SFU RAMP LAB

Powerbot

Navigation, Pan-Tilt Unit, Octree Service Client, and Troubleshooting

Charanpreet Parmar

8/25/2011

csp6@sfu.ca

Table of Contents

Move Base

One of the central components of the navigation stack for ROS is the move_base node. This node will input information about transforms (topic: /tf), odometry (topic: /odom), the map (topic: /map), sensor information as LaserScans or PointClouds (topic: needs to be defined manually) and a goal for the robot (topic: /move_base_simple/goal). For our purposes, we do not use the Adaptive Monte-Carlo Localization (AMCL) and the sensor information from both the laser and the sonar are LaserScans and published via the /scan and /myscan topics respectively.

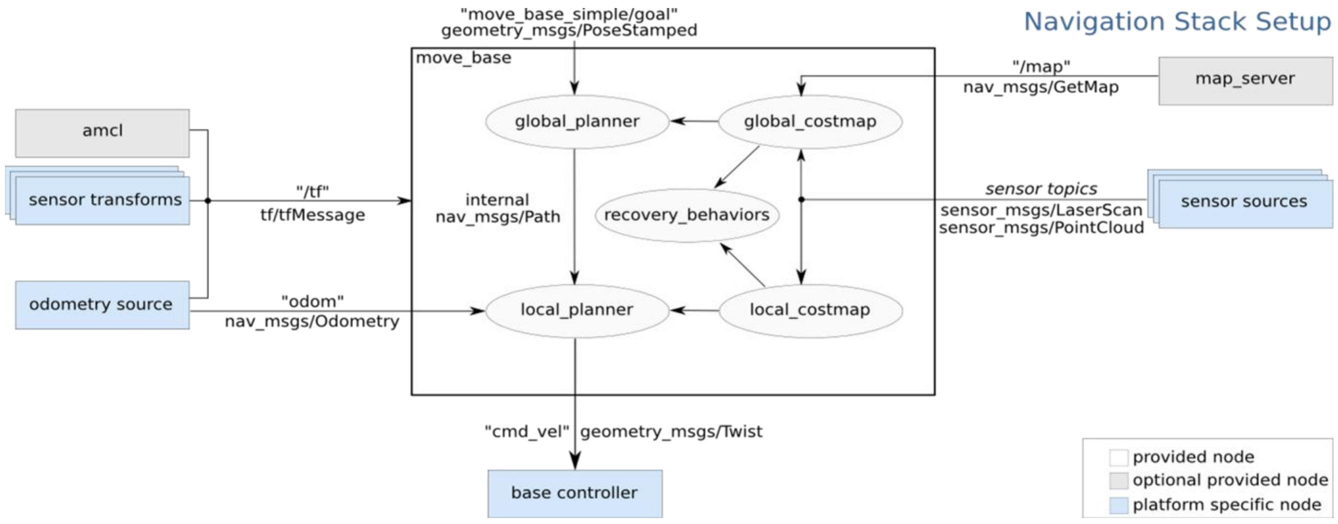

Figure 1: Navigation Stack Setup Diagram via http://www.ros.org/wiki/move_base

Once everything is properly connected, move_base can properly command the robot to navigate from its current position to nearly any goal. The majority of the internal components of the node do not need to be modified beyond the basics or at all (global_costmap, local_costmap, global_planner, recovery_behaviour) as they don't need to be as diverse as the local_planner in order to function on the multitude of robots that ROS can run on. All that was done for these components is a GMapping node was setup to create a Simultaneous Localization and Mapping (SLAM) map to publish to the /map topic and the sensor information was then published to the SLAM node, as well as to move_base in order to create the local and global costmap. In order to reduce the probability of the robot behaving peculiarly, the recovery behaviour, which consists of behaviour such as spinning in place in an effort to clear obstacles from the costmap and other such hampering behaviour for the robot's navigation, is simply disabled. The global and local costmap also need to be configured, but this simply involves setting which topics to use for

LaserScans and PointClouds, as well as window size and a few other relatively straight-forward parameters to be set. It is also important to ensure transforms are done correctly as otherwise it will result in an error or even loss of data. The parameters to be set are: the frame_id for the transformation, which for PointClouds is embedded into the header so is optional to fill out; the data type for the data, which can either be PointCloud or LaserScan; the Boolean variables for marking and clearing obstacles for the data source; the topic on which the data comes from; the obstacle range and ray tracing range for marking and clearing obstacles; the persistence of the obstacle, which is the expected rate for the data to be updated, 0 uses the latest data ; and for PointClouds, there is a max and min height that can be set to only use data within a certain height range.

Global and Local Costmap

During simulation, there appeared to be an issue with using the sonar to mark and clear the global and local costmaps. What would happen was that each sonar sensor would have a field of vision of 30 degrees and it would update the distance from the obstacles at a relatively low frequency. This would lead to errors in situations if objects around it were moving, such as a person, it would take it much too long for it to mark the person on the costmap or clear them, which would lead to the robot refusing to go to an area where there was nothing and even go to areas where a person had just come. Another problem was that if two adjacent sonar sensors had a drastic difference between them, the sonar_to_laser node would interpret that there was an edge or wall type obstacle between the two sensors, effectively trapping the robot within the range of the sonar. As such, it was decided that it would be best to remove the marking capabilities of the sonar from the costmaps, effectively making it only useful for clearing out obstacles from behind it since there is no other sensor there. This removed any possibility of the sonar ruining any of the global or local paths generated with any phantom obstacles, but the low range sensing capabilities of the sonar are still retained within the failsafe node. The cost of path that goes reverse is high enough so as to force the Powerbot to use a reverse path as a last resort only so there is little risk involved with the sonar being the only source in the rear of the Powerbot, but with the failsafe any risks should be even further minimized. At this point, there is only the local_planner to be setup for the navigation stack, and although it too is simple, there are some subtle details that we must carefully take into consideration.

Local Planner

Planner Options

There are two built-in options in ROS when choosing a local planner. The base_local_planner can either use trajectory roll-out or a dynamic window (DWA) approach when making the local path. The dynamic window approach is typically considered better for robots with higher accelerations and speeds since it will be able to plan more diverse paths by taking into account the robots acceleration limits. For slower robots, which is the case for the Powerbot as speed restrictions need to be implemented to reduce possible damage to peripherals such as the arm and sensors, trajectory roll-out is typically more efficient. The major difference between the two options is that the DWA option takes the robots accelerations into account and does not simulate a path. As they both, more or less, use the same algorithm for planning the path, there is little difference in choosing between them, apart from whether or not acceleration or computing power or time are significant factors, which they aren't in our case. The second option is the use the newly released dwa_local_planner, which, as the name suggests, uses the dynamic window approach in order to plan paths. This is a package was just released with the currently latest version of ROS, Diamondback, and was listed as the experimental local planner for C-Turtle. This planner has the advantage of being dynamically reconfigured, unlike the base_local_planner, as well as being a faster planner in certain situations. Being dynamically reconfigurable allows the parameter to be tweaked without having to relaunch the node. In other words, it allows you to change parameters as the Powerbot is moving. This makes it much easier to adjust parameters and tweak them for the most optimized behaviour. After much experimentation with both, I found better results with the base_local_planner using trajectory roll-out. Although both technically worked, I observed that both the local planners using the DWA approach seemed to preferred to go at much slower speeds than desired and also had difficulty making turns occasionally, most likely due to some parameters being set non-harmoniously or sub-optimally, and the trajectory roll-out planner ended up working much better and is backwards compatible with C-Turtle, unlike the dedicated DWA planner, which can be advantageous when dealing with the more out of date ROS machines in the lab.

Parameters

The vast majority of the parameters are simple enough to set with few complications arising due to them being set incorrectly. Most of the parameters are basic information about the robots capabilities so as to allow the planner to create as best of a path as possible. For the most part, there should be no problem setting the maximum angular and linear velocities, acceleration limits, escape velocities arbitrarily so long as the robot is capable of attaining what they are set at. The velocity samples, which are used to discretize the velocities into a velocity space, should be slightly higher than the defaults; I found best results with 11 for x-direction and 20 for theta-direction, and the path and goal bias can also be changed, though there is little need since it works fine at default settings. It is very important to make sure holonomoic_robot is set to false as otherwise the planner will attempt to move in the y-direction (ie, perpendicular to the direction it as actually capable of moving). Most of the other parameters can be left at their default values. The key parameters for the navigation are dwa, sim_time, min_vel_x and xy_goal_tolerance.

