Implementation: Roles

Abstract: This document describes the implementation of roles in the ARC_ROS project. Roles in this instance describe an expected behavioral pattern or set of expected actions or tasks a robot operates under to participate in a team structure to perform USAR operations.

The following figure describes the file hierarchy of initializing a robot's behavior set without a role.

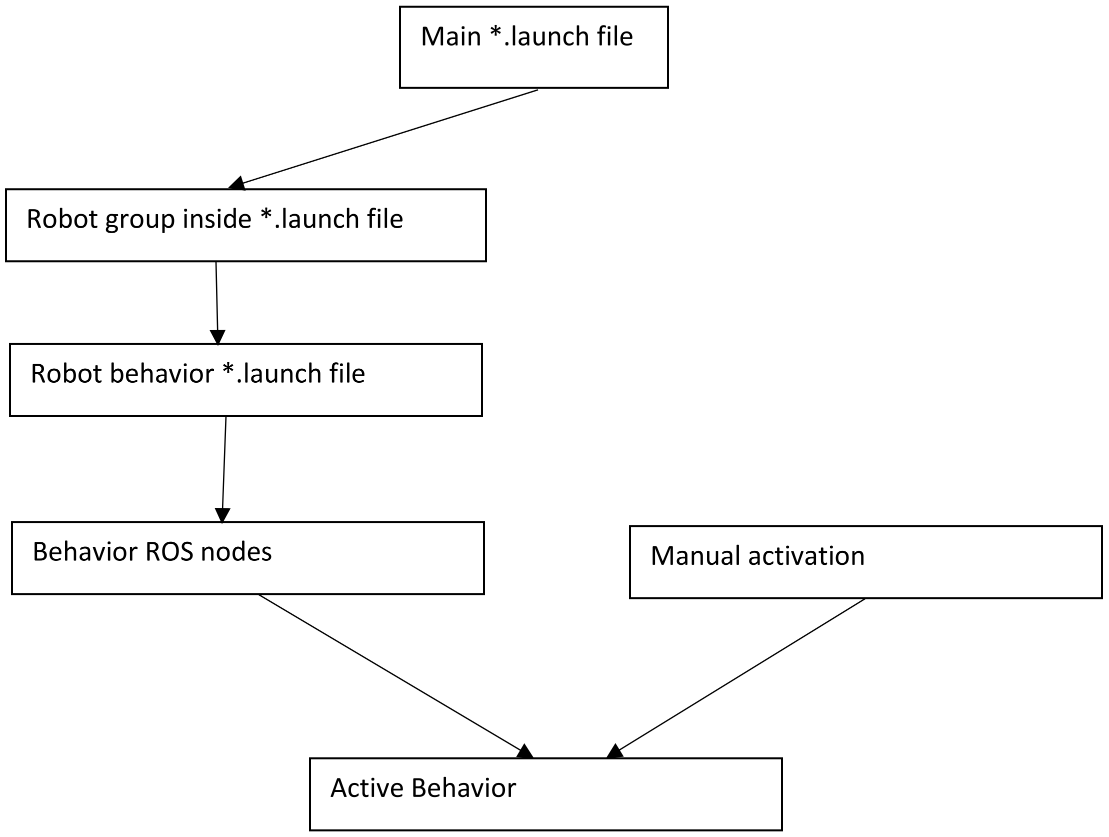


Fig 1. A representation of the launch file hierarchy to active behavior

Launching a simulation starts with a main launch file in the arc_control package. In the main launch file, we designate which secondary launch files we want for each of our declared bots. Based on the robot type defined in the main *.launch file we nest the appropriate behavior *.launch files on execution.

