Simulating Mowito Rosbot Documentation

Release 0.0.1

Mowito

Jun 01, 2021

Contents

Mowito’s Navigation Stack

CHAPTER 1

About Mowito Navigation Stack

1.1 Overview

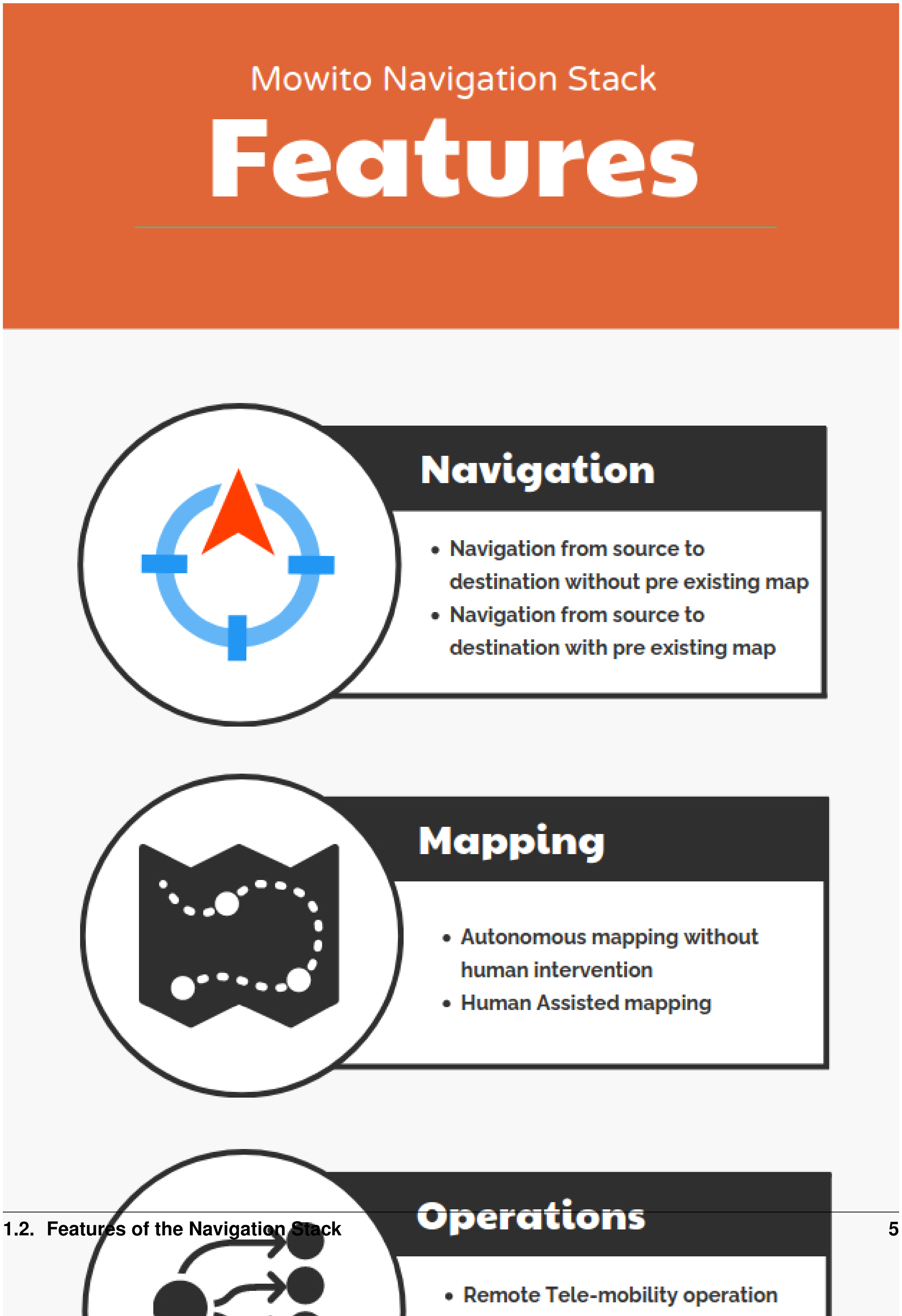
Mowito's Navigation Platform, is a software with a module dedicated for each specific task of navigation, such as planning, controlling, recovering etc.

Each module can be configured, tested and even replaced by another custom module. All the platform needs are the details about the task or location of the goal, and then based on the sensor inputs, it will drive the robot to perform the given task or reach the goal.

The critical features of the platform are its flexibility and the fast obstacle avoidance. Our controllers are optimized to detect the obstacles and correct the path at a high frequency, without waiting for the obstacle to clear the path.

You can see our controller in action on our website mowito.in

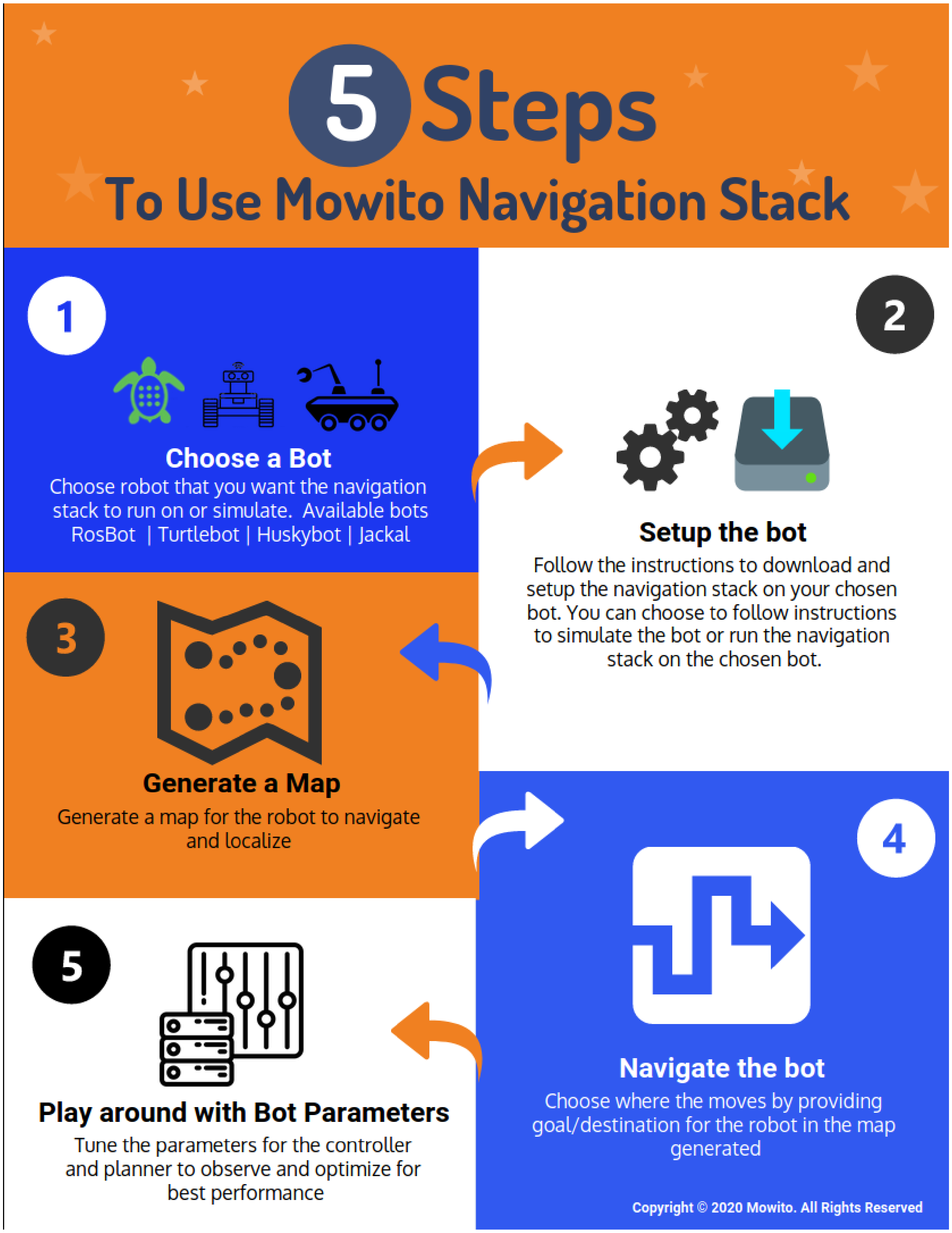
1.2 Features of the Navigation Stack


Mowito Navigation Stack
Features
Navigation
• Navigation from source to destination without pre existing map
• Navigation from source to destination with pre existing map
Mapping
• Autonomous mapping without human intervention
• Human Assisted mapping
Operations
• Remote Tele-mobility operation

CHAPTER 2

How to use Mowito Navigation Stack

The following are the steps to be followed while using the Mowito Navigation stack.


5 Steps
To Use Mowito Navigation Stack
1
Choose a Bot
Choose robot that you want the navigation stack to run on or simulate. Available bots RosBot | Turtlebot | Huskybot | Jackal
2
Setup the bot
Follow the instructions to download and setup the navigation stack on your chosen bot. You can choose to follow instructions to simulate the bot or run the navigation stack on the chosen bot.
3
Generate a Map
Generate a map for the robot to navigate and localize
4
Navigate the bot
Choose where the moves by providing goal/destination for the robot in the map generated
5
Play around with Bot Parameters
Tune the parameters for the controller and planner to observe and optimize for best performance
Copyright © 2020 Mowito. All Rights Reserved

CHAPTER 3

Installation Guide

3.1 System Requirements

3.1.1 Hardware Requirements

Processor :

Intel core i5 or higher (minimum 4 cores)

ARM v7/v8 (minimum 4 cores)

Memory :

Minimum : 4GB RAM

Recommended : 8GB RAM

Network :

WiFi 2.4/5 GHz

Sensors(Only when running on Bot Hardware) :

1. 2D/3D LiDAR
    a) For SLAM and obstacle avoidance : 30m (wh) minimum range
    b) For Obstacle Avoidance : 10m (wh) minimum range
2. Wheel encoder : 1000 pulse/rotation (minimum)
3. IMU : MPU 9250, Xsense MTi-3 AHRS, Bosch BNO055
4. GPS (When operating outdoors)

3.1.2 Software Requirements

Operating System

Ubuntu 18.04 or higher

Mandatory tools

Robot Operating System (ROS) Melodic of Noetic

3.2 Setting up Mowito Navigation Stack

3.2.1 User Registration

Register yourself on this website https://mowito.in/navigation_stack.html

We need your email to mail you the password, and to count how many people are using Mowito.