The dwa is a boolean variable that toggles between the trajectory roll-out and and dynamic window approach for path planning. Again, I found best results using trajectory roll-out by setting this to false. The other three parameters need to be set harmoniously so as to eliminate any undesired behaviour. The key for the local planner to working well is to keep the xy_goal_tolerance low, minimum velocity high and the sim_time high. A low goal tolerance allows the robot to get much closer to its desired goal before halting the planner commands. A high minimum velocity ensures the robot will move faster, as there are situations where the planner prefers to go at the minimum allowed velocity (or even lower) and having this too low can cause simple paths to take greatly longer than desired or even needed for adequate performance. Increasing the sim_time, which is the length of time a path will take to execute, will let the planner plan for a longer period of time and results in a much smoother path and even causes the robot to prefer moving at different velocities (closer to maximum instead of minimum, or vice versa). It may seem odd at first to plan for a length of time instead of distance, but by planning for a length of time instead of a distance, it allows the paths to be much smoother over that period of time, but makes it slightly more complex to set the other parameters.

The key to these three parameters is that the goal tolerance must be at least twice as large as the magnitude of the sim_time multiplied by the minimum velocity, which is the shortest distance the Powerbot can travel. This ensures that the planner will be able to plan a path to a point within a radius of xy_goal_tolerance from the goal. If this is not true, the planner will fail to consistently find a path to the goal and simply rotate in place once it reaches the outer edge of the acceptable circular region (of radius xy_goal_tolerance from the goal) for an acceptable final pose as any paths it plans will cause a local minima type situation. By this I mean that it will reach a point where the score of any path is lower than its current position (as all paths are either farther away from the goal or equidistant to the goal but at a different position) and will thus want to stay in the same spot, but at the same time it will want to reach the goal and this results in simply spinning in place, and if you're lucky the robot may drift into an acceptable x,y-position. To illustrate the issue, see Figure 2. If the shortest path it can travel is larger than two times the distance xy_goal_tolerance, it will never reach the goal.

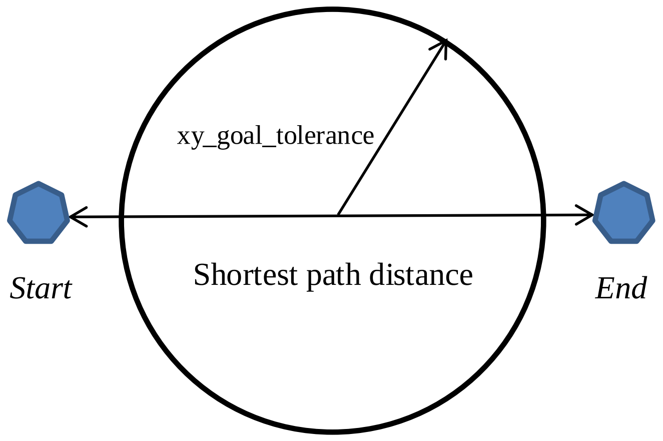

Figure 2: Planning for shortest distance

In order to find the best combination of values, it is best to first select an appropriate xy_goal_tolerance for the purposes of the navigation and then experimenting to select the other two parameters. I found best results with a sim_time of 2.62 seconds and a minimum velocity of 0.035 metres per second, which is enough to reach a goal tolerance of 5 centimetres, although there are much better results for slightly larger tolerances such as 7.5 centimetres in as far as total time for path execution is concerned, at the cost of a slightly larger distance from the goal.

Pan-Tilt Unit

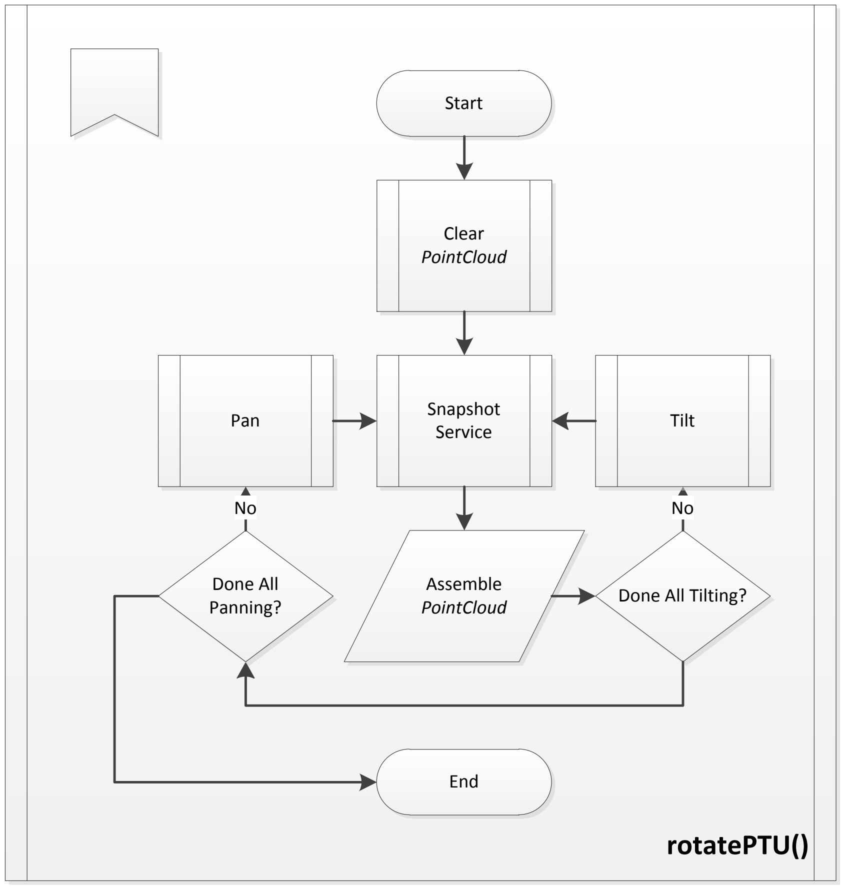

Figure 3: rotatePTU Flowchart

Overview

Due to an error with the Swiss Ranger 4000 (SR4K) involving the phantom obstacles appearing when the camera moves, there needed to be a work around developed in order to use the SR4K data. The one we devised was simply to stop the Powerbot every so often, based off of its translational and rotational distance and angle traversed or based off of a timer, and then take and assemble multiple snapshots of what the SR4K can see during the time the Powerbot is stopped. In order to get the static images without moving the base, we simply use the Pan-Tilt Unit (PTU) to pan and tilt to various positions and then take a snapshot of what it sees to be assembled into a larger PointCloud. This gave us the ability to use a clean SR4K data, as well as greatly increasing the field of vision for the SR4K.

Server

The control for the PTU is split into two parts: the actionlib server and the actionlib client. Actionlib is similar to the ROS service call in the sense that it allows users to request other nodes to perform a function or calculation like a service, but has added options such as not waiting for a reply, cancelling after the goal or objective has been sent, the ability to receive feedback from the server and other features that allow users to perform tasks while waiting for a reply from the server. The node which we use for the PTU is ptu46, which is dependent on the ptu_control package. ptu46 itself comes with the server node, written in python, and the launch file to launch the server as well as the controller. The ptu_control on the other hand contains the necessary headers and actionlib generated files need for the client and server. The way the server operates is that it will subscribe to an actionlib sensor_msgs/JointState topic and move the PTU accordingly. It also publishes task progression information via a Feedback actionlib topic and the ability to cancel goals via an actionlib Cancel topic. There are two of each type of these topics: one set for setting a goal, and another for resetting the PTU to the (0,0) position. For the client we will use, the one for resetting is simply ignored and not used for our uses.

Client

In order to be able to control the Powerbot's movement, a node needed to be created to subscribe to the move_base generate velocity commands and then either let them pass through or set them to zero in order to stop the Powerbot. As such, the brake node was made to do so. It will subscribe to the /cmd_vel topic from move_base and publish the same data, or the zeroed set when necessary, to the /p2os/cmd_vel topic. For this node to control the PTU the way we intended, it needed to subscribe to a topic with appropriate odometry or transform data. For simplicity, the /pose topic is used (/odom in move_base by default; remapped in the launch file), although there are plenty of other options to use. Once we can see where the Powerbot is, we can use this data to then see how far it has moved as well. We will use both translational and rotational distance and angle traversed to then control whether or not to stop the Powerbot and run our snapshotter and assembler, which takes snapshots of PointClouds and then assembles them. Since our assembled PointCloud will have a horizontal range of well over 180 degrees, a larger angular tolerance could and should be used. The default is set to 60 degrees in both directions, giving a 120 degree range that the Powerbot can move before having to rescan. For translational distance, since there is not much likely to change while the powerboat moves forward and back, a larger 2 metre range is given before each rescan. Both of these setting allow us to balance interfering with the navigation and having valid data, but are easily tweaked for different situations. The timer was created to allow the Powerbot to rescan if it has not moved for a certain time frame. This is critical for allowing obstacles which may interfere with movement to be cleared out from the assembled PointCloud. An example of this may be if there are many people walking by the Powerbot or surrounding it. If the Powerbot scans them during one of its scans, it will see the obstacle there even long after the person or obstacle has gone. In order to correct for these types of situations that can possibly arise, a timer was created.

