based, but quite deprecated software
- Currently it has:
    - Robot Description files
    - Robot ROS messages
    - Joystick teleoperation facility

ROS2 Robots Systems

Turtlebot3

- 2-D pose estimation (rviz)
- 2-D navigation goal

- Turtlebot3 follows path and arrives at destination
- If unexpected obstacle blocks the path, Robot can detect them to avoid
- Turtlebot3 uses:
  - Robot’s encoder
  - IMU sensor
  - Distance sensor
- Saved map-> contains field info -> use in node

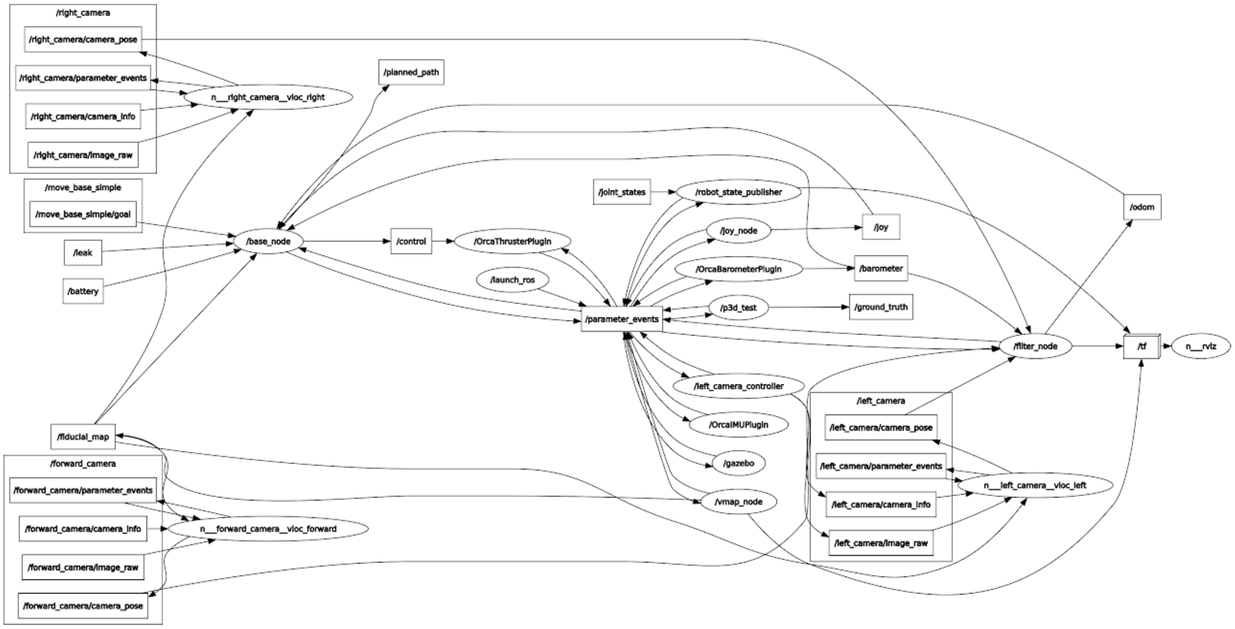

Rviz
Waypoint follow
Navigation and Control layers, could be replaced in diffferent robotics systems
bt_navigation
Recovery
Local Planner
Global Planner
Hardware Specific layers
drive- differential controller
Sensors, Actuators

Turtlebot3 system architecture for ROS2

Orca2

- Underwater robot, actually it’s a modified version of BlueROV
- Runs on ROS1 as well as ROS2
- Has its own navigation stack with some similarity to ROS
- Has a base nodes which has different callbacks linked to it

