COSC69.13/269, Spring Term 2021, PA3 - Coordinaton Mechanisms in ROS, Anmol Chachra

My ros package multirobot_shapes implements a ros node that has one leader and remaining followers. By default there is 1 leader and 2 followers. This package sets up communication between leader and follower and make everyone go to an n shaped regular polygon vertices, where n is the same as number of robots in the world, i.e. 3 robots will form a triangle. The specs for triangle can be controlled in leader node.

Method description

Program is broadly controlled by the following 3 parameters (Note: for now you have to set these values directly in the node file, this will be changed in later releases): 'N' to define the number of robots. Be careful while modifying this parameter, as you will need to spawn the same number of robots in the launch file of gazebo 'launch/multirobot_shapes_gazebo.launch'; 'SIDE_LEN' to define the size of side of the polygon, and 'INIT_POSE' to define the initial position of the desired polygon.

Package has 3 nodes - leader, follower and static_multirobot_shapes_tf_broadcaster. The launch file sets up the broadcaster for each spawned robot adding it to the world reference frame. See 'frame.pdf' for the graph that demonstrates connections between various nodes after the broadcaster finishes setting up the relative frames. Then it proceeds to run the leader node for robot_0 and follower node for robot_1 and robot_2.

The leader node instantiate 'Leader' class object, calls the 'set_waypoints' method - a method to automatically calculate the target waypoints given aforementioned parameters, and the publish a boolean message on 'init_coordinate' topic. The purpose for this is to let the followers know that the leader is now ready to accept collaboration for the task. Then it goes ahead to start 'registration_service' and call rospy.spin() to stay awake indefinitely.

The follower node instantiate 'Follower' class object, it then indefinitely checks if the subscriber to 'init_coordinate' topic is set in the object, i.e. the value is True. If the value is True, i.e. the leader is now accepting collaboration, it exits the while loop and go ahead to register itself with the leader using service client and sending it's namespace value, aka robot_name.

Now, when the followers successfully registers themselves with the leader, the leader get the 'odom' world reference frame for those followers using a 'TransformListener'. Once it has the expected number of followers registered, i.e. N-1, where N is the number of robots in the world, it spawns a thread that runs 'assign_waypoints' method to greedily assign the waypoints - for every waypoint, it selects the closest robot and assigns it that robot.

Then it spawns a thread and make leader go to its assigned waypoint. While

that thread is running, it spawn 2 other threads for each follower that runs 'send_goal' method. 'send_goal' method sets up the action client, calculate goals (dist - distance to the desired point, angle - rotation to be made before moving the desired distance) and send them via that client to the follower robots.

Each 'Follower' class object has its own action server to receive the action from the leader. Once the follower gets the goal, it simply rotates for 'goal.angle' and travel distance for 'goal.distance' using 'move_forward' and 'rotate_in_place' methods

Cool Feature: Choosing threads to never stop the registration service or stop the leader and follower moving towards the assigned waypoints, allowing parallel execution of tasks.

Cool Feature: 'leader' and 'follower' nodes use laser scan to identify close obstacles. If say 1 robot identifies that the other robot is in it's path and is close, it stops until its path is cleared, then it continues moving towards its goals. This is achieved by adding the lost duration when the robot was not moving to the start_time of that robot, thus keeping the desired movement duration.

Evaluation

Program was evaluated for 3 robots in gazebo where leader is 'robot_0' and other 2 are followers with namespace 'robot_1' and 'robot_2'. There positions are (-2, 1, 0), (2, 1, 0) and (0, 2, 0) respectively. There are orientations are (0, 0, 0), (0, 0, 1.57) and (0, 0, 1.57) respectively. The waypoints start at (2, -2, 0) and other 2 waypoints when calculated are at (4, -2, 0) and (0, 3, 0). The robots rotate correctly and travel slowly to the desired positions. Please see the video in media folder for demonstration. One thing to note was that it required slow velocities to get to the correct behavior, otherwise a lot of drifts were happening.