Since we want to waste as little time as possible panning and tilting, there was a simple algorithm developed to reduce the total number of pans and tilts needed to scan a given range. The basic pattern which it will pan, tilt, assemble and snapshot is as follows: snapshot, tilt, snapshot, assemble, pan, snapshot, assemble, tilt, snapshot, assemble, pan… where it will tilt an appropriate number of times per pan. For further tweaking of the pattern, the angle between each pan and tilt, as well as the total range to pan and tilt, can be set via the Parameter Server. It is important to note that since the SR4K data does not immediately clear out phantom obstacles, the node needs to wait a brief duration before taking the snapshot. To do so, the node is put to sleep after the panning or tilting is completed. From experimentation, the lowest period to sleep with consistently clear data was found to be 0.4 seconds. This can be set much lower, but there is no guarantee that all obstacles will be cleared out.

In order to have accurate data in the PointClouds that we will assemble, we need to ensure that the transform from the SR4K and he PTU is as accurate as possible and the data is as recent as possible. Since we don't want to rescan frequently, and also since the Costmap will throw out data that is too old, there needs to be a function that will update the timestamp on the data. Although it may seem like a good idea to do it in the brake node, there will be several times where the brake node will be too busy to update a timestamp during pan-tilt sequences and other computationally or time heavy processes. Therefore, it is best to create a new thread, via creating a new node in the package that will update this timestamp as frequently as possible. To do so, the brake node will publish a PointCloud, to which this PCL_publisher node will subscribe and then output the refreshed data as a PointCloud and PoitnCloud2. For the transform, we once again want the transform updater to be in its own thread to allow more frequent updating without adversely affecting the brake node so it too is put into its own node. The transform data for the PTU itself comes from the /ptu/state topic which gives the PTU's most current known location. From here we simply publish this data and create a static publisher transform publisher between the SR4K and PTU.

The snapshotter, which has been referred to several times previously, is used in order to take snapshots of the SR4K at a series of positions. It is found in the laser_assembler package, which is in the laser_pipeline stack, and can be used to assemble PointClouds in a buffer over a certain time. Since we only want one PointCloud per snapshot, the buffer size option is set to 1 and the time period is set to be between the last snapshot and the current time. This node offers a service, which the brake node then calls, to retrieve the snapshots. To assemble these snapshots, the points and other channel data are then added with any previous snapshots taken in the pan-tilt sequence into a PointCloud in the brake node which is published after finishing each tilt section in the pan-tilt sequence. The reason for publishing so frequently is so as to allow move_base's global and local planners to use this PointCloud data to plan out the path's to reduce total planning time once it has finished the pan-tilt sequence. The algorithm used to do this can be seen in Figure 3 at the beginning of this section. Once completed, this will result in one big PointCloud made up off several snapshots taken using the SR4K and the PTU.

Parameters

The client for the PTU was designed to have the ability to tweak any of the parameters that are control how the PTU operates. The key parameters to tweak are the distance tolerance and angular tolerance for the node. These are the parameters that will allow the user to select the range the Powerbot can travel before a rescan is needed. The range over which to pan and tilt can also be set. This can be tweaked for a smaller or larger range to either gather more data or smaller to speed up the pan-tilt process. To go deeper into this, one can also set the angular step taken during the panning or tilting of the PTU to change the density of the points taken. Smaller steps will create a denser and more detailed cloud due to overlap between snapshots, while a larger step covers a larger area in a shorter time at the cost of total number of points. If the need arose, it is also possible to bias the PTU to aim towards a certain direction. This can be useful for when the PTU unit itself is found to not be mounted properly to the Powerbot by being off by a few degrees, or other situations where the PTU needs to aim more towards a certain direction than another. Another parameter that can be experimented with is the timer. Since it was added rather late into the nodes development, it was not as thoroughly tested as the other parameters and left much room for tweaking. This is the parameter that controls the time period for which to guarantee that a PTU rescan is performed. Parameters to be cautious with while tweaking are the pan and tilt velocities. Although the manual says that they are capable of speeds up to 60 degrees per second, I found that their performance was unstable and the servo motors would slip and lose their zero position. This may be in part to due to the weight of the SR4K, but either way I advise against pushing the velocities beyond 30 degrees per second. For the most part, these parameters do not need to be tweaked, but the option to do so is there.

For the PCL_Publisher node, the only parameter to be tweaked is the period at which to update the PointCloud and PointCloud2 that it publishes, as well as the names of the topics if need be. The ptuTF node, which publishes the transformation for the PTU, is completely automated since there is no need to tweak this node. The topic it subscribes to can be changed if needed, but this is only likely to occur if the server was to be redesigned.

Swiss Ranger 4000 and Intensity

Another issue with the SR4K was that it would fold back points beyond its set range. That is, if for instance it was set to have a range of 5 metres, a point at 7 metres would be folded back to 2 metres (7 metres – 5metres). To fix this, we increased the range as much as we could (to approximately 10 metres) so that any point that would be folded back could be filtered out by amplitude thresholding. However, while running some tests involving tweaking the parameters for the SR4K node, primarily for the amplitude thresholding, it was found that even if the threshold is set to its maximum value, a point at which the pointcloud is reduced to only a handful of points, the folded back points will remain. That is to say, it is impossible to filter all the transient point based solely off of the amplitude of the point. However, I also found that the intensity of the folded back points was much lower (value of ~100 vs ~8000), so it may be viable to filter the points based off of their intensity. Since there was no intensity filter on hand or built into the SR4K, there may be some adverse effects in doing this, such as filtering out wanted points, or anything else that I may not have considered, but there doesn't seem to be another solution to the folding back of points than this.

Octree Service

Overview

The Octree Service was a simple way to incorporate the octree into the costmap. Simply put, the octree service takes in two points relative to the /map frame and returns whether the box created has any potential obstacles within it or not. Unknown points are ignored since most of the map will be unknown initially so any area queried behind the Powerbot will be an obstacle, rendering the Powerbot immobile since the Powerbot will be assumed to be in collision. Once this reply is received, the client can add that point to the costmap, in our case via an artificial PointCloud around the Powerbot. Unfortunately, this process can be very time consuming as the service will take some time before it responds. This limits the ranger over which we can actually scan for obstacles in order to balance the total area surveyed and the time taken before the surveyed area can be updated. After looking into the time it typically takes for a query, as well as the distance traveled by the Powerbot over a given time frame, it was decided to go with a 2 metre radial search. This was primarily because a 2 metre radius would be sufficiently large so as to allow for the global and local planner to plan their paths for at least the distance that they can travel in the time it takes to query that area.

Client

Due to the significant time taken by the service to do its calculations, the client which would call the service was designed in such a way so that it would query the points nearest to the Powerbot first and then expand radially outwards. In order to reduce set up time within the service, 50 points would be sent in each service call, the total number of points being 197 for a resolution roughly equivalent to the volume of a slightly inflated robotic arm. The client was made so as to allow any volume on top of the Powerbot to be queried at any rate, as well as the number of points to be queried so the 50 cells per service call and total number of cells can be changed via the parameters.

Server

See Dianna Yee’s Report.

Parameters

There are some parameters that can be changed for the Octree Service Client. The circle about the Powerbot for which the service is called is discretized into several square cells defined by what the cell size and diameter parameters are given. The diameter of this circle can then be found by multiplying the cell size and the cell diameter. Since the size of the circle can be changed, the rate at which the circle is updated at also needs to be set to help compensate or make up for the time lost or gained. The number of cells to check at a time, ie the number to send in each service call, can also be changed, as well as the physical distance from the powerboat at which to place the centre of the circle.

Navigation Stack Setup and Troubleshooting Guide

Setting up the Simulator launch file

1. Create a ROS package to place all the needed .yaml and .launch files. You can do this by running:

   ```
   roscreate-pkg <Name of project>
   ```

   and replacing <Name of Project with the name of your project>. Since there are no needed dependencies, you do not need to add them.

2. Create your central .launch file. For example, use Sim.launch. Inside the file, enter

   ```
   <launch>
           <master auto="start"/>
           <param name="/use_sim_time" value="true"/>

   </launch>
   ```

   You will use this template to add all necessary nodes and files to this .launch file **BETWEEN** the <launch> and </launch>. There are example .launch files in the appendix.