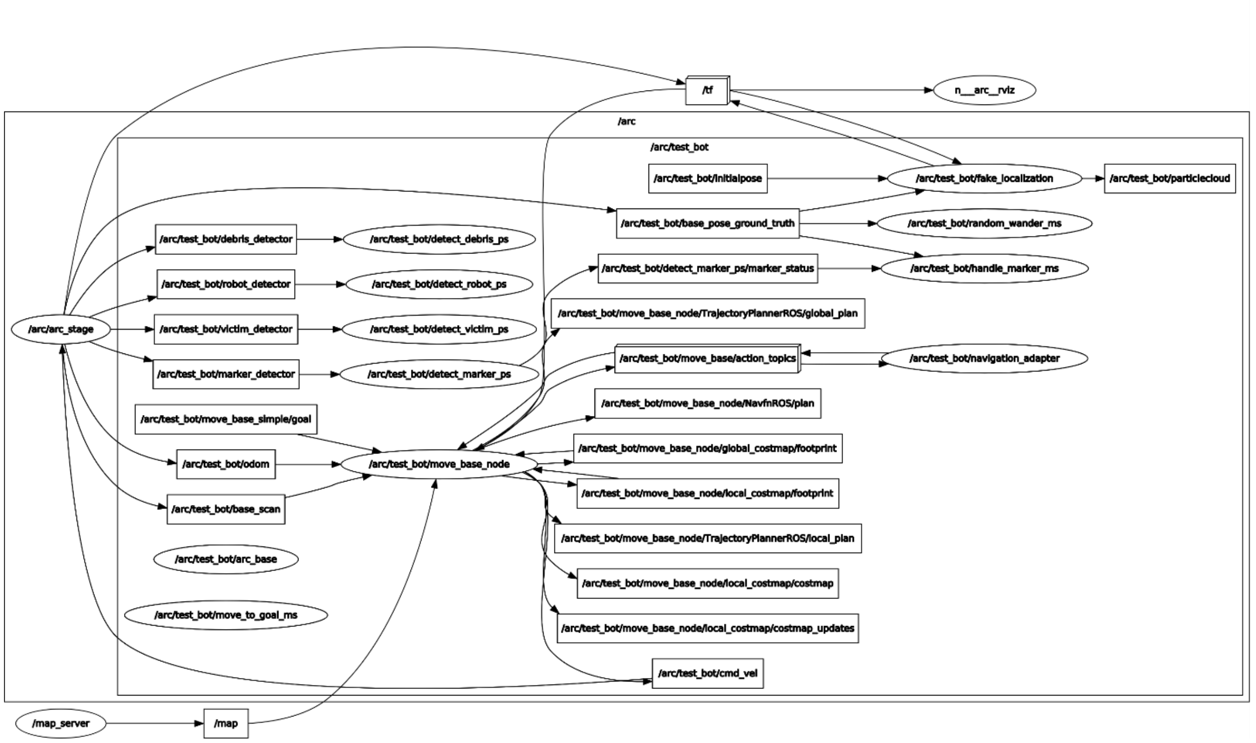

/tf
n__arc__rviz
/arc
/arc/test_bot
/arc/test_bot/initialpose
/arc/test_bot/fake_localization
/arc/test_bot/particlecloud
/arc/test_bot/base_pose_ground_truth
/arc/test_bot/random_wander_ms
/arc/test_bot/debris_detector
/arc/test_bot/detect_debris_ps
/arc/test_bot/detect_marker_ps/marker_status
/arc/test_bot/handle_marker_ms
/arc/test_bot/robot_detector
/arc/test_bot/detect_robot_ps
/arc/test_bot/move_base_node/TrajectoryPlannerROS/global_plan
/arc/arc_stage
/arc/test_bot/victim_detector
/arc/test_bot/detect_victim_ps
/arc/test_bot/move_base/action_topics
/arc/test_bot/navigation_adapter
/arc/test_bot/marker_detector
/arc/test_bot/detect_marker_ps
/arc/test_bot/move_base_node/NavfnROS/plan
/arc/test_bot/move_base_simple/goal
/arc/test_bot/move_base_node/global_costmap/footprint
/arc/test_bot/odom
/arc/test_bot/move_base_node
/arc/test_bot/move_base_node/local_costmap/footprint
/arc/test_bot/base_scan
/arc/test_bot/move_base_node/TrajectoryPlannerROS/local_plan
/arc/test_bot/arc_base
/arc/test_bot/move_base_node/local_costmap/costmap
/arc/test_bot/move_to_goal_ms
/arc/test_bot/move_base_node/local_costmap/costmap_updates
/arc/test_bot/cmd_vel
/map_server
/map