We won't spam. :)

3.2.2 Installing the Mowito on Computer (amd64 or x86)

1. Create a ROS workspace directory structure (would be useful in running simulation)

   ```
   mkdir -p ~/mowito_ws/src/
   ```

2. Clone the repo in the workspace you just created, using

   ```
   cd ~mowito_ws/src/
   ```

   for Ubuntu 18 - ROS Melodic

   ```
   git clone -b melodic https://github.com/mowito/mowito_amd64.git
   ```

3. Remove any previous installation of Mowito stack

   ```
   cd mowito_amd64
   ```

   for Ubuntu 18 - ROS Melodic

   ```
   ./remove_mowito.sh melodic
   ```

4. Install the new Mowito stack

   for Ubuntu 18 - ROS Melodic

   ```
   ./setup_mowito.sh melodic
   ```

3.2.3 Installing the Mowito on the Robot -ROSbot (arm64, armhf)

Checkout *installation instructions for ROSbot*. You can try out same steps on the turtlebot and other robots as well.

CHAPTER 4

Step 1 : Choosing a Bot

4.1 For Simulation

Mowito provide packages for the simulation of following robots:

1) ROSbot (Melodic and Noetic)
2) TurtleBot (Melodic)
3) Husky (Melodic)
4) Jackal (Melodic)

4.2 For Real Robot testing

Although Mowito's navigaion stack can work on multiple kind of wheeled robots, we currently provide documentation for the following robots:

1) *ROSbot*

You can adapt the steps and launch files for your own robot or contact Mowito (puru@mowito.in) to create custom launch files for your robot.

CHAPTER 5

Step 2 : Setup the Bot

This section shall provide the instructions to setup the stack on the chosen Bot.

5.1 Setup for Simulation

There are 3 steps to setup the chosen bot for simulation purposes

5.1.1 Step 1 : Cloning the chosen bot in the mowito_ws (the one you setup during installation)

For ROSBot, run the following command

```
cd ~/mowito_ws/src/ && git clone https://github.com/mowito/
mowito_rosbot.git     cd ~/mowito_ws/src/ && git clone https://github.
com/mowito/mw_mprims.git
```

For TurtleBot, run the following command

```
cd ~/mowito_ws/src/ && git clone https://github.com/mowito/
mowito_turtlebot.git
```

For Husky, run the following command

```
cd ~/mowito_ws/src/ && git clone https://github.com/mowito/
mowito_husky.git
```

Note: To use velodyne and slam toolbox with husky, switch to the branch **velodyne_with_husky**

```
cd ~/mowito_ws/src/mowito_husky && git checkout
velodyne_with_husky
```

For JackalBot, run the following command

```
cd ~/mowito_ws/src/ && git clone https://github.com/mowito/
mowito_jackal.git
```

Step 2 : Install the dependencies

```
cd ~/mowito_ws/ && rosdep install --from-paths src --ignore-src -r -y
```

Step 3 : Build the workspace

```
catkin_make
```

Step 4 : FOR TURTLEBOT ONLY

Run the following commands

```
source <path_to_mowito_ws>/devel/setup.bash
```

```
export TURTLEBOT3_MODEL=waffle_pi
```

CHAPTER 6

Step 3 : Generate a Map

The Mowito Navigation Stack provides three methods to generate a map.

6.1 Mapping for Simulation Pursose

6.1.1 Method 1 : Manual Map generation via remote control robot exploration

Step 0 : Source the workspace

```
source <path_to_mowito_ws>/devel/setup.bash
```

Step 1 : Launch the sim_mw_mapping node

For ROSBot, run the following command

```
roslaunch mowito_rosbot sim_mw_mapping.launch
```

For TurtleBot, run the following commands

```
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

In another terminal, run the following commands:

```
export TURTLEBOT3_MODEL=waffle_pi
```

```
roslaunch mowito_turtlebot turtle_mowito_mapping.launch
```

For Husky, run the following command

```
roslaunch mowito_husky sim_mw_mapping.launch
```

For Jackal, run the following command

```
roslaunch mowito_jackal jackal_mw_mapping.launch
```

Step 2 : Launch the remote control for providing commands to the bot

in another terminal, run the following command :

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

6.1.2 Method 2 : Map generation by providing goal destination for navigating robot for exploration

Here, the robot will explore the map based on the goal destination provided by the user on RViz.

For ROSBot, run the following command

```
roslaunch mowito_rosbot sim_mw_navigation_with_no_map.launch
```

For TurtleBot, run the following command

```
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

In another terminal, run the following commands:

```
export TURTLEBOT3_MODEL=waffle_pi
```

```
roslaunch mowito_turtlebot turtle_mowito_nav_no_map.launch
```

For Husky, run the following command

```
roslaunch mowito_husky sim_mw_navigation_with_no_map.launch
```

For using cartographer for mapping/ SLAM instead of default mw_mapping, use the following commad:

```
roslaunch mowito_husky sim_mw_navigation_with_no_map.launch
cartographer:=true
```

For using slam toolbox for mapping/ SLAM with velodyne, use the following commad:

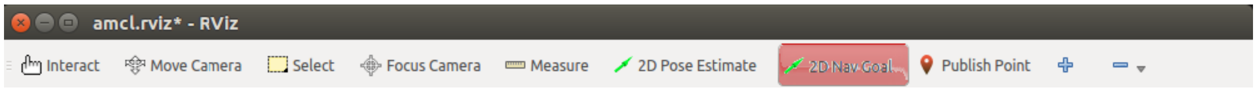
```
roslaunch mowito_husky sim_mw_navigation_with_no_map_slam_toolbox.
launch
```

For Jackal, run the following command

```
roslaunch mowito_jackal jackal_mw_nav_no_map.launch
```

The goal can be provided on RViz using the “2D Nav Goal” feature provided on RViz.

The icon is highlighted in red in the image below.



6.1.3 Saving the Map

Once you are done creating the map on rviz, save the map on a new terminal exeute the following:

```
cd && rosrun map_server map_saver -f mymap
```

the map (pgm and yaml) is saved in the home directory with the name mymap.pgm and mymap.yaml

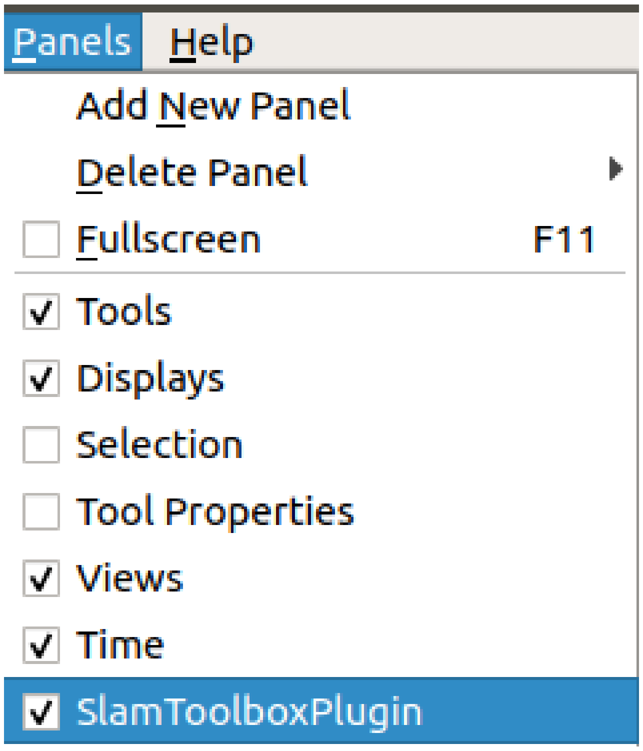
For Huskybot

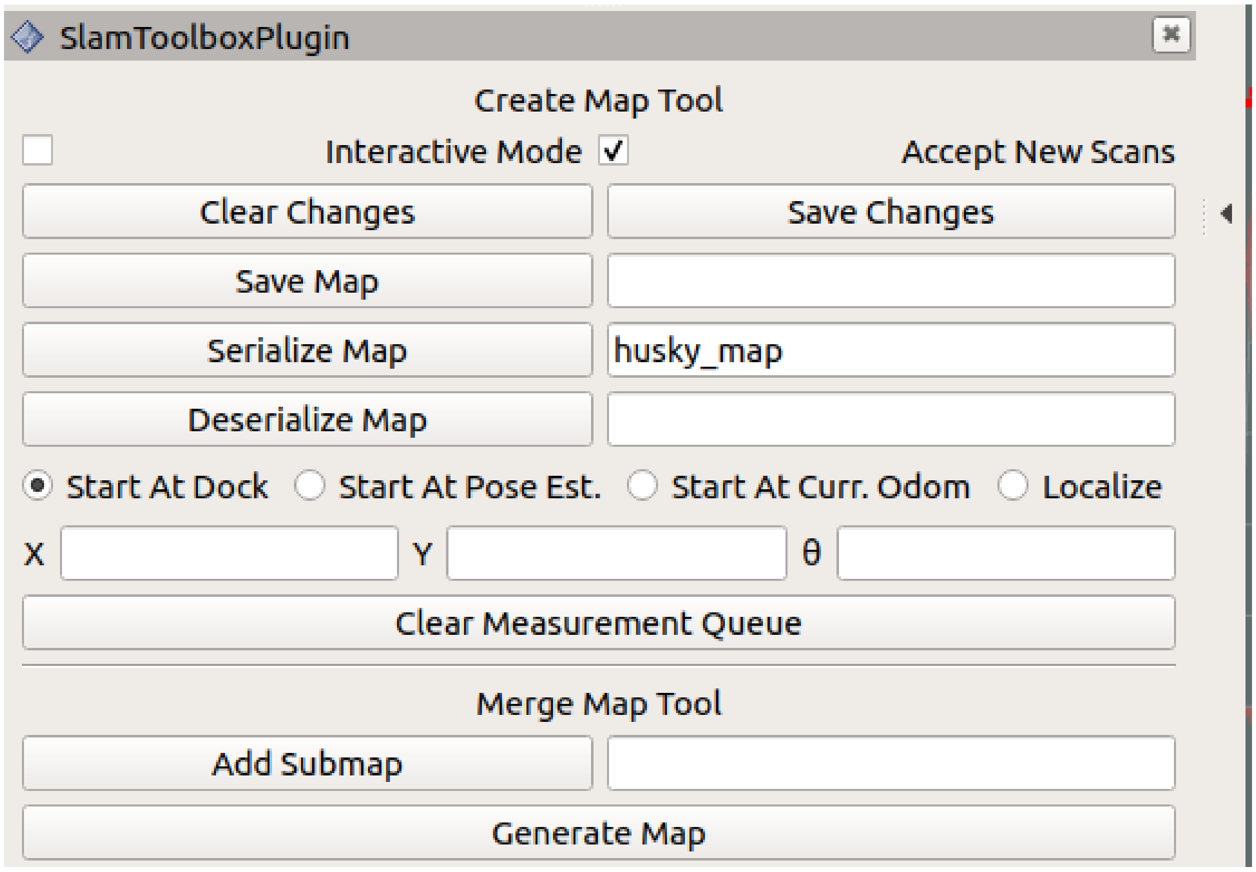
1) if you were using cartographer to build the map , run the following command

   ```
   rosrun mowito_husky save_carto_map.sh map_name
   ```

   the map (pbstream) is saved in the home directory with the name map_name.pbstream. If no map_name is given then it would save as map.pbstream

2) if you were using slam toolbox to build the map, open the slam toolbox plugin in Rviz by clicking the panels and give a name for the map and store it using serialize map option.





the map is saved in the .ros folder in the home directory with the name husky_map.posegraph and husky_map.data.