3. The first node to be added will be the stageros node. This is the simulator that will give you all the laser and sonar day, as well as information for the optometry, and the robots position as time goes by. Add it to where the blank line is after <param name="/use_sim_time" value="true"/>.

   ```
   <node pkg="stage" type="stageros" name="sim"args="[Full Path to .world file]">
           <!--<remap from="/cmd_vel" to="/stageros/cmd_vel"/> -->
           <remap from="/laser_scan" to="/scan"/>
           <remap from="/odom" to="/pose"/>
           <param name="base_watchdog_timeout" value="0"/>
   </node>
   ```

4. **(OPTIONAL)** If you want to use the simulator shunt, uncomment the second line (the remapping of the /cmd_vel topic) and add the following:

   ```
   <node pkg="sim_to_p2os" type="sim_to_p2os" name="sim_shunt">
           <remap from="/cmd_vel" to="/p2os/cmd_vel"/>
   </node>
   ```

   This node was primarily created as a simulator for the p2os driver node and is not needed.

5. Next we can add the Simultaneous Localization and Mapping (SLAM) node which will generate our /map topic from the /scan from the stageros node. Add the following to your launch file:

```
<node pkg="gmapping" type="slam_gmapping" args="scan:=scan" name="SLAM">
        <param name="maxUrange" value="18.0"/>
        <param name="maxRange" value="18.0" />
        <param name="xmax" value="20.0"/>
        <param name="ymax" value="20.0"/>
        <param name="xmin" value="-20.0"/>
        <param name="ymin" value="-20.0"/>
        <param name="delta" value="0.05"/>
        <param name="map_update_interval" value="0.5"/>
        <param name="linearUpdate" value="0.5" />
        <param name="angularUpdate" value="0.5" />
</node>
```

6. **(OPTIONAL)** If you want to use sonar in your simulation, you can add the following lines:

```
<node pkg="sonar_to_laser" type="sonar_to_laser" name="SonartoLaser" respawn="true"/>
<node pkg="tf" type="static_transform_publisher" name="Sonar_Scan" args="0 0 0.445 0 0 0
/base_link /sonar 100" />
```

The first node will call on the Sonar to Laser program and the second will generate a static transform for the /sonar to /base_link frame. You will also need to modify the .world file you are using for your simulation. In matt2.inc, uncomment the powerbot_sonar ranger definition at   the top of the file and in the powerbot pioneer_base definition. You need to use the stage node made my Matt to use the sonar in stageros so make sure you have it installed. You do not need to overwrite the default stage package, just make sure when you add it to the ROS_PACKAGE_PATH it is higher on the list.

7. The next node will be move_base. This is the central node for the navigation for the robot. It will take in the sonar, laser, and point cloud data and add it to the costmap. It will generate the local and global paths. It will give the robot it's command velocities. It will do most of the transforms for you. It is very important that this is set up correctly. Add the following:

```
<node pkg="move_base" type="move_base" respawn="false" name="move_base"
```

```
output="screen">
                              <rosparam file="$(find <Name of project>)/costmap_common_params_sim.yaml"
command="load" ns="global_costmap" />
                              <rosparam file="$(find <Name of project>)/costmap_common_params_sim.yaml"
command="load" ns="local_costmap" />
                              <rosparam file="$(find <Name of project>)/local_costmap_params.yaml"
command="load" />
                              <rosparam file="$(find <Name of project>)/global_costmap_params.yaml"
command="load" />
                              <!--<rosparam file="$(find <Name of project>)/base_local_planner_params.yaml"
command="load" /> -->
                              <!--<rosparam file="$(find <Name of project>)/dwa_local_planner_params.yaml"
command="load" />-->
                              <rosparam file="$(find <Name of project>)/navfn.yaml" command="load"
ns="NavfnROS" />
                              <param name="controller_frequency" value="5.0"/>
                              <param name="planner_patience" value = "0.0" />
                              <param name="recovery_behavior_enabled" value="FALSE" />
                              <!--<remap from="/cmd_vel" to="p2os/cmd_vel" /> -->
               </node>
```

Remember to replace the <Name of project> with the name of your project. Also, uncomment the last line (the remapping of the command velocity) if you will be using the optional simulator to p2os node in step 4. Also, remember to choose between the base local planner and dwa local planner my uncommenting which one you need. **Only uncomment ONE at a time.**

8. **(OPTIONAL)** If you want to have a 3D URDF model of the Powerbot, add the following:

```
<include file="$(find <Name of project>)/display.launch"/>
```

9. Now that you have all the nodes you need, you can now add the .yaml files, and any other .launch files, needed for this project. YAML files are sensitive to TABs and spaces so be careful when creating them. You can find .YAML files in the index. If you want to modify the parameters, use the parameter table provided.

10. If you want to add additional components to the simulator, such as the SR4K, refer to the Robot.launch file in the appendix, as well as the costmap_common_params_robot.yaml. For testing other packages, such as the Brake package, refer once again to the Robot.launch and costmap_common_params_robot.launch and remap and reroute topics appropriately.

11. Once all the files are copied or created, the simulator should run perfectly fine. If there are any errors, here are a few tips:
    a. If there is an error regarding a .yaml or .launch file, check if the error provided can give you a hint about what's wrong. Usually the error will tell you enough information about why you received the error. If the error involves including a node, often the problem lies in the node not being properly built. To do so, change into the node packages directory and then run “rosmake --pre-clean”. This will often fix most problems related to nodes. If you are changing any parameters, make sure you have done so correctly. You can verify setting parameters by using rosparam commands.
    b. Run rxgraph to see if the topics go where they should go. For example, you need to make sure that the velocity commands from move_base reach stageros somehow. Run rxgraph, select “All Topics” and find move_base. Find a topic with “cmd_vel” in the title (like “/p2os/cmd_vel”) that is published from move base (indicated by an arrow coming out of move_base) and that it is going to where it's supposed to go. The same can be done for sensor data.
    c. Run “rostopic list” and “rostopic echo” to check if your topics are being published or have valid data. Often a topic will exist but will not be published to, causing errors.

Setting up the Powerbot launch file

1. Make sure you have neccesary nodes installed before running the Powerbot. If you are using the SR4K, you will need: The brake node package, along with all its dependencies, the sonar to laser node package, along with all its dependencies, the ptu46 package and its dependencies the swiss ranger node, along with all its dependencies. Dependencies can be found listed in the manifest files. If they still fail to build, try getting an unmodified version of them off of the subversion server and “rosmake --pre-clean” them.

2. On top of the files needed for the simulator, the Powerbot uses some additional nodes which require additional launch files. Make sure you all the files you need. There may or may not be an error given for missing files as it may otherwise just use default parameters.

3. The Robot.launch file can be created following the same instructions as for the simulator, with some additional instructions given in the appendix under each file. Include or exclude files and nodes where desired. Refer to step 11 or the Troubleshooting section for additional help. It is important to note that behaviour in the simulator does not always map to behaviour on the Powerbot. Thus, you may find that you may have to set parameters differently on the Powerbot than your simulator to achieve the same behaviour.

Troubleshooting:

- **The LaserScan is not displaying in RVIZ when running the Powerbot**
  - Try running “roswtf” on the terminal of the Master computer. If it gives you and error or warning regarding Receiving out-of-date/future transforms between the Powerbot and the Master (typically regarding LMS, Failsafe, PTU Server, P2OS, and any other nod running on Powerbot), then the culprit is that the clocks (and in turn the timestamped headers) are not synchronized with that of the master. To correct this, run this command on the powerbot:
    - sudo date--set='-<time difference given by roswtf, in seconds> seconds'

- **The Pan Tilt Unit goes too slowly or does not pan or tilt as much as I want**
  - Make sure that the parameter you set for the panning or tilting is in Degrees per second and not radians.
  - If the Pan Tilt Unit goes too fast, occasionally some slipping may occur for the servo motors on which the Unit pans or tilts. In this case, the only way to correct the problem is to reset the Pan Tilt Unit so it can reset its extents

- **The Pan Tilt Unit is making loud noises or jerking violently**
  - Chances are your velocities are too large for the Pan Tilt Unit to achieve. Keep in mind the advertised prices are without a load. I found 30 degrees per second to be stable in terms of performance.

- **The Pan Tilt Unit doesn't follow it's typical up, down, side pattern and zig zags when panning and tilting**
  - This occurs when the brake node fails to acquire a PointCloud from the Assembler. This usually means that the SR4K is not connected or the Assembler settings have been modified so it does not operate the way it was intended to.