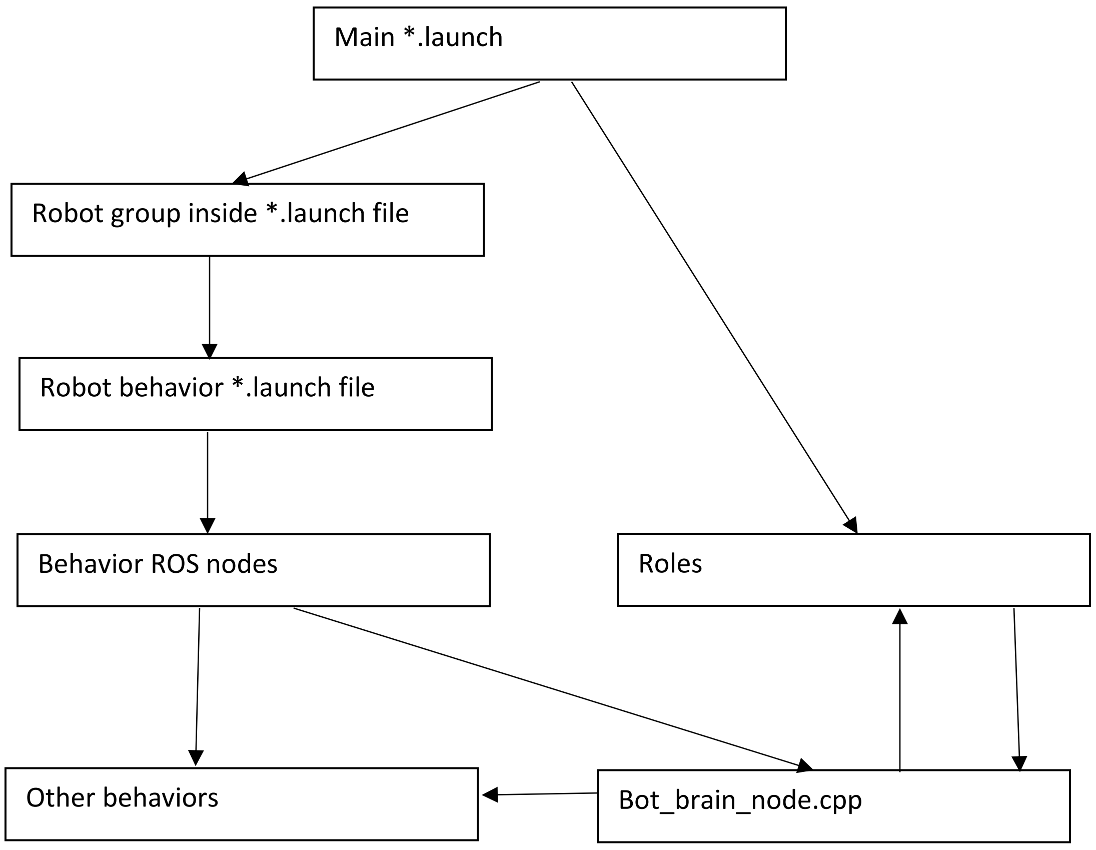
Fig 2. An example graph of the nodes launched from a main *.launch file where one bot labled test_bot is nested.

The robots are defined using separate launch files that are called in the main file. These are the robot behavior launch files located in the arc_control package, in the *robot* directory. These behaviors launch files are used to define the behaviors of each robot type and executes the appropriate behavior nodes for that bot. Each different bot has its own launch file with different defined expected behaviors.

The behavior nodes are the individual ROS nodes that describe and implement the individual behaviors that a robot could exhibit and can handle the distribution of tasks. These behaviors are contained in the arc_behaviour package.

However, the actual function of activating a set behavior of a bot is done manually by imputing a service call. A user can manually input a toggle service call to a particular robot, particular behavior effectively having to directly control the decisions the bot should make by manipulating its behavior.

With Roles this hierarchy changes:



In the main launch file, 4 new nodes are initialized above the bots, these are their roles, which are ROS nodes which house the information and behavior of the different role's types. These roles nodes are housed in a new package called Arc_roles.

The system retains separate behavior launch files at the moment, to account for the physical differences of each robot but, now every file executes every behavior universally. In other words, any bot can perform every behavior, uncoupling the two. The bot specific launch file also, initializes an additional behavior called bot_brain.

This is an additional behavior is implemented to control the other behavior based on the role designated. When the bot brain behavior decides to choose the best role for its bot. Bot_brain sends a service message to each role node and is returned a suitability value. Bot_brain then uses that suitability to enable and disable the appropriate behaviors of that bot's role. This effectively removes the need for manual control and the implementation of roles enable the decoupling of bot type to bot behavior.

Change Log:

- New Package added Called Arc_Roles
  - Contains the role object
  - Contains the executable ROS nodes for each role type
- New Role Object
  - Contains the core functions relavent to any given role.
- New Role Ros Nodes
  - Implement the new Role Object and creates instances of the different role types.
- New Service Msg QuerryRole.srv in arc_msgs
  - Used by bot_brain to query a role node, asking for that bots appiptitude.
  - Structure; Request.bot_type //given some bot type
    Response.app //return some aptitude for that bot type to be that role

- New Testing file test_simple_role.launch in the test folder of the arc_control package
  - A main *.launch file currently being used to test my implementation of roles.
- New behavioral *.launch files for each bot type
  - Contains nodes and parameters to initialize each type of robot in the stage.
- New behavior file called bot_brains
  - Contains the core functions of querying role nodes and assigning the bots roles.

Improvements to implement in the last month

- Suitability value needs to be properly implemented, it is currently controlled by a Magic number and not doing the proper calculation.
- Additional test files need to be implemented to cover the breadth of the implemented features.
- Role_check
  - Currently the roles of a bot are only defined once it starts up and doesn't have the functionality to switch roles properly, thus role_check is a natural continuation of roles.
- Task Management
  - Now that roles have been defined, I can start assigning tasks based on the role of a robot.
- Task Discovery
  - Under task Management, certain roles implement the discovery of tasks
- Role-Based Task Assignment
  - Certain roles can assign bots with certain roles different tasks.