RBE 3002 Final project Appendix

# Approach

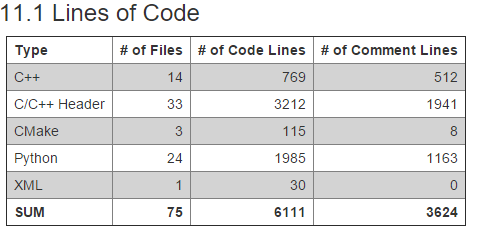
An effective approach to successfully exploring an unknown area using the turtlebots.

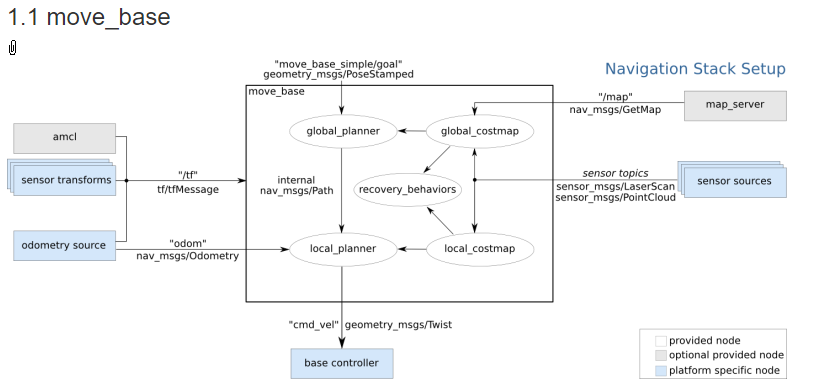
1. Use a basic motion to fill in the map that can be sensed from the initial position.
2. Create or use a package or node to find a frontier that is offset into known space.
3. Use your A-star algorithm to plan a path to the frontier.
4. Select your first way point and pass it to the **move\_base\_simple/goal**.
5. Figure out when the move base node has finished.
6. Subscribe to the **move\_base/status** and wait for goal reached and receive status updates. Status 1 = goal set and planning ; status = 2 navigating ; status 3 = goal reached
7. Or Subscribe to the **move\_base/result** to get msg when the goal is reached. Also create a timeout and either publish a new goal or publish a **move\_base/cancel** message.
8. Repeat steps 1-4 until all frontiers of a sufficient size has been reached.
9. Let us know you are done. Flash some lights, play some music, or just print it to the terminal.

# Move\_base

The ROS navigation stack utilized in the default setup of the turtlebots is very advanced and uses thousands of lines of code to successfully navigate. For this reason we recommend using it to ensure accurate navigation. The following statistics on the move base package elude to how complicated the work you have completed so far really is.

These statistics do not including Gmapping or AMCL





Below is an example python program for driving a robot using the move\_base action library server. Warning! we would recommend publishing and subscribing to the topics instead of using the client server approach in the example. The following is only intended as an educational reference.

<http://ros-by-example.googlecode.com/svn/trunk/rbx_vol_1/rbx1_nav/nodes/nav_test.py>

# Open Slam

<http://openslam.org/> is the organization that created gmapping. Please feel free to read up on the research and approaches used for this advanced set of software. Together with the AMCL and move base packages the robot can successfully navigate to a goal.

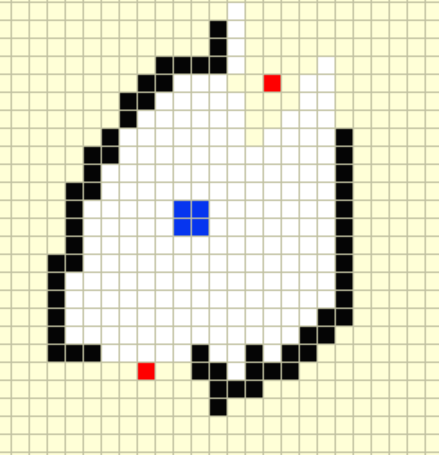
# Frontiers

The recommended approach for frontier exploration is as follows, but is by no means meant to limit which way you would like to implement the frontier search.

Implement a breadth first search looking for a cell that is marked -1 for unknown space. Then breadth first search from that cell looking for neighbors who have a value of -1 and a neighbor greater than 0 but less than the cost of a wall or obstacle. Use the number of frontier cells to estimate the length and centroid of the frontier. If no frontier of sufficient size is found you are complete!

If neighbor == -1 and (neighbor.neighbors >= 0 and neighbor.neighbors < obstacle cost):

Add to Frontier

Make sure to offset your centroid to an achievable goal for your Astar and for your move\_base node. If your waypoints are conservative the frontier should fill in before you reach the end goal. 

It will be crucial for your frontier and Astar searches to be rerun after each way point is achieved.

Optionally you may also want to implement a rotation behavior using the move\_base node when the robot achieves a waypoint to ensure the robot senses the obstacles. Specifically if the robot is approaching from a non-ideal angle or if the map is lagging on its updates.

# Using Ros Packages effectivly

To use outside packages effectively it is extremely important to have a properly configured work space. Most packages use a combination of files to work effectively. Check out the <http://wiki.wpi.edu/robotics/ROS_File_Types> to learn more about file types.

1. Setup a workspace and package.
2. Use rosrun to execute a python program that you have.
3. Make a launch file to start your node.
4. Examine the packages you would like to use by adding them to your workspace. Then start at either the launch file or program/node (python or C++) you would like to use.
5. Look up the syntax of XML, YAML, robot, and URDF files as necessary.