Alternatively, in order to save the map, on a new terminal execute the following:

```
rosservice call /slam_toolbox/serialize_map "husky_serialize"
```

CHAPTER 7

Step 4 : Navigate the Chosen Bot

This section shall provide instructions on how to navigate the chosen Bot.

7.1 Navigation for Simulation Purpose

Step 0 : Source the workspace

```
source <path_to_mowito_ws>/devel/setup.bash
```

Step 1 : Place the robot at the origin of map (the place where you started mapping)

Step 2 : For running the entire system with mowito's controller, run the following command

For ROSBot, run the following command

```
roslaunch mowito_rosbot sim_mw_navigation.launch
```

If you want to use the map created in the previous section use the following command

```
roslaunch mowito_rosbot sim_mw_navigation.launch map_name:=mymap
```

For TurtleBot, run the following command

```
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

In another terminal, run the following commands:

```
export TURTLEBOT3_MODEL=waffle_pi
```

```
roslaunch mowito_turtlebot turtle_mowito_nav_map.launch
```

If you want to use the map created in the previous section use the following command

```
roslaunch mowito_turtlebot turtle_mowito_nav_map.launch
map_name:=mymap
```

For HuskyBot, run the following command

```
roslaunch mowito_husky sim_mw_navigation.launch
```

If you want to use the map created in the previous section use the following command

```
roslaunch mowito_husky sim_mw_navigation.launch map_name:=mymap
```

For using cartographer for mapping/ SLAM instead of default mw_mapping, use the following command:

```
roslaunch mowito_husky sim_mw_navigation.launch cartographer:=true
```

For using slam toolbox for mapping/ SLAM with velodyne, use the following commands:

1. move the map data to .ros folder in your system by running the following two commands:

   ```
   cd <path_to_mowito_ws>/src/mowito_husky/husky/mowito_husky/maps/
   ```

   ```
   cp husky_serialize.data husky_serialize.posegraph ~/.ros/
   ```

2. set the name of the map file and map start pose [x,y,theta] in mowito_ws/src/mowito_husky/husky/mowito_husky/config/slam_toolbox_config/slam_toolbox_localization.yaml:

   ```
   map_file_name:  husky_serialize
   ```

   ```
   map_start_pose:  [0.0, 0.0, 0.0]
   ```

3. run slam toolbox for mapping/ SLAM with velodyne:

   ```
   roslaunch mowito_husky sim_mw_navigation_slam_toolbox.launch
   ```

For Jackal, run the following command