- **The Pan Tilt Unit pauses for too long after it has Panned or Tilted**
  - In order to get rid of the transient points from the data from the SR4K, the node is but to sleep for ~0.4 seconds so that the PointCloud and Transform data has and sufficient time to settle to the values we want. This can be lowered, but needs to be done manually in the code.

- **The Powerbot will not move when I give it a goal**
  - Make sure that the Powerbot can actually move. There may be situations where the inflation of the obstacles will be too large and not allow the Powerbot to go anywhere
  - Check to make sure if the Goal is valid. Errors are usually thrown out for goals which are unreachable.
  - Make sure that the centre of the Powerbot is not inside an inflated obstacle. If it is, the Powerbot assumes it has made a collision and will not move. This typically happens when you do not have sonar enabled and thus can't clear out surrounding obstacles, when the Powerbot has wandered into a densely packed area, or a multitude of other situations where the Powerbot may fail to clear surrounding obstacles.

- **When I'm running Robot.launch, or Sim.launch, the robot will stop moving and/or I get warning about failing to run a loop at a certain rate or update the map.**
  - This usually happens when you're running to many things for ROS to run on your computer at its desired rate. The best solution is to lower the update rate for whichever component (typically local or global map- update rate, or control loop) to slightly slower than the rate that it is indicated it is going. The global map update rate is in the global costmap settings, the local map update rate is in the local costmap file and the control loop update rate is a parameter in the move_base node. If you want to keep the current rate the same, you will have to lower one or both of the other two.

- **A node I downloaded will not build or gives me .so related error**
  - Often there are many junk files that are set to look for files relative to the original owners workspace set up. The easiest way to fix this problem is to do a clean build by running:
    - rosmake --pre-clean

This will get rid of any files that you do not need to build the project and will fix this issue. If the problem persists, try renaming or deleting the /build folder manually. If the problem still persists, it is best to go through the error dialogue from the terminal where you are building line by line to find the root cause of the problem.

❖ **The Powerbot is make a consistently beeping noise**

- ➢ You may wish to refer to the Powerbot manual (http://www.ing.unibs.it/~arl/docs/documentation/Aria%20documentation/Old%20and%20useless/Misc.%20Mob.%20Robots%20doc./PowerBotMan4.pdf)
- ➢ Common causes of this beeping is are low battery is the battery is running low, in which case it is best to turn it off and charge it, or if one of the big red buttons near the back has been pressed, in which case twist them in the direction indicated on the button

❖ **The Powerbot will not move, even though it has a valid path planned and is sending velocity commands and will not run via the Joystick either**

- ➢ This is likely due to the motor brake not being released. You can find the button to do this in the rear of the Powerbot, under the clear plastic cover for the pannel. The button is labeled “Motor” and is beside the “Reset” button. A good indication of whether or not the motor has been pressed is indicated by the “STATUS” LED. If it is flashing rather slowly (~1 Hz), the motor is not enabled. This test only holds true for when P2OS is running on the Powerbot as the LED will not flash rapidly otherwise unless the “GO” button is pressed on the Joystick.

❖ **How do I run the DWA Planner so as to allow me to reconfigure the parameters dynamically?**

1. Make sure you are using the dwa_local_planner in the Sim/Robot . Go into the launch files and comment out:
   - <rosparam file="$(find NavStack_config)/base_local_planner_params.yaml" command="load" />

   by wrapping it with <!-- and --> so it becomes:
   - <!--<rosparam file="$(find NavStack_config)/base_local_planner_params.yaml"

command="load" />-->

and vice versa for the dwa_local_planner_params.yaml directly below it if you have not done so already .

2. Launch the Sim/Robot
3. In a terminal type:
    - rosrun dynamic_reconfigure reconfigure_gui
4. In the Dropdown menu select /move_base/DWAPlannerROS

❖ **How do I run the swissranger so as to allow me to reconfigure the parameters dynamically?**

1. Launch swiss.launch, or another launch file you've made for the swissranger.
2. In a terminal type:
    - rosrun dynamic_reconfigure reconfigure_gui
3. In the Dropdown menu select /swissranger

❖ **When I try to run the PTU server or P2OS on the Powerbot, it gives me an error.**

➢ This is typically caused by insufficient privileges to access the RS232 or USB ports on the Powerbot. To fix this run:

- sudo chmod a+rw /dev/ttyS0

or

- sudo chmod a+rw /dev/ttyUSB0

where ttyS0 is the RS232 port for P2OS and ttyUSB0 is the USB port for the Pan Tilt Unit

❖ **When I try to launch the Pan Tilt Unit server, it says it can't find or run "ptu_action_server.py" (or any other Python file in general)**

➢ This is due to read/write permissions for the Python file. To fix this, run the following command in ptu46 directory:

- sudo chmod a+rw ptu_action_server.py

- **I can't use gedit when I ssh onto the Powerbot**
  - This can be corrected by using ssh -Y instead of ssh -X. It may still give you an error, but running it again usually works.

- **How can I check if the Powerbot has me as it's ROS Master?**
  - You can run the command:
    - echo $ROS_MASTER_URI

    To see what the current Master's IP adresss is. It is very important to insure the port used for the master URI is 11311. Run ifconfig to see your local IP address. You can test to see if it is working by running roscore on the master and rostopic list on the powerbot via ssh.

- **I'm having difficulty connecting to both the Powerbot and SR4K at the same time**
  - Check your Ipv4 address for your ethernet and wireless connections by running ifconfig. If they are both of the form 192.168.1.XXX, there lies your problem. What is happening is that the computer assumes that the Powerbot (which is connected via wireless network RAMP2) and the SR4K (which is connected to via Ethernet cable) are on the same network and can thus be connected to solely off of wireless connection or solely over your Ethernet connection. To correct this, I suggest switching the SR4Ks static IP address to the form 192.168.2.XXX (instructions to do so can be found in Dianna Yee's report) and changing any launch files to match this new IP address.

- **When I try to launch the simulator, stageros keeps crashing**
  - This is probably because either your world file(s) have been modified incorrectly or you are trying to use the sonar with the standard version of stage. In the case of the latter, install the necessary package is indicated in the instructions above.

Files:

base_local_planner_params.yaml

```
TrajectoryPlannerROS:
  max_vel_x: 0.2
  min_vel_x: 0.03
  max_rotational_vel: 0.35
  min_in_place_rotational_vel: 0.15
  acc_lim_th: 0.40
  acc_lim_x: 0.2
  escape_vel: -0.1
  vx_samples: 11
  vtheta_samples: 20
  path_distance_bias: 0.8
  goal_distance_bias: 0.8
  occdist_scale: 0.01
  heading_lookahead: 0.500
  yaw_goal_tolerance: 0.050
  xy_goal_tolerance: 0..75
  sim_time: 2.6200
  sim_granularity: 0.025
  angular_sim_granularity: 0.017453293
  oscillation_reset_dist: .05
  dwa: false
  holonomic_robot: false
```

NOTE: You only need this if you are using base local planner. This will be defaulted to if no local planner is given.

dwa_local_planner_params.yaml

```
base_local_planner: "dwa_local_planner/DWAPlannerROS"
DWAPlannerROS:
  holonomic_robot: false
  max_trans_vel: 0.2
  min_trans_vel: 0.05
  max_vel_x: 0.2
  min_vel_x: -0.05
  max_vel_y: 0
  min_vel_y: 0
  max_rot_vel: 0.35
  min_rot_vel: 0.1
  acc_lim_x: 2
  acc_lim_y: 0
  acc_lim_th: 1
```

```
sim_time: 1.5
sim_granularity: 0.025
path_distance_bias: 32
goal_distance_bias: 24
yaw_goal_tolerance: 0.05
xy_goal_tolerance: 0.05
occdist_scale: 0.01
stop_time_buffer: 0.2
oscillation_reset_dist: 0.05
forward_point_distance: 0.325
scaling_speed: 0.1
max_scaling_factor: 0.2
vx_samples: 11
vy_samples: 0
vth_samples: 20
penalize_negative_x: true
prune_plan: true
```

NOTE: You only need this if you will be using the dwa local planner.

costmap_common_params_sim.yaml

```
footprint: [[-0.450,-0.430],[-0.630,-0.160],[-0.630,0.160],[-0.450,0.430],[0.300,0.430],[0.490,0.160],[0.490,-0.160],[0.300,-0.430]]
inflation_radius: 0.75
cost_scaling_factor: 10.0
transform_tolerance: 2.0
observation_sources: laser_scan_sensor #sonar_scan_sensor
laser_scan_sensor: {sensor_frame: base_laser_link, data_type: LaserScan, topic: /scan, marking: true, clearing: true, obstacle_range: 17.9, raytrace_range: 17.9}
#sonar_scan_sensor: {sensor_frame: base_link, data_type: LaserScan, topic: /myscan, marking: false, clearing: true, obstacle_range: 5.2, raytrace_range: 5.7, observation_persistence: 0}
```

