Turtlebot Deployment Movement in ROS

Daniel Heideman

1 Summary

This report covers work done with Robot Operating System (ROS) nodes that move one of a group of turtlebots to a specified goal position using odometry supplied by an overhead camera. Modifications to the Navigation Stack's move_base node's parameters to work with multiple robots are discussed here, although it is not yet fully functional.

2 Big Picture

The goal of the use of turtlebots and ROS in Dr. Cortes's and Dr. Martinez's labs is to test multi-robot deployment algorithms with real robots. Using an overhead camera tracking unique aruco markers on top of each robot to provide absolute odometry for each robot, the turtlebots will each be assigned a goal position in their own Voronoi cell to collectively cover a region with preference given to areas deemed to be of high importance.

A TurtleBot 2, with Microsoft Kinect 3d sensor and Kobuki base

3 My Contributions

This quarter I worked on getting the goal-based movement node to run in unique namespaces for each robot to allow multiple robots to operate simultaneously with the same ROS master computer. While most message topics are remapped to include the robot namespaces with a simple <group> tag in the launch file, tf message topics require special consideration. While the published message topics will include the namespaces, the nodes will not automatically update the topics they try to read from. Thus parameters must be modified to allow the nodes to read from the correct topics.

4 Methodology

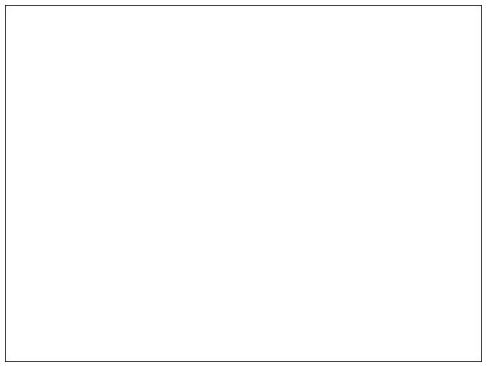
This section contains a list and high-level descriptions of the movement nodes worked with during Winter Quarter 2015.

4.1 Move Base

The move_base node is a node in the ROS Navigation Stack that will plot and follow a path through the robot's environment based on an obstacle costmap. This node is often used in conjunction with the turtlebot's Kinect, Kobuki wheel encoder and gyro sensors and an AMCL or SLAM node, utilizing a SLAM-generated map to move through a room. However we wish to use this node without these sensors and nodes, instead providing odometry with an overhead camera and giving a blank map indicating an obstacle-free space for use as a goal-driven movement node for multiple robots.

To overcome the problems with nodes looking for data on the wrong topics due to the namespacing required for running multiple robots, some move_base parameters need to be modified. Specifically the "robot_base_frame" parameter of the local and global costmaps needs to be adjusted to include the turtlebot's namespace from "/base_footprint" to something similar to "/boticelli/base_footprint", where "boticelli" is the example robot's namespace. This can be done in the local_costmap_params.yaml and global_costmap_params.yaml files in the params folder of the Navigation Stack package, which must be manually changed for each turtlebot, or in the launch file using a tag such as: <param name="move_base/global_costmap/global_frame" value="$(arg robot)/base_footprint"/>.

A blank white map with dimensions of 640x480 m is used as the costmap, provided by the map_server node. The dimensions are given as such to allow the camera odometry to be given in either meters or pixels, as the size of the costmap does not matter, so long as the odometry and goal values stay within the bounds of the map. However using camera pixels as the unit for the odometry does not work well with move_base, as ROS measures in meters and doing so will make move_base think that the turtlebot will have moved 170 meters when it has really only moved one meter and regulate the motor speed accordingly.



lab_workspace.pgm, 640x480 pixels
(white with black border)