```
roslaunch mowito_jackal jackal_mw_nav.launch
```

Here, “mymap” is the map that was generated in the earlier step (Step 3 : Generate the map)

Step 3 : in the rviz, click on the second top panel, click on the nav goal option, and click on the displayed map to give goal to the robot

Step 4 : look at the output on the rviz, the path planned and the motion of the robot

7.2 Route Based Navigation

7.2.1 Overview

- In addition to the waypoint navigation feature in simulation, one can also simulate the robot by giving route points via route.yaml (type) file.
- This feature provides user the ability to give pre-planned goals.
- There are two ways for using this feature. We will be using the example of husky robot simulation to explain this:

Note: While using this feature, it is highly recommended that one uses the **genroute** planner for optimal results.

There are two ways to go about using Routes, based on when the user wants to change the planner types.

A. Selecting the type of (Global) Planner before launching the stack

1. Open the mission_executive_params.yaml file located inside the *mowito_husky/husky/mowito_husky/config/mission_executive_config* folder of the mowito_husky package.

2. Change the planner tag to `genroute`.
3. Now, lets run navigation with map:

   For ROSBot run the following command

   ```
   roslaunch mowito_rosbot sim_mw_navigation.launch
   ```

   For TurtleBot run the following commands

   ```
   roslaunch turtlebot3_gazebo turtlebot3_world.launch
   ```

   In another terminal, run the following commands:

   ```
   export TURTLEBOT3_MODEL=waffle_pi
   ```

   ```
   roslaunch mowito_turtlebot turtle_mowito_nav_map.launch
   ```

   For Husky run the following command

   ```
   roslaunch mowito_husky sim_mw_navigation.launch
   ```

   For Jackal run the following command

   ```
   roslaunch mowito_jackal jackal_mw_nav.launch
   ```

4. In a new terminal, run the set_route_client node with the appropriate file path to the route.yaml (type) file. Sample route files are available in the samples folder of the mowito_husky package:

   ```
   rosrun executive set_route_client path/to/route/file
   ```

B. Changing the (Global) Planner during the run (after launching the stack)

1. Lets run navigation with map:

   For ROSBot run the following command

   ```
   roslaunch mowito_rosbot sim_mw_navigation.launch
   ```

   For TurtleBot run the following command

   ```
   roslaunch turtlebot3_gazebo turtlebot3_world.launch
   ```

   In another terminal, run the following commands:

   ```
   export TURTLEBOT3_MODEL=waffle_pi
   ```

   ```
   roslaunch mowito_turtlebot turtle_mowito_nav_map.launch
   ```

   For Husky run the following command

   ```
   roslaunch mowito_husky sim_mw_navigation.launch
   ```

   For Jackal run the following command

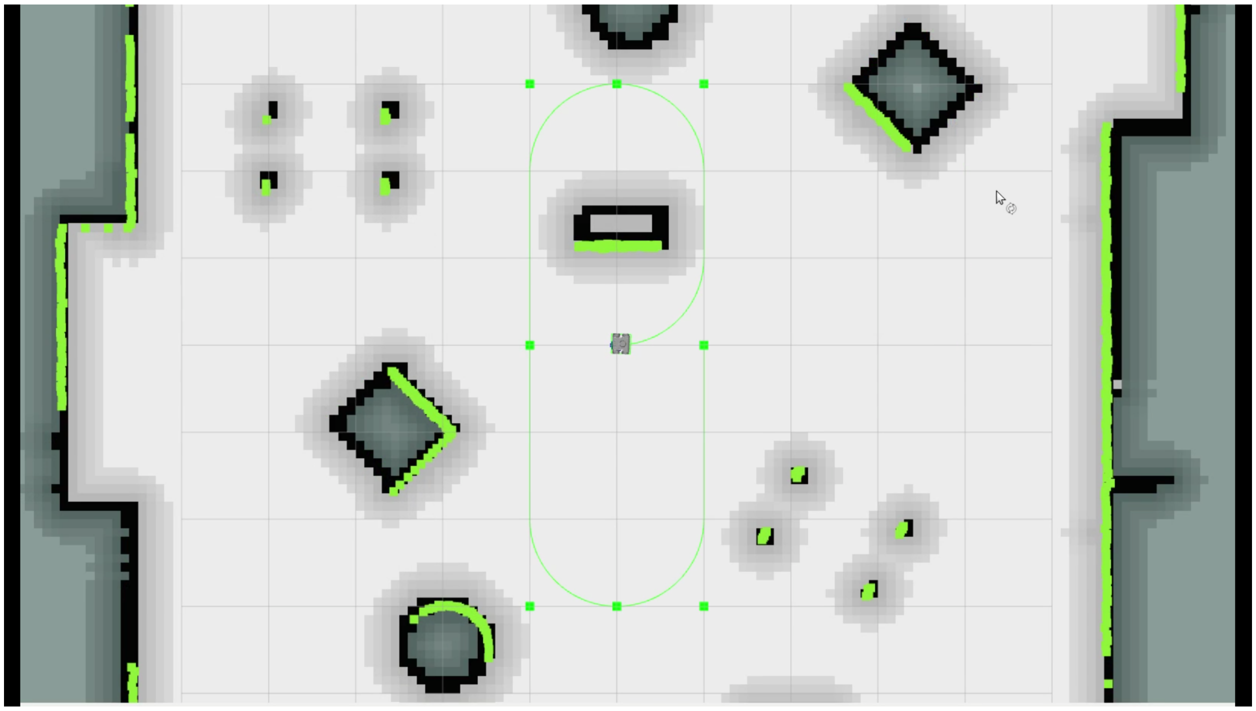
   ```
   roslaunch mowito_jackal jackal_mw_nav.launch
   ```

2. Use the change_planner and change_controller services to change the planner and controller respectively. For this, in a new terminal, execute:

   ```
   rosservice call /mission_executive/change_planner genroute
   ```

4. Now, in a new terminal, run the set_route_client node with the appropriate file path to the route.yaml (type) file. Sample route files are available in the samples folder of the mowito_husky package:

   ```
   rosrun executive set_route_client path/to/route/file
   ```



Example: rosbot following a given route

CHAPTER 8

Step 5 : Configuring Bot Parameters

One of the good things about Mowito's Navigation stack is that you can easily configure it for different situations. You can find the configuration files in the config folder of the mowito packages for the respective robots:

1. ROSbot - `mowito_ws/src/mowito_rosbot/config`
2. Turtlebot - `mowito_ws/src/mowito_turtlebot/mowito_turtlebot/config`
3. Husky - `mowito_ws/src/husky/mowito_husky/config`
4. Jackal - `mwowito_ws/src/mowito_jackal/mowito_jackal/config`

The following pages will get into more details about different config files, and how can they be used for your purpose.

We will be focussing on configuring:

1. Controller - :ref: *MaxL Controller<maxl_planner>*
2. *Mission Executive*
3. Costmap

CHAPTER 9

Config : Controller - MaxL

9.1 Overview

MaxL Planner is a package that is used to drive the robot. It issues the linear and angular velocity commands that are needed to reach the goal.

9.2 Robot Parameter Description

9.2.1 1. Robot Configuration Parameters

Parameter	Units	Description
use_laser	true/false	If true, the robot uses the rpLidar Sensor otherwise uses velodyn Sensor for planning
pathFolder	File path	The relative path to the path folder
pathFile	String	The name of the path
autonomy-Mode	true/false	If true, calculates the relative goal for the robot to follow

9.2.2 2. Linear speed and acceleration

Parameter	Units	Description
maxSpeed	S.I (m/s)	Maximum possible linear velocity
maxAccel	S.I (m/s^2)	Maximum possible linear acceleration

9.2.3 3. Turning Parameters

Parameter	Units	Description
yaw_gain	(Numeric) eg.2.5	Yaw gain used when robot is in motion
stop_yaw_gain	(Numeric) eg. 0.6	Yaw gain used when robot is stopped/almost stopped
max_yaw_rate	S.I (rad/s)	Maximum angular velocity for the robot

9.2.4 4. Inflation

Parameter	Units	Description
x_inflate	S.I (m)	Obstacle Inflation in the x direction
y_inflate	S.I (m)	Obstacle Inflation in the y direction

9.2.5 5. Frame Names

Parameter	Units	Description
map_frame	String	Name of the map frame
robot_frame	String	Name of the robot base frame
velodyne_frame	String	Name of the velodyn Sensor frame
laser_frame	String	Name of the rpLidar Sensor frame

9.2.6 6. Topic Names

Parameter	Units	Description
odomTopic	String	The topic name which publishes the odometry
velodyneTopic	String	The topic name which publishes the velodyn sensor data
scanTopic	String	The topic name which publishes the rpLidar sensor data

9.2.7 7. Robot Footprint

Parameter	Units	Description
vehicleLength	S.I (m)	Length of the vehicle
vehicleWidth	S.I (m)	Width of the vehicle

9.2.8 8. Obstacle Ranges

Parameter	Units	Description
obstacle_horizon	S.I (m)	Parameter used for cropping the pointcloud
min_path_range	S.I (m)	Minimum path range for finding the path
initial_path_scale	(Numeric) eg. 1.0	Initial path scale value. Path Scales scale the paths and distances. Low pathScale means path elongation and vice-versa.
min_path_scale	(Numeric) eg. 0.75	Minimum path scale value. For particular local goal, pathScale starts with initial value, finds a path, then value of path scale is decreased to find a longer solution path, till it hits the minPathScale.
path_scale_step	(Numeric) eg. 0.25	Path Scale step value

9.2.9 9. Pure Pursuit Parameters

Parameter	Units	Description
min_lookahead	S.I (m)	The minimum lookahead on the global path for the robot
max_lookahead	S.I (m)	The minimum lookahead on the global path for the robot
closest_point_index_search	(Numeric) eg. 10	Search for closest point index within this range of previous closest point
min_radius	S.I (m)	Minimum radius the robot can take from current to goal pose
max_radius	S.I (m)	Maximum radius the robot can take from current to goal pose
max_omega_radius	S.I (m)	Radius set when condition for straight line is satisfied
max_y_deviation	S.I (m)	Maximum deviation in the lateral direction
lookahead_point_distance	S.I (m)	Used to find the point in the global path to follow
lookahead_factor_val	(Numeric) eg. 0.088	Controls the senstivity of movement of lookahead goal. Lower the value lower the change in the postion of lookahead goal.
lookahead_jump_threshold	S.I (m)	If the change in the position of lookahead goal is greater than this value, it would be considered a jump (oscillation)

9.2.10 10. MaxL Miscellaneous Parameters

Parameter	Units	Description
direction_threshold	(degrees) eg. 120	The fan size (in degrees) on either side of robot wrt relative goal
high_accuracy_multiplier	(Numeric) eg. 0.4	High accuracy multiplier for reaching the goal (0,1]
vis_pointcloud	true/false	Parameter to enable visualisation of detailed data (pointcloud data)
use_odom_velocity	true/false	Parameter to take velocity from odom messages
reverse_enabled	true/false	Parameter to enable reverse motion for the robot