NOTE: Uncomment the sonar scan from the observation sources as well as its properties in the last line of the file if you are using it. Also, make sure the topics are correctly named if you have modified them elsewhere.

costmap_common_params_robot.yaml

```
obstacle_range: 8.0
raytrace_range: 10.0
footprint: [[-0.450,-0.430],[-0.630,-0.160],[-0.630,0.160],[-0.450,0.430],[0.300,0.430],[0.490,0.160],[0.490,-0.160],[0.300,-0.430]]
inflation_radius: 0.4
cost_scaling_factor: 10.0
transform_tolerance: 2.0
observation_sources: laser_scan_sensor #sonar_scan_sensor #point_cloud_sensor
laser_scan_sensor: {sensor_frame: LMS100, data_type: LaserScan, topic: /scan, marking: true, clearing: true, obstacle_range: 18.0, raytrace_range: 18.0}
#sonar_scan_sensor: {sensor_frame: base_link, data_type: LaserScan, topic: /myscan, marking: false, clearing: true, obstacle_range: 4.995, raytrace_range: 4.995}
#point_cloud_sensor: {sensor_frame: /camera, data_type: PointCloud, topic: /Assembled_PCL,
```

max_obstacle_height: 1.6, min_obstacle_height: 0.15, marking: true, clearing: true, obstacle_range: 9.9, raytrace_range: 9.9, observation_persistance: 0}

NOTE: To include an additional observation source, such as the SR4K or the Sonar, uncomment the lines that exclude them. If you want to use the SR4K but not the sonar, remember to reorder the observation sources to allow you to do so as the #in front of the sonar_scan_sensor will still comment out the point_cloud_sensor

global_costmap_params.yaml
```
global_costmap:
  global_frame: /map
  robot_base_frame: /base_link
  map_type: costmap
  unknown_cost_value: 1
  track_unknown_space: true
  obstacle_range: 10.0
  raytrace_range: 10.0
  update_frequency: 0.5
  publish_frequency: 0.5
  static_map: true
```

local_costmap_params.yaml
```
local_costmap:
  global_frame: odom
  map_type: costmap
  robot_base_frame: base_link
  obstacle_range: 2.0
  raytrace_range: 2.0
  update_frequency: 5.0
  publish_frequency: 1.0
  inflation_radius: 0.35
  static_map: false
  rolling_window: true
  width: 3.0
  height: 3.0
  origin_x: -1.5
  origin_y: -1.5
  resolution: 0.025
```

navfn.yaml
```
allow_unknown: true
```

display.launch
```
<launch>
	<arg name="gui" default="False" />
	<param name="robot_description" textfile="$(find <Name of Project>)/my_urdf.xml" />
	<param name="use_gui" value="$(arg gui)"/>
	<node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher" />
	<node name="robot_state_publisher" pkg="robot_state_publisher" type="state_publisher" />
</launch>
```

NOTE: This file is only needed if you are using the 3D URDF model of the Powerbot. You MUST also include my_urdf.xml (below).

<u>my_urdf.xml</u>

```
<robot name="powerbot">
 <link name="base_footprint">
        <visual>
        <origin xyz="-0.138 0 .275" rpy="0 0 0" />
                <geometry>
                        <box size="0.800 0.660 0.270"/>
                </geometry>
                <material name = "yellow">
                        <color rgba="1 1 0 1"/>
                </material>
        </visual>
 </link>
 <link name="wheel_right" >
        <visual>
                <geometry>
                        <cylinder radius="0.1315" length=".070"/>
                </geometry>
                <material name = "black">
                        <color rgba="0.2 0.2 0.2 1"/>
                </material>
        </visual>
 </link>
 <joint name="joint1" type="fixed">
        <parent link="base_footprint"/>
        <child link="wheel_right"/>
        <origin rpy="0 1.57075 1.57075" xyz= "0 0.33 0.1315" />
 </joint>
 <link name="wheel_left" >
        <visual>
                <geometry>
                        <cylinder radius="0.1315" length=".070"/>
                </geometry>
                <material name = "black"/>
        </visual>
 </link>
 <joint name="joint2" type="fixed">
        <parent link="base_footprint"/>
        <child link="wheel_left"/>
        <origin rpy="0 1.57075 1.57075" xyz= "0 -.330 0.1315"/>
 </joint>
 <link name="wheel_small_right" >
        <visual>
                <geometry>
```

```
				<cylinder radius="0.080" length=".035"/>
			</geometry>
			<material name = "black">
				<color rgba="0 0 0 1"/>
			</material>
		</visual>
	</link>
	<joint name="joint3" type="fixed">
		<parent link="base_footprint"/>
		<child link="wheel_small_right"/>
		<origin rpy="0 1.57075 1.57075" xyz= "-0.378 0.33 0.080" />
	</joint>
	<link name="wheel_small_left" >
		<visual>
			<geometry>
				<cylinder radius="0.080" length=".035"/>
			</geometry>
			<material name = "black"/>
		</visual>
	</link>
	<joint name="joint4" type="fixed">
		<parent link="base_footprint"/>
		<child link="wheel_small_left"/>
		<origin rpy="0 1.57075 1.57075" xyz= "-0.378 -.330 0.080"/>
	</joint>
	<link name="bottom_base" >
		<visual>
			<geometry>
				<box size="0.9000 .660 .070"/>
			</geometry>
			<material name = "grey">
				<color rgba="0.5 0.5 0.5 1"/>
			</material>
		</visual>
	</link>
	<joint name="joint5" type="fixed">
		<parent link="base_footprint"/>
		<child link="bottom_base"/>
		<origin rpy="0 0 0" xyz= "-0.088 0 0.105"/>
	</joint>
	<link name="top_base" >
		<visual>
			<geometry>
				<box size="0.9000 .660 .070"/>
			</geometry>
			<material name = "grey"/>
		</visual>
	</link>
	<joint name="joint6" type="fixed">
		<parent link="base_footprint"/>
```

```
        <child link="top_base"/>
        <origin rpy="0 0 0" xyz= "-0.088 0 0.445"/>
  </joint>
  <link name="laser_model" >
        <visual>
              <geometry>
                    <box size="0.100 .100 .150"/>
              </geometry>
              <material name = "blue">
                    <color rgba="0 0 1 1"/>
              </material>
        </visual>
  </link>
  <joint name="joint7" type="fixed">
        <parent link="base_footprint"/>
        <child link="laser_model"/>
        <origin rpy="0 0 0" xyz= "0.312 0 0.216"/>
  </joint>
</robot>
```

NOTE: You only nee my_urdf.xml if you will be using the 3D URDF model of the Powerbot.

Sim.launch

```
<launch>

        <master auto="start"/>
        <param name="/use_sim_time" value="true"/>
        <node pkg="stage" type="stageros" name="[Full Path to .world file]">
              <remap from="/cmd_vel" to="/stageros/cmd_vel"/>
              <remap from="/laser_scan" to="/scan"/>
              <remap from="/odom" to="/pose"/>
              <param name="base_watchdog_timeout" value="0"/>
        </node>
        <!--<node pkg="sim_to_p2os" type="sim_to_p2os" name="sim_shunt">
                    <remap from="/cmd_vel" to="/p2os/cmd_vel"/>
        </node>-->
        <node pkg="gmapping" type="slam_gmapping" args="scan:=scan" name="SLAM">
              <param name="maxUrange" value="18.0"/>
              <param name="maxRange" value="18.0" />
              <param name="xmax" value="20.0"/>
              <param name="ymax" value="20.0"/>
              <param name="xmin" value="-20.0"/>
              <param name="ymin" value="-20.0"/>
              <param name="delta" value="0.05"/>
              <param name="map_update_interval" value="0.5"/>
              <param name="linearUpdate" value="0.5" />
              <param name="angularUpdate" value="0.5" />
        </node>
        <!--<node pkg="sonar_to_laser" type="sonar_to_laser" name="SonartoLaser"/>
        <!--<node pkg="tf" type="static_transform_publisher" name="Sonar_Scan" args="-0.10 0 .45 0 0 0
```

```
/base_link /sonar 100" /> -->
        <node pkg="move_base" type="move_base" respawn="false" name="move_base"  output="screen">
                <rosparam file="$(find NavStack_config)/costmap_common_params_sim.yaml"
command="load" ns="global_costmap" />
                <rosparam file="$(find NavStack_config)/costmap_common_params_sim.yaml"
command="load" ns="local_costmap" />
                 <rosparam file="$(find NavStack_config)/local_costmap_params.yaml" command="load" />
                 <rosparam file="$(find NavStack_config)/global_costmap_params.yaml" command="load" />
                        <!--<rosparam file="$(find <Name of Project>)/base_local_planner_params.yaml"
command="load" /> -->
                <!--<rosparam file="$(find <Name of Project>)/dwa_local_planner_params.yaml"
command="load" /> -->
                <rosparam file="$(find NavStack_config)/navfn.yaml" command="load" ns="NavfnROS" />
                <param name="controller_frequency" value="5.0"/>
                <param name="planner_patience" value = "0.0" />
                <param name="recovery_behavior_enabled" value="FALSE" />
                <!--<remap from="/cmd_vel" to="/p2os/cmd_vel"/>-->
        </node>
        <include file="$(find NavStack_config)/display.launch"/>
</launch>
```

