PA4: Auction-based task allocation ROS package

Author: Liam Prevelige

COSC69.13, Spring 2021

Method Description

This submission is a modification of PA3, changing the sending of waypoints through an auction-based allocation system. To achieve this behavior, the following updates to PA3 needed to be made: (1) `random_waypoints`: created a node to publishes random waypoints that the robots should cover. There is a check to ensure that waypoints are not too close to each other. These waypoints are the items that the auctioneer asks bidders to bid for during the auction time. (2) `auctioneer` modified the leader node for an auctioneer to send waypoints for the other robots to evaluate and, according to the bids, return the allocations to each robot following the sequential auction mechanism. (3) `bidder`: modified the follower node for the bidders that return the values associated to the waypoints.

The `tf_coordination` node, used to broadcast world coordinates to the robots, has the same function as that of PA3. Aside from minor refactoring changes, the code continues to subscribe to followers' publishing of local positions and transforms them to world coordinates.

The `random_waypoints` node is responsible for creating a service that interacts with the auctioneer, sending randomly generated waypoints as needed. A 2D array called *waypoint_map* stores a simple boolean value at each index representing one of the*n* number of possible waypoints - in my implementation, *n* is 10*10, since each target waypoint is set to be at least 1m away from the next. The Gazebo coordinate location represented by each index in _waypoint_map_ is the product of the index and the minimum distance (in this case min distance =1). Every time a new waypoint is generated, the corresponding index is set to 1, with each generation of a random waypoint ensuring its related value in _waypoint_topic_ is 0. After creating a random x,y coordinate, `random_waypoints` sends the coordinates to the auctioneer using a service, getting "success" or "break" in response. These messages relate to the next actions of `random_waypoints` - if "success," then the auctioneer has successfully receieved the waypoint and generates a new random waypoint to be used, and if "break," then the auctioneer has received enough waypoints to allocate one task to each bot, and stops sending more coordinates.

The `auctioneer` node functions similarly to the `leader` of PA3; it registers each bidder individually and stores their corresponding world coordinates. Note that the robustness of this process has been improved, with the success of the service for registration only being passed if the auctioneer successfully receives the bots name *and* finds its world coordinates by the tf broadcaster. Although the world coordinates are stored, nothing is done with them by the auctioneer directly, although is available for changes to this implementation. It's (perhaps incorrectly) assumed that the transformations by the tf broadcaster are automatically used by the bidders. The main modifications that have occurred between `auctioneer` and `leader` stem from the process to get and allocate waypoints to the bidders. While the registered bidders have yet to receive a task, a service connects the auctioneer to receive a random waypoint from `random_waypoints`, using a second set of services to request bids from all bidders. The response to this service is each bid - in this case just each bots' distance from the target waypoint based on either their initial position or last assigned waypoint. The minimum bid is then processed, with another service being useed to notify the winning bidder of their new task. This assignment is stored in a dictionary, with the process repeating itself until all bidders have received a target waypoint. Once all bidders receive a task, a notification to start movement is published to all bidders.

The `bidder` node functions similarly to the `follower` of PA3; the node publishes its initial position to the tf broadcaster and subsequently registers it's name with the auctioneer, with the assumption that the auctioneer is robot_0. Note that robustness has been improved here relative to PA3; if the auctioneer fails to respond with an indiciation of successfully finding the name and translated coordinates from `tf_coordination`, the bidder continues trying to register. As stated above, it's assumed (perhaps incorrectly) that the transformation from the broadcaster automatically updates its odom readings. Following registration, the bidder looks to connect with the auctioneer, using a custom service, to get information about a target waypoint. Once this is received, a bid is calculated by determining either current distance from the waypoint, if no tasks have been assigned yet, or the distance from the last assigned waypoint otherwise. This bid is returned back to the auctioneer. If won, a separate service is set up to listen for the auctioneer's notification of the target waypoint. This target is added to a list to be iterated over later. Then, once the auctioneer publishes a message to begin movement (subscribed to by the bidder), all won tasks are iterated over by the bidder, with the bot moving to each coordinate using basic geometry and basic movements.

Evaluation

The program works in using auction-based task allocation to create a multi-robot system that moves to random coordinates. As demonstrated in the video, the nodes successfully make bids and move to specific world waypoint coordinates. Although the case where a bot wins multiple waypoints wasn't shown, this was found to be relatively common when one or two robots were initialized far away from the [0..10] meter range. Services also ensure the confirmation of passed messages, which was a potential issue in PA3. A launch file is used to deploy the system in Gazebo with 3 robots.

There are some limitations to the implementation of the behavior:

Collision is not accounted for in this implementation. Although it's not that likely to occur given the nature of a sequential auction, there may still be cases where the path to waypoints overlap with similar timing of movement.

In addition, throughout the code, there are explicit assumptions to the following points: robot_0 is the leader, there are three robots, and services will eventually connect (i.e. the auctioneer continues trying to send the same waypoint until success). As a result, the code is not easily adapted to more general examples.

Finally, the movement of the robots is decent, but could still likewise be improved. Rather than having long periods of forward movement, a smaller, more incremental process of moving forward - and checking the heading's alignment with the desired angle - could produce more fluid movements.