truncated_fan_angle	(degrees) eg. 10	The fan size (in degrees) on either side of robot wrt relative goal when there is no obstacle detected by the robot

9.2.11 11. Parameters for Oscillation Detection by Path Index

Parameter	Units	Description
pi_osc_senstivity	(Numeric) eg. 5	Controls the senstivity of jump detection. If this value is high, even small changes in the value of selected path index are considered an oscillation and vice-versa
pi_osc_threshold	(Numeric) eg. 10	Everytime an oscillation is detected, a count is increased. If this count goes above this threshold, oscillations are considered true and not just an error in detection
osc_det_by_score_path	true/false	Flag to switch on/off the critic/method of oscillation detectino by path index. If false, the above mentioned params would be rendered ineffective.

9.2.12 12. Parameters for Oscillation Detection by Angular Velocity

Parameter	Units	Description
av_osc_sample_window	S.I.(s)	Time period/window over which frequency of oscillation is calculated
av_osc_freq_threshold	(Numeric) eg. 3.5	If the frequency of change in angular velocity direction per av_osc_sample_window is more than this value, it is considered an oscillation
osc_det_by_ang_vel	true/false	A flag which gives user the choice to use this method of oscillation detection. If false, oscillation detection by this method will stop

9.2.13 13. Scoring Parameters

Parameter	Units	Description
scoring_algo_index	(Numeric) eg. 1	This parameter decides which scoring algorithm will be used to score paths. Currently, we have 4 different scoring algoritms to chose from
scoring_algo_four_sensitivity_factor	(Numeric) eg. 0	This parameter is used only by scoring algo number four. It controls the amount of time for which oscillation mitiagtion will last. The larger the value, the longer the oscillation mitigation will work to remove oscillation
in_place_rotation_penalty	(Numeric) eg. 0.05	Higher value penalises in place rotation more
goal_direction_preference	(Numeric) eg. 0.2	Higher value means controller prefers paths oriented towards the goal

CHAPTER 10

MaxL Controller Tuning Guide

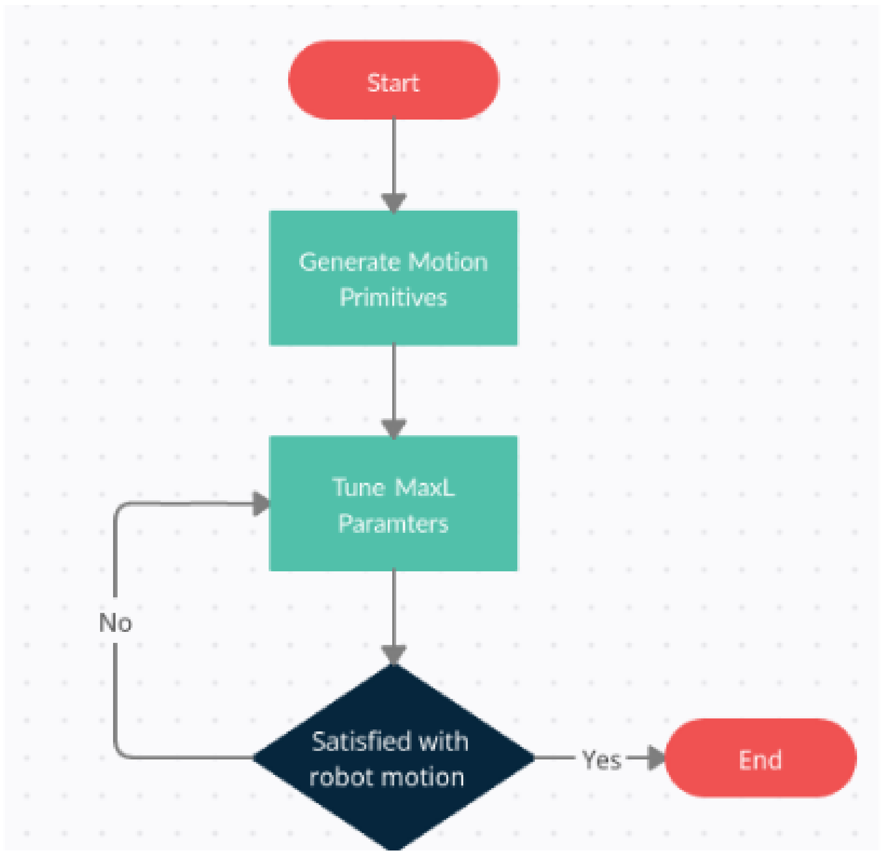
This is a guide that will describe the steps to tune the MaxL controller for any deployment of the Mowito Navigation Stack. This guide shall provide all the parameters that are required to be tuned, their significance and description of what the parameters mean.

This guide is typically meant for the end users who will be using the Mowito Navigation stack and have deployed the navigation stack on their respective hardware. This guide will only address the controller and obstacle avoidance functionality of the Mowito Navigation Stack.

10.1 Steps to tuning the MaxL Controller

The MaxL controller is a proprietary state of the art control and obstacle avoidance system that has the ability to process information and control the robot and avoid obstacles with a refresh rate as high as 50 Hz. In order to use the MaxL controller provided in the Mowito Navigation Stack, the controller is required to be tuned.

The following flow chart shall highlight the steps to follow for tuning the MaxL controller.



The process of tuning the controller involves two major steps :

10.2 Step 1 : Generating Motion Primitives for the Controller

The Motion Primitives are a set of precomputed paths that the robot can take while the robot is in motion. Whenever an obstacle confronts the robot, some of the precomputed paths are blocked and the controller chooses a path from the set of paths that are not blocked.

While tuning the controller generation of these precomputed paths is a mandatory first step. To generate the motion primitives, the following information is required :

1. Robot Length
2. Robot Width

The motion primitives would be generated using a tool provided by Mowito.

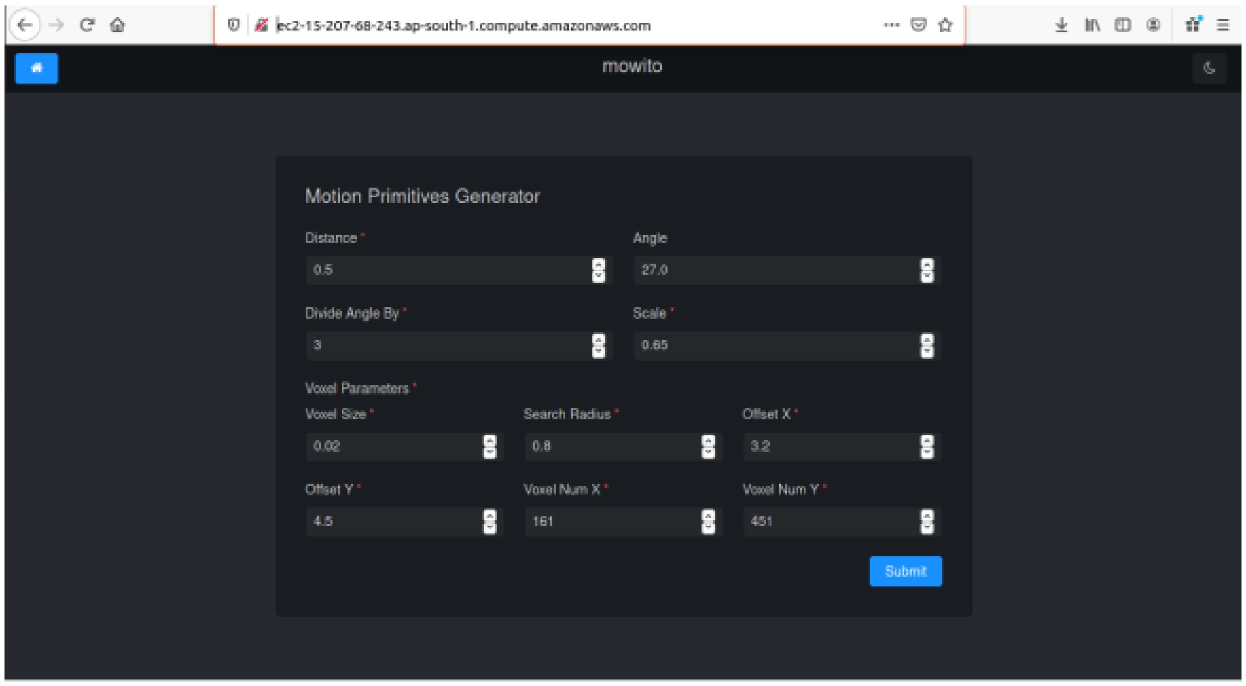
Here are the steps to generate the motion primitives:

10.2.1 Accessing the motion primitives generator web tool

The motion primitives are generated using a web tool developed by Mowito. So inoder to generate the motion primitives, the user must access the web tool.

Here is the link to the web tool

Upon accessing the web tool, the user will land onto the following page :



Motion Primitives Generator

Distance * 0.5

Angle 27.0

Divide Angle By * 3

Scale * 0.65

Voxel Parameters *

Voxel Size * 0.02

Search Radius * 0.8

Offset X * 3.2

Offset Y * 4.5

Voxel Num X * 161

Voxel Num Y * 451

10.2.2 Setting the motion primitive parameters to generate the motion primitives

Inorder to generate the motion primitives, certain parameters are required to be set. The parameters that are required to be set by the user are :

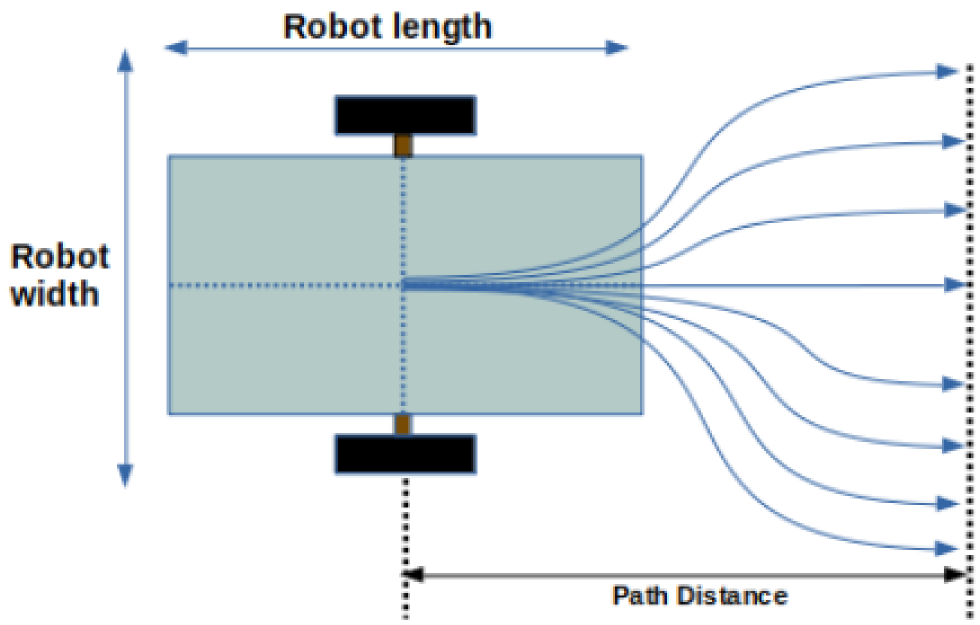
1. Path Distance or simply Distance
2. Search Radius

The aforementioned parameters are the **ONLY TWO PARAMETERS** that the **USER MUST SET**. Tampering any other parameter shall generate wrong motion primitives.

The details of the two parameters are as follows :

1. Path Distance

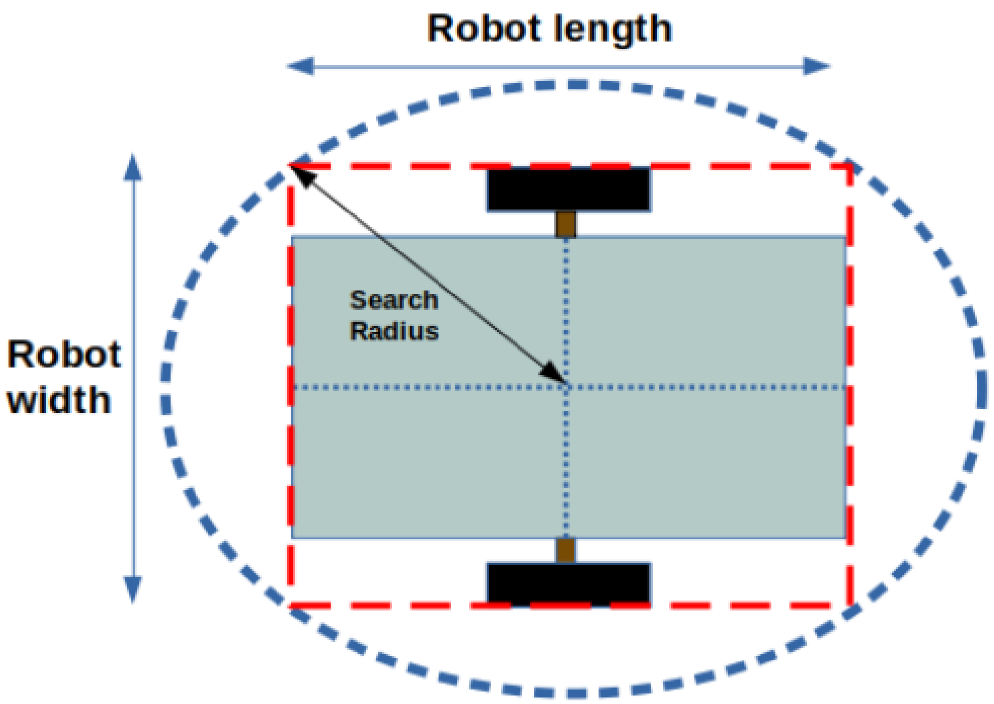
The distance basically indicates the length of the motion primitives from the center of the robot. The following diagram gives an illustration of the path distance.



The path distance value shall remain within the following bounds : Minimum path distance : (Robot Length)/2 Maximum path distance : obstacle horizon distance (shall be explained in section 4)

2. Search radius

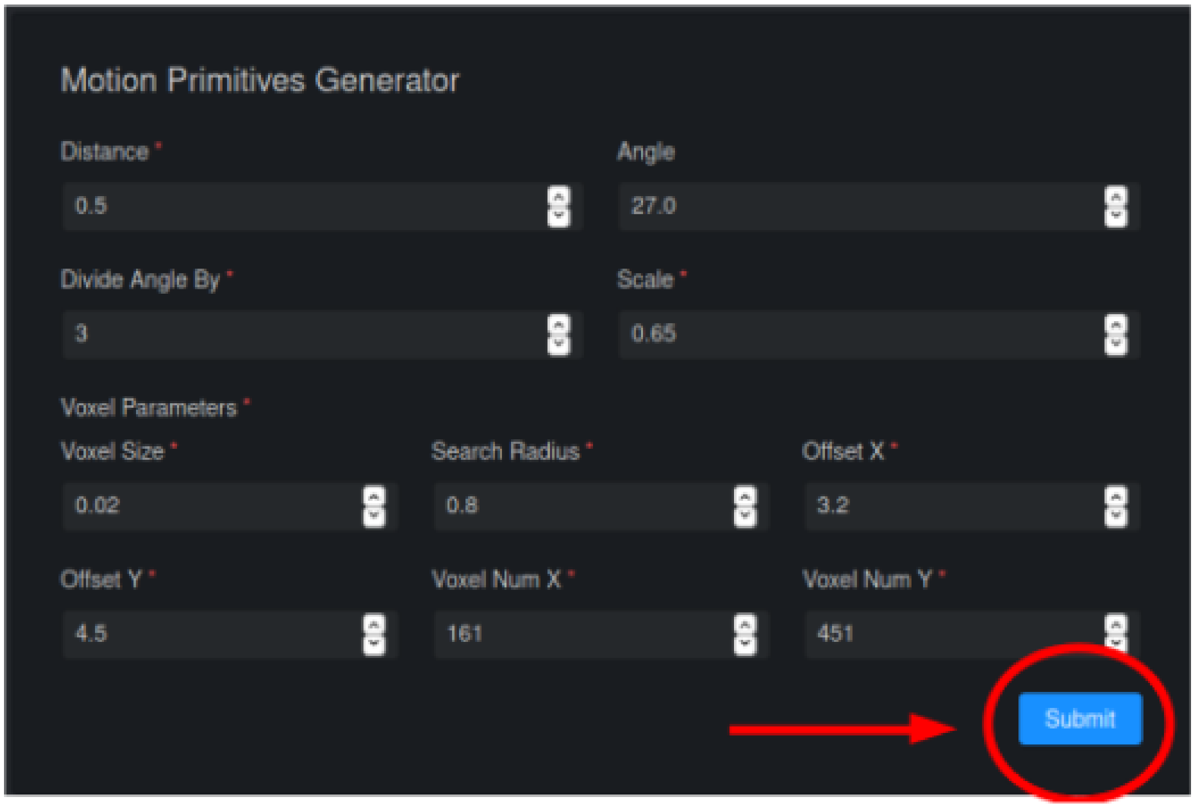
The search radius for the motion primitives shall be set a value equal to the radius of the circle that encircles the robot. The search radius parameter is illustrated in the following diagram.



Basically a higher search radius will provide a greater safety shield around the robot while the algorithm selects a path. However, a higher search radius will also lead to lesser free paths being available when the robot is confronted by an obstacle.

Thus it would be wise and apt to set the search radius to a value = radius of the circle encircling the robot.

10.2.3 Hit the Submit button

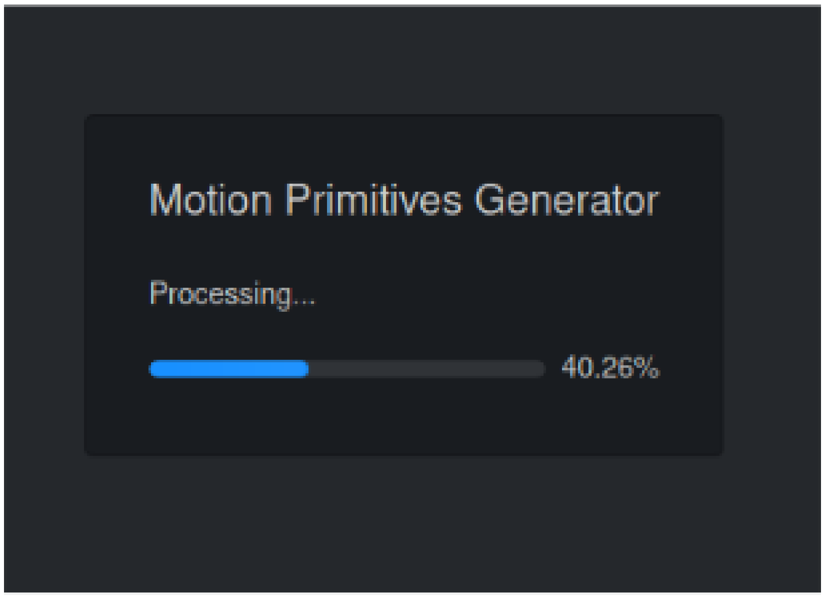

Motion Primitives Generator

Distance *	0.5
Angle	27.0
Divide Angle By *	3
Scale *	0.65

Voxel Parameters *

Voxel Size *	0.02
Search Radius *	0.8
Offset X *	3.2
Offset Y *	4.5
Voxel Num X *	161
Voxel Num Y *	451

10.2.4 The motion primitives will begin generation and a progress bar is displayed to track it

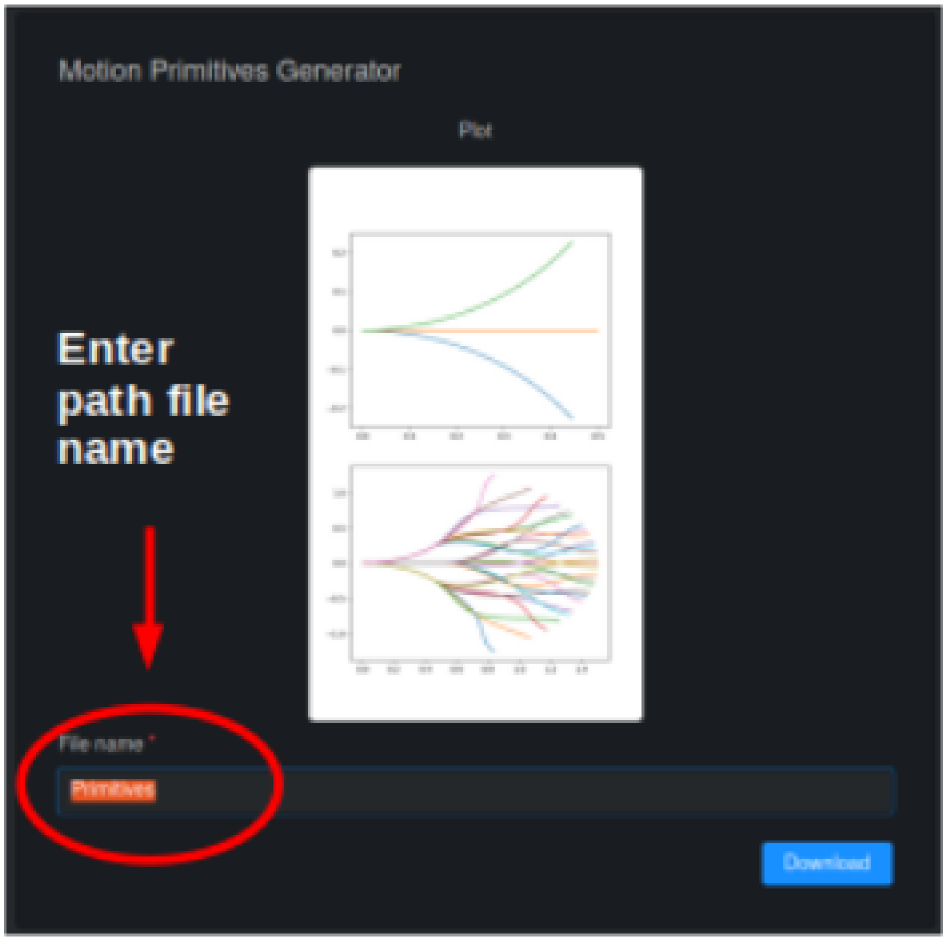

Motion Primitives Generator

Processing...

40.26%

10.2.5 After completion

the web tool will display the motion primitives and will display the paths generated. Further the tool will prompt the user to enter the name for the paths that are generated



A general convention to name the motion primitive file is given below

mw_mprim_dxdd_rxrr

d = path distance r = search radius

For example, the naming of the path file for motion primitives with path distance = 1.2 m and search radius = 0.55 m would be as follows :

mw_mprim_1x20_0x55

Another example, the naming of the path file for motion primitives with path distance = 0.75 m and search radius = 0.65 m would be as follows :

mw_mprim_0x75_0x65

10.2.6 Hit the download button

Uncompress the downloaded folder and place it in the active working directory in your robot workspace.

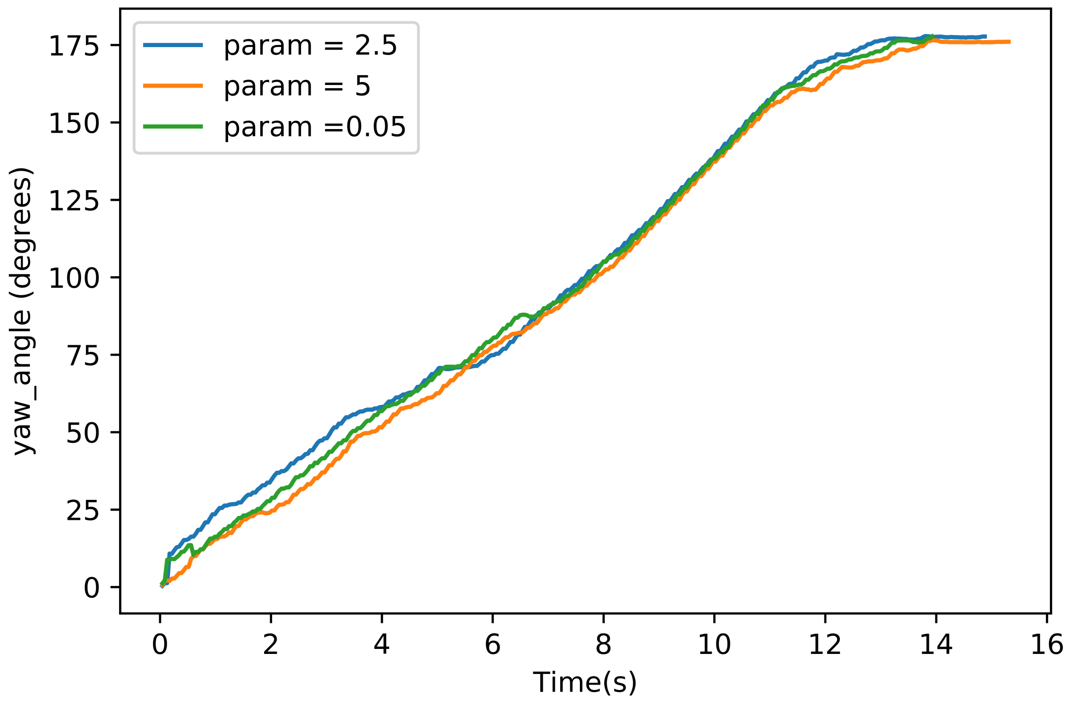
10.3 Step 2 : Configuring the MaxL parameters

The MaxL parameters are the parameters that help the algorithm decide what path to select during the robot motion when confronted by an obstacle and otherwise. There are four categories of MaxL parameters that the users can configure based on various condition.

The parameters can be editted using the mw_maxl_planner.yml file which is located in the controller_config folder.

10.3.1 Parameters influneced by Bot architecture:

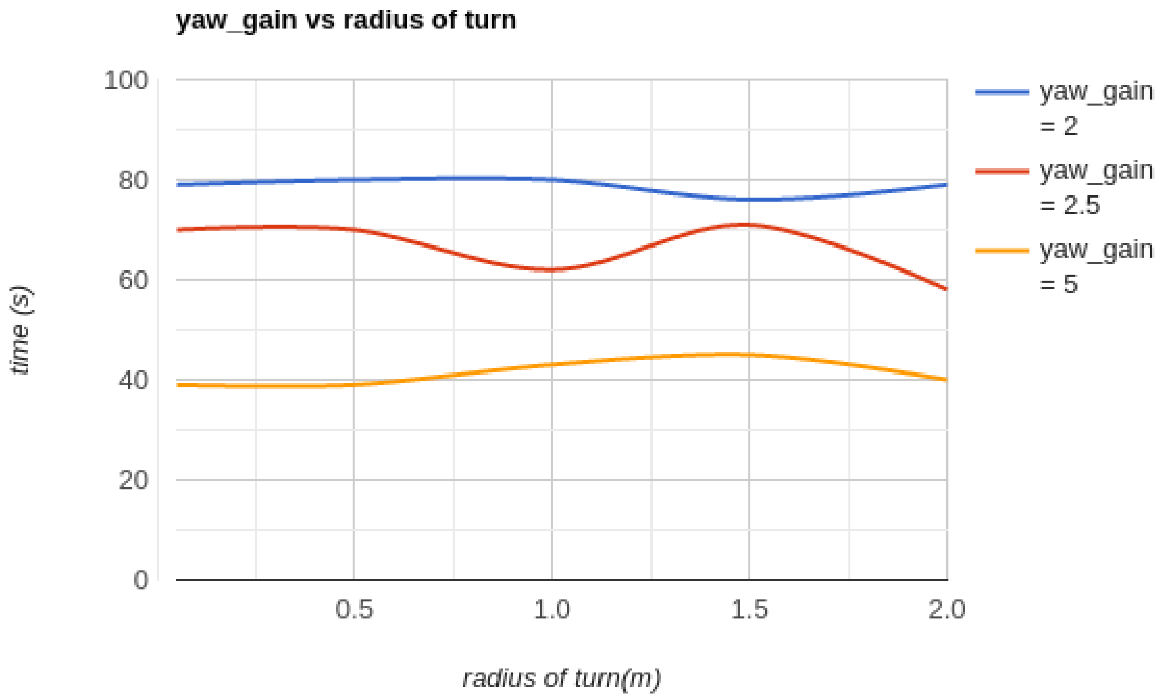
- **vehicleLength** : Specifies the robot length. Unit : m
- **vehicleWidth** : Specifies the robot width. Unit : m
- **maxSpeed** : Specifies the maximum speed the robot can operate at. Unit : m/s
- **maxAccel** : Specifies the maximum acceleration the robot can operate at. Unit : m/s^2