NOTE: Remember to comment and uncomment packages and nodes as needed

Robot.launch

```
<launch>
        <master auto="start"/>
        <node pkg="gmapping" type="slam_gmapping" args="scan:=scan" name="lasernode">
                <param name="maxUrange" value="18.0"/>
                <param name="maxRange" value="18.0" />
                <param name="xmax" value="20.0"/>
                <param name="ymax" value="20.0"/>
                <param name="xmin" value="-20.0"/>
                <param name="ymin" value="-20.0"/>
                <param name="delta" value="0.05"/>
                <param name="map_update_interval" value="0.5"/>
                <param name="linearUpdate" value="0.5" />
                <param name="angularUpdate" value="0.5" />
        </node>
        <!--<node pkg="sonar_to_laser" type="sonar_to_laser" name="SonartoLaser"/>
        <!--<node pkg="tf" type="static_transform_publisher" name="Sonar_Scan" args="-0.10 0 .45 0 0 0
/base_link /sonar 100" /> -->
        <node pkg="move_base" type="move_base" respawn="false" name="move_base"  output="screen">
                <rosparam file="$(find <Name of Project>)/costmap_common_params_robot.yaml"
command="load" ns="global_costmap" />
                <rosparam file="$(find <Name of Project>)/costmap_common_params_robot.yaml"
command="load" ns="local_costmap" />
                <rosparam file="$(find <Name of Project>)/local_costmap_params.yaml" command="load" />
                <rosparam file="$(find <Name of Project>)/global_costmap_params.yaml" command="load" />
                <!--<rosparam file="$(find <Name of Project>)/base_local_planner_params.yaml"
command="load" /> -->
```

```
			<!--<rosparam file="$(find <Name of Project>)/dwa_local_planner_params.yaml"
command="load" /> -->
			<rosparam file="$(find <Name of Project>)/navfn.yaml" command="load" ns="NavfnROS" />
			<param name="controller_frequency" value="5.0"/>
			<param name="planner_patience" value = "0.0" />
			<param name="recovery_behavior_enabled" value="false" />
			<remap from="/odom" to="pose" />
			<remap from="/cmd_vel" to="/p2os/cmd_vel"/>
		</node>
		<!--<include file="$(find <Name of Project>)/display.launch"/> -->
		<node pkg="tf" type="static_transform_publisher" name="base_footprint" args="0 0 0 0 0 0 /base_link
/base_footprint 100" />
		<!--<include file="$(find <Name of Project>)/brake.launch"/> -->
</launch>
```

NOTE: In order to use the Sonar, uncomment the appropriate area. Also, if you want to use the SR4K, uncomment the last line to include brake.launch and comment out the "/cmd_vel" being remapped to "/p2os/cmd_vel"

<u>brake.launch</u>

```
<launch>
		<node pkg="laser_assembler" type="point_cloud_assembler_srv" name="AssemblerServer" output
="screen">
			<param name="~fixed_frame" value ="/map"/>
			<param name="~max_scans" value ="1"/>
			<remap from="/scan_in" to="/swissranger/pointcloud_raw"/>
		</node>
		<!--<include file="$(find ptu46)/ptu46_actions.launch"/>-->
		<node pkg="Brake" type="brake" name="Brake" output="screen">
			<param name="dist_tolerance" value="2.0"/>
			<param name="theta_tolerance" value="1.04719755"/>
			<param name="pan" value="30.0"/>
			<param name="pan_range" value="180.0"/>
			<param name="pan_offset" value="0.0"/>
			<param name="tilt" value="30.0"/>
			<param name="tilt_range" value="30.0"/>
			<param name="tilt_offset" value="0.0"/>
			<param name="timer" value="30"/>
			<param name="pan_vel" value="30.0"/>
			<param name="tilt_vel" value="30.0"/>
		</node>
		<node pkg="Brake" type="ptuTF" name="ptuTF">
			<param name="basetoPTU_x" value="0.265"/>
			<param name="basetoPTU_y" value="0.0"/>
			<param name="basetoPTU_z" value="0.725"/>
		</node>
		<node pkg="Brake" type="PCL_Pub" name="PCL_Pub">
				<param name="period" value="0.1"/>
		</node>
		<include file="$(find NavStack_config)/swiss.launch"/>
```

```
</launch>
```

NOTE: In order to use the SR4K, you must include the brake launch file. If you are using the Pan Tilt Unit on your own computer and not the Powerbot, you can launch the server automatically by uncommenting the appropriate line

__**swiss.launch**__

```
<launch>
	<node pkg="swissranger_camera" type="swissranger_camera" name="swissranger" respawn="false">
		<param name="auto_exposure" value="1"/>
		<param name="integration_time" value="-1"/>
		<param name="modulation_freq" value="11"/>
		<param name="amp_threshold" value="500"/>
		<param name="ether_addr" value="192.168.2.33"/>
	</node>
	<node pkg="tf" type="static_transform_publisher" name="camera_frame_publisher" args="0.112 0
0.025 1.570796327 3.141592654 1.570796327 /PTU_Frame /camera 100"/>
</launch>
```

NOTE: The ether_addr value may be different that the one listed in this launch file. The default is 192.168.1.42 and is reverted to this if it does not connect to anything within 20 seconds from boot.