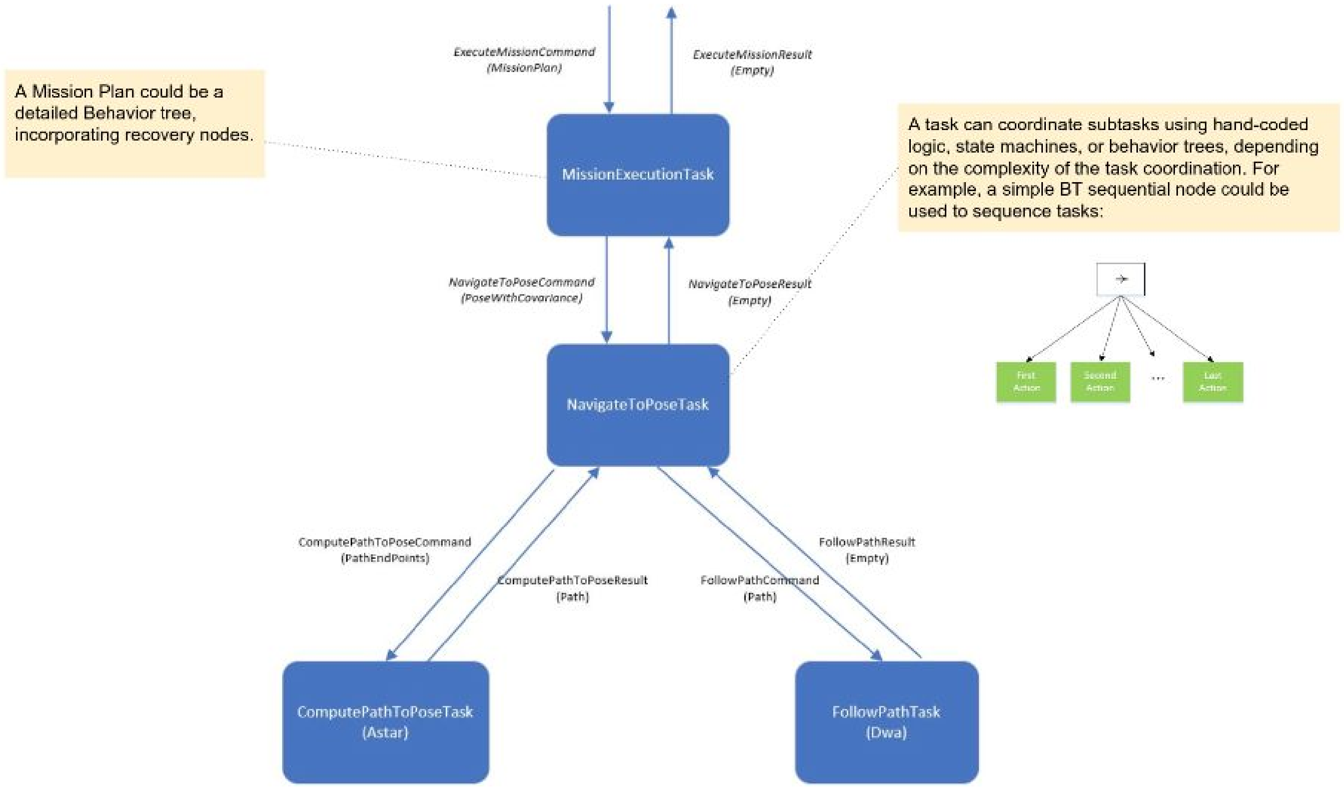
- Most of the messages are ros custom messages, few they have created their own
- Controllers mainly include (they are PID based):
  - Simple controller
  - Deadzone controller
  - Jerk controller
  - Depth controller
- Orca2 code has been more based on data obtained from visual slam and it uses aruco markers for navigation



<u>ROS2 Navigation Stack: an overview</u>

- Navigation 2: send robot to a designated destination in a given environment
- Uses data created in ROS2 SLAM
- Control over params: max-min vel, rot. vel,accel, tolerance
- Global Planner- global plan
  - Requires a map of the environment to calculate the best route

- Local Planner (DWB planner used)
  - Transform global path -> suitable waypoints
  - Creates new waypoints -> dynamic obstacles; vehicle constraints
- Use of navigation stack on an arbitrary robot
  - ROS required
  - TF transform tree (tf-maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc between any two coordinate frames at any desired point in time.)
  - Sensor data using correct ROS message type Needs to be configured for shapes and dynamics of a robot to perform at a high level
  - Planar laser mounted somewhere on the mobile base (map building and localization); docs.



A Mission Plan could be a detailed Behavior tree, incorporating recovery nodes.

A task can coordinate subtasks using hand-coded logic, state machines, or behavior trees, depending on the complexity of the task coordination. For example, a simple BT sequential node could be used to sequence tasks:

- **As explored in above existing systems, they are using a modified version of Ros navigation stack to customize their requirements**
- **Ros navigation stack is more based on laser scan input, EKF, UKF based packages provided by ROS facilitate robot localization using GPS, IMU sensors as observed.**