- **min_lookahead** : Specifies the minimum lookahead point the robot must reach on the global path when the robot is in motion. Unit : m. Nominal value : (Robot length / 2) * 1.1
- **in_place_rotation_penalty** : This parameter specifies the weight factor to be used while scoring the different free paths available when the robot is confronted by an obstacle. The following plot shows yaw angle Vs time, of a ROSbot when executing a 360^0 U-turn with various values of in_place_rotation_penalty



10.3.2 Parameters influneced by environment and trajectories:

- **pathFolder** : Specifies the path for the motion primitives folder where path files are located.

- **max_lookahead** : specifies the maximum lookahead point the robot must reach on the global path when the robot is in motion . Unit : m
- **max_yaw_rate** : Specifies speed at which the robot performs on spot turn. Unit : rad/s
- **yaw_gain** : Related to rotation of robot while in motion. Following plot shows performance of a ROSbot (time took to complete) on a given trajectory with different turn radii.



- **goal_direction_preference** : Weight factor to be used while scoring the different free paths available when the robot is confronted by an obstacle. In cluttered environment it is recommended to have lower values. Nominal value : 0.8
- **obstacle_horizon** : specifies the distance to which the robot must look inorder to detect an obstacle. Units : m Nominal value : 1.5 m. There is a constraint on this parameter as follows.It should be greater than path distance of the motion primitives.

Users **should only** change the above mentioned parameters and **should not** change any other parameter values in the mw_maxl_planner.yml file

CHAPTER 11

Config : Mission Executive

Below is a brief of the parameters in the Mission Executive Config file.

Parameter	Description
map_frame	This parameter stores the map/global frame id.
robot_frame	This parameter stores the robot frame id.
loop_frequency	The frequency at which the mission_executive node runs.
max_time_lag	Maximum time in which the feedback should be recieved by the controller action server during the controlling state.
decay	The factor by which the velocity of the robot is decreased if the feedback lag from the controller action server passes the set threshhold (`max_time_lag`).
planner	The planner name. User can chose between `NavfnPlanner` and `genroute`.
controller	The controller name. User can chose between `mw_maxl_planner`, `trajectory_planner` (open source local planner).
plan_topic	The topic name at which the path consisting of waypoints is published.
cmd_vel_topic	The topic name at which the command velocity of the robot is published.
route_topic	The topic name at which the route markers for visualizing the route in Rviz are published.
goal_queue_topic	The topic name at which the goal queue markers for vizualizing the goals in Rviz are published.
tf_timeout	The time threshold in which the current robot pose should be updated.
max_retries	The maximum amount of times the controller is allowed to reset before aborting the mission.
min_distance_tolerance	If the robot traverses a distance lesser than the this tolerance for a time greater than the `recovery_timeouts`, recovery will be triggered. The greater the value the more sensitive the robot is to triggering recovery.
min_angular_tolerance	If the robot traverses an angular distance lesser than the this tolerance for a time greater than the `recovery_timeouts`, recovery will be triggered. The greater the value the more sensitive the robot is to triggering recovery.
controller_reset_timeout	The time threshold after which the controller is reset.
recovery_timeouts i.e. recoveries	The timeouts for differnt types of recoveries are set in this field. The increasing order of timeouts decides the order in which recoveries will be executed. Recovery with the least timeout will be executed first and so on.
goal_queue_mode	If `true`, the executive will add new goals to a queue and pursue them on a one-by-one basis. If `false`, the new goals will replace the old goal and only the latest goal will be pursued.

CHAPTER 12

Running the Actual ROSBot

If have you a Husarion ROSbot you can try out Mowito's Navigation stack directly on it.

12.1 Connect To ROSBot

12.1.1 1. Get the ROSbot connected to a wifi

1.1. Connect a screen and keyboard with ROSBot, and then connect the ROSbot to the wifi.

1.2. Get the Laptop (ground station) on the same network as the ROSbot. This laptop will be used give goals to the ROSbot and visualize the path, sensore input and other info from the ROSbot

12.1.2 2. Get the IP address of the RObot

2.1. Open the terminal

2.2. execute the followin command `hostname -I` The output is the IP address of the ROSbot, note it down.

12.1.3 3. SSH into the ROSbot

3.1. Open the terminal on your laptop and execute the following `ssh husarion@<ip address of the ROSbot>`

3.2. If you avahi daemon running on the robot then you can instead try `ssh husarion@husarion.local`where husarion is the user name and hostname of the ROSbot respectively.

12.1.4 4. Export ROS_IP on ROSBot

4.1. *SSH* into the ROSbot and execute the following : `export ROS_IP=<ip address of the ROSbot>`

4.2. If you have avahi daemon running then you can instead try: `export ROS_IP=husarion.local`

where husarion is the hostanme of your ROSbot

You have to execute the above commands every time you ssh into ROSbot to run Mowito's navigation stack.

12.1.5 5. Setup the Laptop (ground station)

5.1. Export ROS_IP Get the IP address of the laptop, by executing the following on the laptop's terminal `hostname -I`

then on the same terminal `export ROS_IP=<ip address of the laptop>`

5.2. Export ROS_MASTER_URI One same terminal execute the following: `export ROS_MASTER_URI=http://<ip address of the ROSbot>:11311`

You have execute above two commans on each terminal of Laptop (Ground station) which you want to use for communicating to the ROSbot.

12.2 Setup Mowito's Stack On Robot

12.2.1 User Registration

If you have already done, you can skip this step.

Register yourself on this website https://mowito.in/navigation_stack.html

We need your email to mail you the password, and to count how many people are using Mowito.

We won't spam. :)

12.2.2 Installation Mowito Navigation Stack

1. *SSH* into the ROSBot
2. **Create mowito directory** `mkdir -p ~/mowito_ws/src/`
3. **Clone the repo containing the debians:** `cd ~/mowito_ws/src`

   for ROS melodic on arm 64 `git clone https://github.com/mowito/mowito_arm64.git --branch melodic`

   for ROS kinetic on armV7 (armhf) `git clone https://github.com/mowito/mowito_armv7.git --branch kinetic`
4. **Remove any previous installation of Mowito stack** `cd mowito_arm64` or `cd mowito_armv7` based on arm 64 or armV7 respcectively.

   for ROS melodic `./remove_mowito.sh melodic`

   for ROS kinetic `./remove_mowito.sh kinetic`

5. **Install the new Mowito stack** For ROS melodic `./setup_mowito.sh melodic`

   For ROS kinetic `./setup_mowito.sh kinetic`

6. **In the end, the setup will ask for the uesr registeration.** Use the user name you used on the registration website and password that was mailed to you. You can use any name as robot name.

12.2.3 Installation of Mowito Rosbot Package

Mowito Rosbot package simply contains the necessary launch files and config files, which Mowito team create for easy deployment on ROSbot.

1. *SSH* into the ROSBot
2. clone the Mowito ROSbot package into the mowito_ws

   `cd ~/mowito_ws/src && git clone https://github.com/mowito/mowito_rosbot.git`
3. Build the Mowito ROSbot package.

   `cd ~/mowito_ws && catkin_make`

   of if you use catkin build tools

   `cd ~/mowito_ws && catkin build`
4. source the mowito_ws whenever you need to run mowito_rosbot

   `source ~/mowito_ws/devel/setup.bash`

   TIP:: you can add the above command in you `~/.bashrc` so that its atuomatically executed everytime you open the terminal.

12.3 Navigation - Without Map

During this phase, for navigation the robot, you can use two methods

1. Manual Navigation - Using Tele-Operation
2. Autonomou Navigation - By giving goals through Rviz

12.3.1 Method 1 : Manual Navigation

Step 0 : SSH into the rosbot and on it source the workspace