Flow Charts

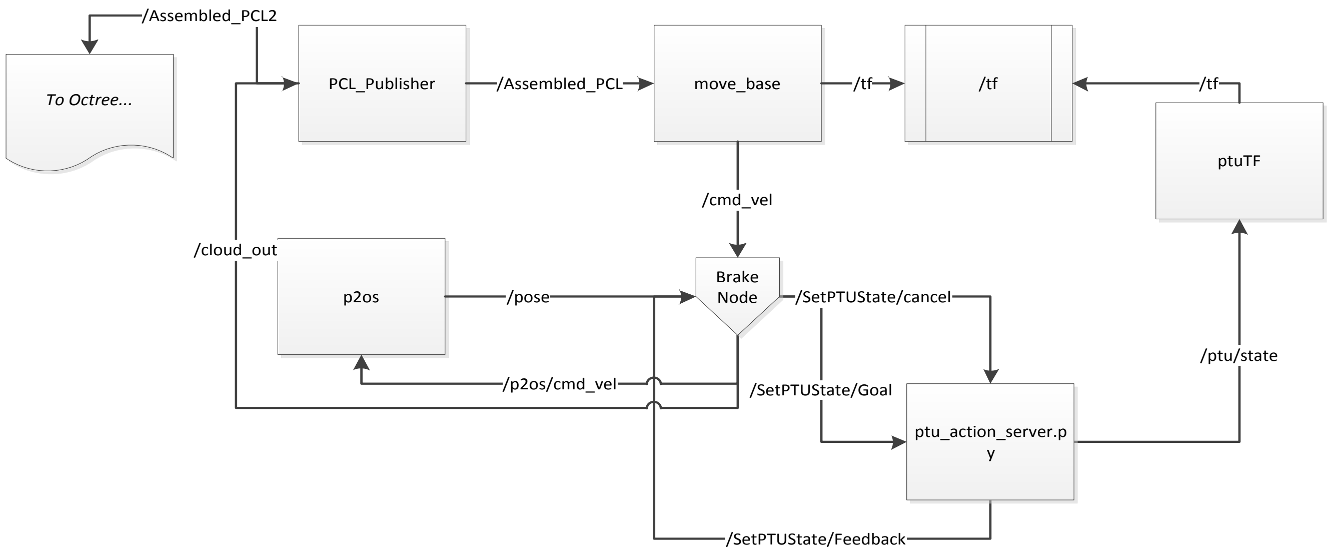

Figure 4: Overview of Brake Node

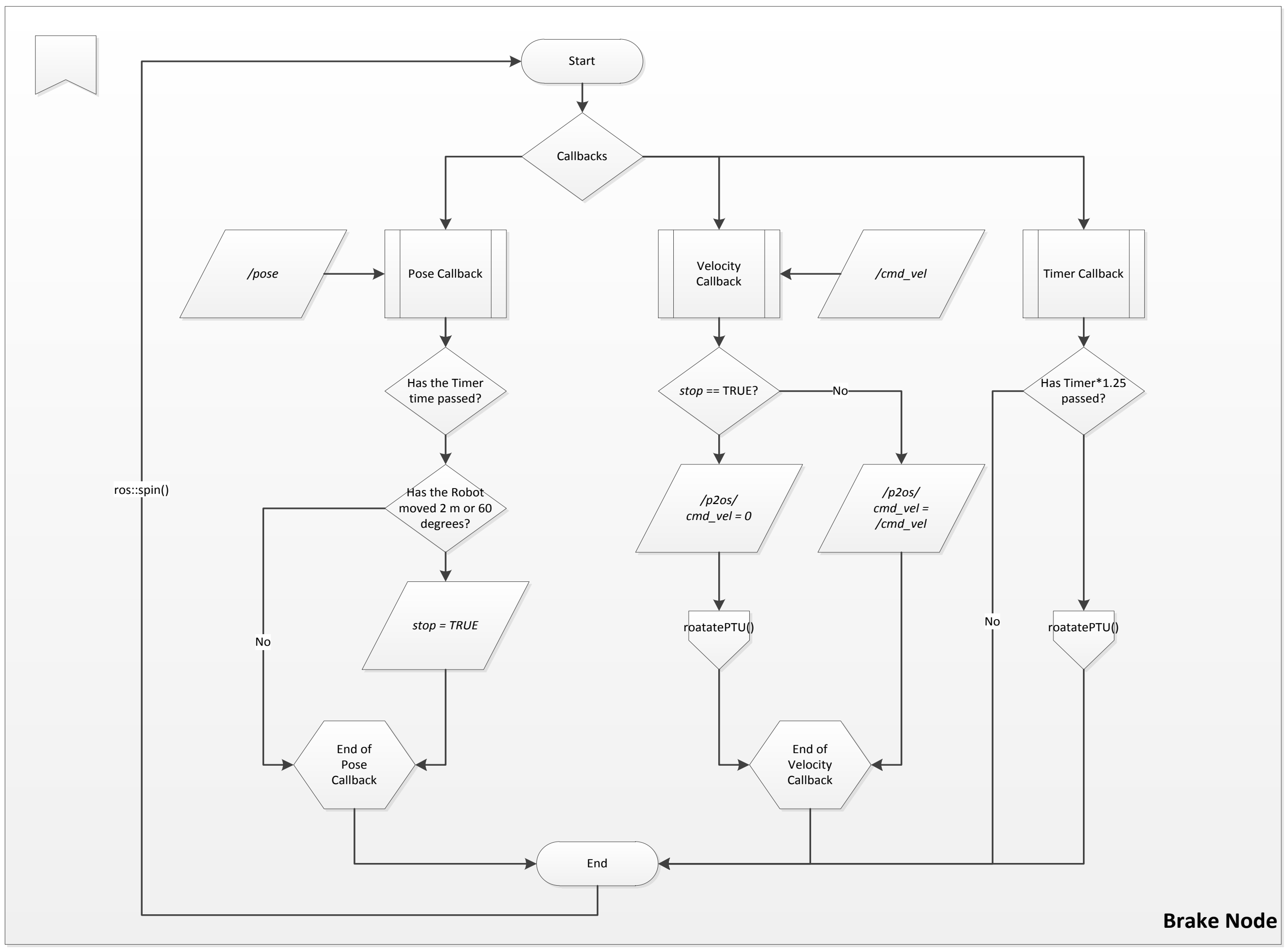

Figure 5: Brake Node

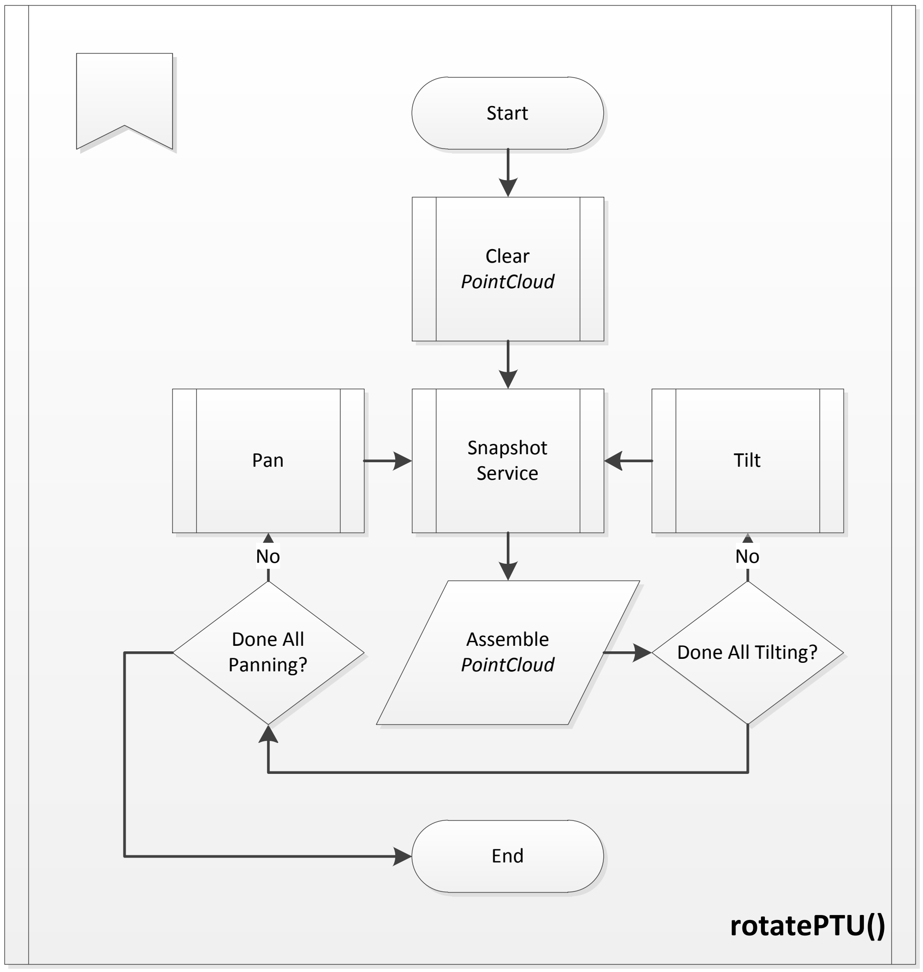

Figure 6: Rotate PTU function Flow Chart

Works Cited

Erik Karulf, D. L. (2010, March 16). *ptu46*. Retrieved September 6, 2011, from ROS.org: http://www.ros.org/wiki/ptu46

Marder-Eppstein, E. (2011, August 24). *navigation*. Retrieved September 7, 2011, from ROS.org: http://www.ros.org/wiki/navigation

Patrick Beeson, R. R. (2010, November 2). *swissranger_camera*. Retrieved September 6, 2011, from ROS.org: http://www.ros.org/wiki/swissranger_camera

Additional Sources

Brian P. Gerkey and Kurt Konolige. "Planning and Control in Unstructured Terrain ".
Discussion of the Trajectory Rollout algorithm in use on the LAGR robot.
http://pub1.willowgarage.com/apubdb_html/files_upload/8.pdf

D. Fox, W. Burgard, and S. Thrun. "The dynamic window approach to collision avoidance".
The Dynamic Window Approach to local control.
http://www.cs.washington.edu/ai/Mobile_Robotics/postscripts/colli-ieee.ps.gz

Alonzo Kelly. "An Intelligent Predictive Controller for Autonomous Vehicles".
A previous system that takes a similar approach to control.
http://www.ri.cmu.edu/pub_files/pub1/kelly_alonzo_1994_7/kelly_alonzo_1994_7.pdf

Powerbot User Manual:
http://www.ing.unibs.it/~arl/docs/documentation/Aria%20documentation/Old%20and%20useless/Misc.%20Mob.%20Robots%20doc./PowerBotMan3.pdf

SR4K User Manual:
http://www.mesa-imaging.ch/dlm.php?fname=customer/Customer_CD%2FSR4000_Manual.pdf