lab_workspace.yaml

```
image: ./lab_workspace.pgm
resolution: 1
origin: [-1.1, -1.1, 0.0]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

5 Tips

Move_base is overkill for what we currently plan to use it for, as its main feature is its ability to plot and follow paths around obstacles described in a costmap and we are not dealing with any obstacles. However it could become very useful if we ever need to avoid obstacles in future implementations.

The node uses the global costmap to determine every movement, so any obstacle to be

avoided must be included. However if there are no obstacles to avoid, we can use a blank white map as described above. This defeats most of the purpose of using move_base in the first place, however, leaving it as just a straight-line movement algorithm.

Move_base does not use standard ROS messages nearly as much as most other nodes, instead making use of the Navigation Stack's action messages and tf for everything except the costmap and direct sensor inputs. Action messages work better than standard ROS messages for setting a goal position and reading move_base's status, as they need to be sent only once. The goal can also be set with standard ROS messages, but special care must be taken to keep the goal to be re-updated too frequently, which causes move_base to get stuck re-initializing its plan and never start moving. The robot's odometry is given in tf frame relations, which makes it hard to sift out data coming from different sources. This ends up being a problem when both the camera and the robot's Kobuki base publish tf relations for the robot's odometry.

6 Pitfalls

There is very little documentation for the move_base node, although the source code is available online. Thus it is very difficult to find the cause of any problems related to its operation.

The velocity smoother that is used by move_base does not work. It starts up but does not subscribe or publish to any of the ROS topics it is supposed to, causing the turtlebot to be unable to move while running move_base. The velocity topic was remapped to bypass the velocity smoother for testing, but the movement ended up being quite rough. The smoother works in other applications, such as the demo AMCL node, but does not work alone or with move base even though its launch file was not altered in any way. This causes "stuck" and "unstucking" errors when running move_base, as the robot is not moving even though the node thinks it is telling it to.

The camera currently gives odometry in camera pixels, but move_base uses units of meters. However to change the units given by the camera would mess up most other nodes using the camera for odometry, and it is not practical to convert between pixels and meters within the launch file for move_base. Without publishing data in both units, the only way to make move_base work is to convert all the other nodes, which is relatively easy except for the number of nodes. However in testing this is not a problem, as move_base is the only node being run that requires odometry, so this problem is not terribly urgent.

Still, there is a problem with overlap between the camera odometry and the Kobuki base odometry, which causes two completely different sets of data to be published on the ROS topic "~/odom" and a combination of camera and Kobuki odoms in one tf transformation. In the standard ROS "~/odom" messages, the two sets of data can be separated out if need be, but in the tfs they cannot be easily separated without changing the frame transform name being published by the camera localization node.

Currently the turtlebot will move seemingly randomly when the move_base node is run with a simple goal set. With a goal of (1, 1), the turtlebot drove relatively straight, then stopped and said it had reached its goal, but it was nowhere near (1,1) in the camera's coordinates. When run again without moving the turtlebot, it spun in place and again said it had reached its goal. Upon pushing it to another location, it found its way back to what it had defined as its goal. However, upon restarting the program or lifting and moving the robot, it moved to a different location. Also, rotating the aruco markers on the robots does not change which way they

wander, but rotating the robot does.

In these ways move_base appears to be using the Kobuki's odometry transform and not the camera transform, while the transform specified in move_base's parameters appears to be a sum of both. Again, changing the name of the tf frame might fix this problem, but this node operates so unfamiliarly that this conclusion has taken the entire quarter to come to and no time was left to test it.

7 Conclusions & Future Directions

For this application, move_base still does not yet work. Even after 10 weeks working on it and looking through the source code, I still have very little idea how it works and what it is doing. There simply is not enough documentation, both for namespaces in ROS and move_base itself. That being said, I do have some proposed solutions I would like to try out but haven't had the time to.

As stated before, I believe that changing the topic in standard ROS messages and in tf the camera publishes its odometry on would allow move_base to work completely off of the camera odometry without the Kobuki odometry getting in the way. This would require changing the camera localization node itself as well as finding and changing any parameters that rely on the Kobuki base odometry, but would also keep this kind of problem happening to any other node later on.

However, I have no solid idea on how to fix the velocity smoother nodelette. Like with move_base, there is very little documentation on it or what a nodelette is. The only lead is the AMCL demo for which it appears to work. It is possible that there is a parameter set somewhere in the AMCL launch file that is omitted in the move_base launch file that allows it to work with one but not the other, but getting the velocity smoother running will not be quick and easy.

If we can get the velocity smoother running before move_base, an alternative is to get the much-simpler first_turn_then_move working smoothly. The first_turn_then_move node is a basic goal-driven movement node that was created to test deployment algorithms while move_base was being set up to work with the camera and multiple robots. It moves in a straight line to the goal without referencing a costmap or checking for obstacles, and has no velocity smoothing implemented. However, given the numerous problems involved with making move_base work with this setup and because the current deployment setup restricts the turtlebots to their own Voronoi cell and they will never need to avoid any obstacles, it may be worth some time to improve first_turn_then_move.

To make the turtlebot movement more fluid, the output velocity of the node would need to be remapped to run through the velocity smoother nodelette borrowed from move_base, along with the camera odometry. The camera odometry would need to be published on a separate topic from the wheel encoder and gyro odometry to allow the node to read only the camera odometry. And like move_base, the odometry must be in meters rather than pixels for the velocity smoother to run properly. However it is limited to use without the possibility of collision, which may become a problem for future deployment algorithms. As such this would be only a temporary fix, with getting move_base working remaining a priority.