```
ssh husarion@husarion.local
```

```
source ~/mowito_ws/devel/setup.bash
```

Step 1 : Launch the mw_mapping node

1. With Mowito Mapping (default)

```
roslaunch mowito_rosbot run_mw_mapping.launch
```

2. With cartographer

```
roslaunch mowito_rosbot run_mw_mapping.launch cartographer:=true
```

Step 2 : Launch the remote control for providing commands to the bot

in another terminal, *SSH* in into rosbot and run the following command :

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

12.3.2 Method 2 : Autonomous Navigation

Here, the robot will explore the map based on the goal destination provided by the user on RViz.

Step 0 : SSH into the rosbot and on it source the workspace

```
ssh husarion@husarion.local
```

```
source ~/mowito_ws/devel/setup.bash
```

Step 1 : Launch the Mowito Navigation without Map

1. With Mowito mapping.

```
roslaunch mowito_rosbot run_mw_navigation_with_no_map.launch
```

2. With Cartographer

```
roslaunch mowito_rosbot run_mw_navigation_with_no_map.launch
cartographer:=true
```

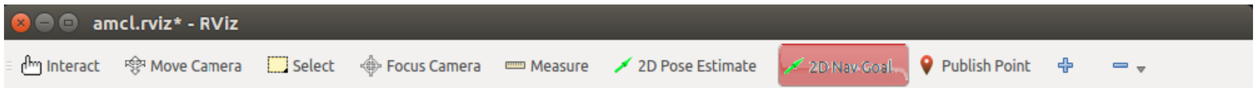
Step 2: Provide Goal

Open another terminal, export *ROS_MASTER_URI and ROS_IP* and then source ROS and *start rviz*:

```
rviz
```

The goal can be provided on RViz using the "2D Nav Goal" feature provided on RViz.

The icon is highlighted in red in the image below.



12.3.3 Saving the Map

Once you are done creating the map on rviz, for saving the map on a new terminal, **ssh into rosbot** and based on the mapping tool you were using execute the following command:

1) Mowito Mapping (defautl)

```
cd && rosrun map_server map_saver -f mymap
```

the map (pgm and yaml) is saved in the home directory with the name mymap.pgm and mymap.yaml

2) Cartographer , run the following command

```
rosrun mowito_rosbot save_carto_map.sh map_name
```

the map (pbstream) is saved in the home directory with the name map_name.pbstream. If no map_name is given then it would save as map.pbstream

12.4 Navigation - With Map

12.4.1 Step 0 : SSH into ROSBot and Source the workspace

```
source <path_to_mowito_ws>/devel/setup.bash
```

12.4.2 Step 1 : Place the robot

Preferable place the robot at the origin of map (the place where you started mapping)

12.4.3 Step 2 : Run the Mowito's Navigation Stack

Open a terminal and SSH into the ROSBOT

1. Using map made from Mowito Mapping (in previous step)

```
roslaunch mowito_rosbot run_mw_navigation.launch
```

If you want to use the map created in the previous section use the following command

```
roslaunch mowito_rosbot run_mw_navigation.launch map_path:=/home/
husarion/mymap.yaml
```

2. Cartographer based Localization

Use this if you created the map from cartographer in the previous step

```
roslaunch mowito_rosbot run_mw_navigation.launch cartographer:=true
```

12.4.4 Step 3 : Give the goals

In another terminal, export the *ROS_MASTER_URI and ROS_IP*, source ros and *start rviz*:

```
rviz
```

In the rviz, click on the second top panel, click on the nav goal option, and click on the displayed map to give goal to the robot.

12.5 Configuring Navigation Stack

Check out our *documentation on configuring Mowito Navigation Stack* on a robot.

12.6 Setting Up Rviz

Rviz is a tool for visualizing what the robot is seeing. Further, it could also provide GUI for the user to interact with the robot. Rviz can be opened in the computer (with screen) - most probably not the ROSbot, using the command `rviz` (after sourcing ROS).

In order to visualize all the interesting information on Rviz you have to add the topics on which they are getting published. You can find more information on this http://wiki.ros.org/rviz/UserGuide

To add a topic of visualisation:

1. On the left "Display" pane, click on "add"
2. Click on "by topic"
3. select the topic name
4. click on "ok"

now you one-by-one you have to add the following topics: `scan` `/map` `/plan` `/costmap/local_costmap/footprint` `/free_paths` `/local_path`

for visualizing the axis of the robot and other frames:

1. On the left "Display" pane, click on "add"
2. Click on "by display type"
3. select "axes"
4. click on "ok"

Once you are satisfied with the configuration, click on File > save config, so that you don't have to configure Rviz everytime you open it.

CHAPTER 13

Interfaces

13.1 ROS-Topics

Below is the list of ROStopics in the Mowito Navigation Stack. In order to check the data from any of the topics below, on terminal 1. source ROS `source /opt/ros/<your ros version>/setup.bash` 2. `rostopic echo <ros topic address>`

Topic name	address	description
Command Velocity Publisher	`/cmd_vel`	it contains the linear and angular velocity which the robot should follow.
Plan Publisher	`/plan`	it contains the global path/plan which the robot will follow
Goal Queue Publisher	`/goal_queue`	Publishes all the goals in the queue
Odometry	`/odom`	Contains the odometry of the robot. It is used as **input** by the Navigation Stack
Mission Executive Status Publisher	`/mission_status`	Contains the navigation status of the robot.

13.2 Service Calls

In order to make a service call, on terminal (after sourcing ROS) do `rosservice call <address of the service> <tab><tab>` . Tab-tab to autocomplete the data structure, which user can modify. Service calls are best done programmatically, rather than through terminal. Here is a list of the service calls in the navigation stack.

Service Name	Address	Description
Set Plan	`/mission_executive/set_plan`	It lets the user to set a custom plan for the mission. In other words, this service is used to when custom global planner is used to plan the path, and pass it to the navigation stack for the robot to follow it. one example of program using this service is `rosrun mission_executive set_plan_client`
Set Route	`/mission_executive/set_route`	It lets the user to set multiple waypoints programatically. One example of a program using this service is `set_route_client` (checkout Route based Navigation in Step 4).
Route Status	`/mission_executive/get_route_status`	returns the status of current route the robot is executing
Change Planner	`/mission_executive/change_planner`	changes the path planner used by the robot dynamically (without terminating the stack).
Change Controller	`/mission_executive/change_controller`	changes the controller used by the robot dynamically
Abort Planner Goals	`/mission_executive/abort_controller_goals`	cancels all the controller goals
Abort Mission	`/mission_executive/abort_mission`	cancels all the planner and controller goals thereby aborting the mission
Abort Controller Goal	`/mission_executive/abort_controller_goals`	cancels all the controller goals.
Trigger Recovery	`/mission_executive/trigger_recovery`	triggers the robot into recovery mode
Set Manual Override	`/mission_executive/set_manual_override`	It lets the user take over the executive giving user complete control over the movement the robot. In Manual mode, the executive will not publish command velocities. The executive will also not accept any goals, routes or plans until this service is called again and manual_mode is set to false
Change primitives	`/mw_maxl_planner/change_mprims`	It allows you to change motion primitives at runtime.For service call mention path to motion primitive's Files you would like to change to as argument as an argument to service

CHAPTER 14

Behavior Tree for the Mowito Stack

14.1 Overview

The behavior tree package provides:

- The ability to use mowito's features in a modular fashion.
- The ability to change the tree nodes dynamically without re-compiling the whole stack.
- Easy-to-use XML templates over which the user can add their own features.

14.2 Behavior Tree Node Description

The behavior tree package for the Mowito Stack provides navigation-specific nodes which can be used directly in a Behavior Tree.

BT Node	Type	Description
follow_path	Action	Invokes Mowito's controller action server and makes the robot follow a given path. The node returns SUCCESS if the controller action server succeeds otherwise returns RUNNING.
generate_path	Action	Invokes Mowito's planner action server and generates a path for a given goal. The node returns SUCCESS if the planner action server succeeds otherwise returns RUNNING.
backup	Action	Invokes Mowito's recovery action server to make the robot move back to a specific pose. The node returns SUCCESS if the recovery action was successful otherwise returns RUNNING.
clear_costmap	Action	Invokes Mowito's recovery action server to delete the current global-costmap. The node returns SUCCESS if the recovery action was successful otherwise returns RUNNING.
isStuck	Condition	Determines whether the robot is stuck or not using the robot's odometry. If the robot is not progressing, the condition will return SUCCESS, otherwise it will return FAILURE
set_waypoint	Condition	This condition takes in the Rviz goals given by the user and checks if all the goals have been pursued. If the goals given by the user are being pursued, the condition returns SUCCESS, otherwise it will return FAILURE.
set_route	Condition	This condition takes in the goals given by the user via a route.yaml (type) file and checks if all the goals have been pursued. If all the goals given by the user are being pursued, the condition returns SUCCESS, otherwise it will return FAILURE.
set_plan	Condition	This condition takes in a plan given by the user via a plan.txt (type) file and checks if the given plan has been pursued. If the plan given by the user is being pursued, the condition returns SUCCESS, otherwise it will return FAILURE
Recovery	Control	This control node is designed for a acheiving a desired recovery behavior. Recovery is a control flow node with two children. It returns SUCCESS if and only if the first child returns SUCCESS. The second child will be executed only if the first child returns FAILURE. If the second child SUCCEEDS, then the first child will be executed again. The user can specify how many times the recovery actions should be taken before returning FAILURE
Reactive	Control	This control node is especially designed for acheiving the desired wayoint behavior. Reactive is a control node with two children. It return RUNNING if either of the child returns RUNNING or SUCCESS. If either of the child returns FAILURE, the node will return FAILURE.

14.3 Example Tree Structures

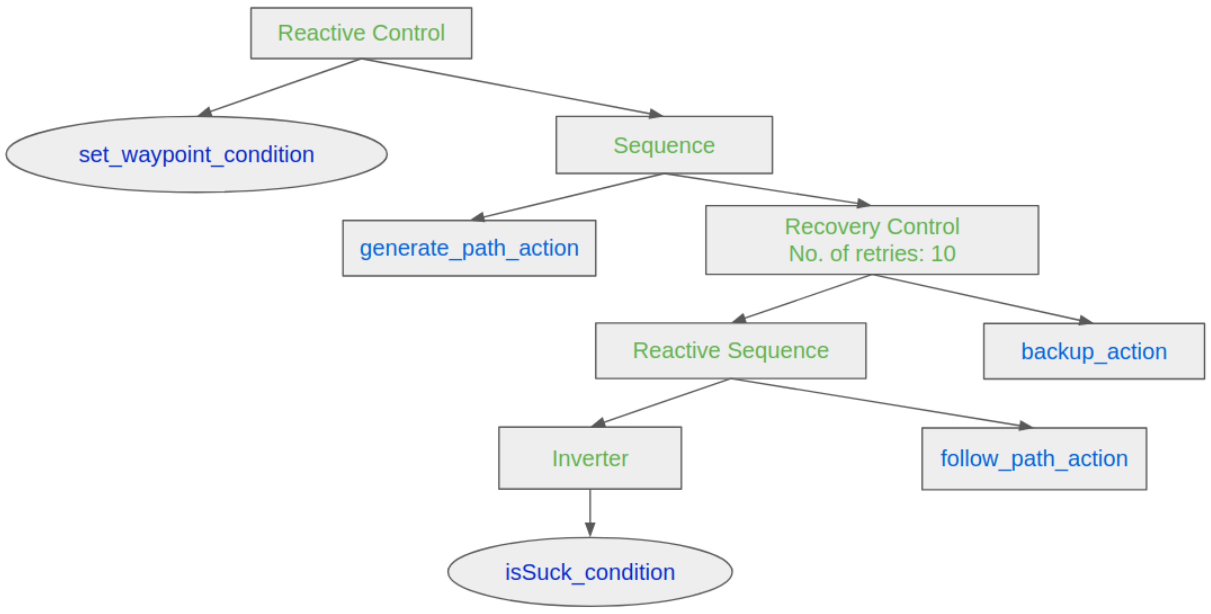
14.3.1 A. Navigate with waypoints and simple recovery actions

The following tree structure can be used for taking multiple goals from the user via the Rviz-Gui. This tree never returns that the action has finished successfully, but will return FAILURE after all the goals have been reached. However, until the system is shut down, the tree will continue to take new goals (if any) from the user and pursue them.

1. To launch the behavior tree for naviagation with waypoints and simple recovery actions, execute:

   ```
   roslaunch behavior_tree sim_bt_nav_waypoint_mode.launch
   ```

2. This behavior tree is contained in the waypoint_navigation_tree.xml file inside the tree folder. The tree folder is alongside the CMakeLists.txt and package.xml files.



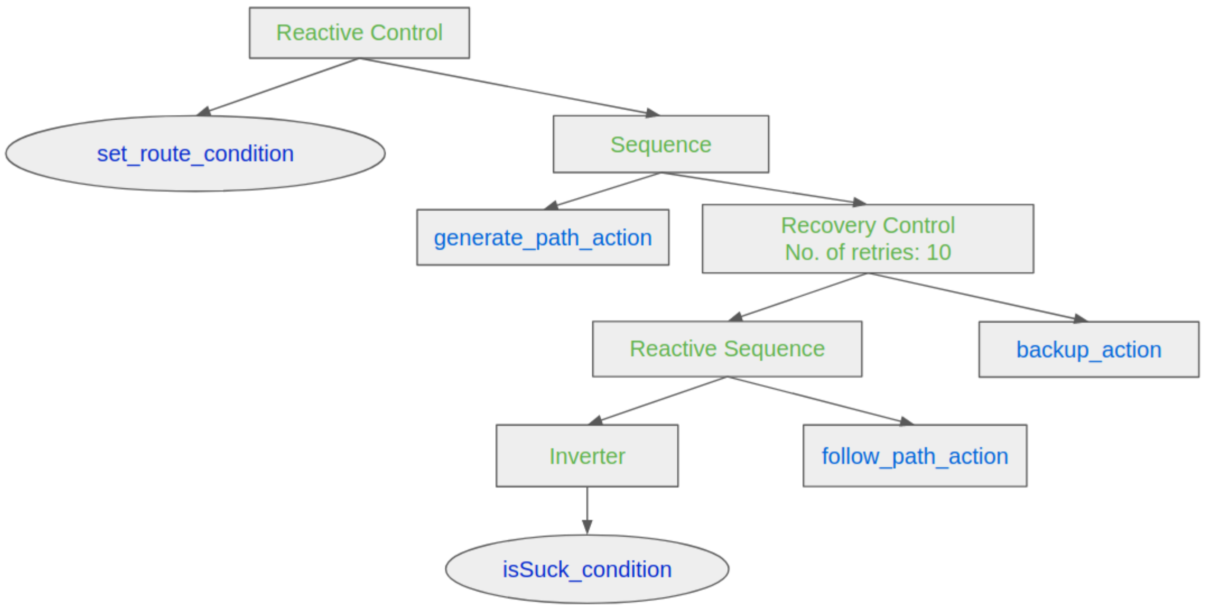
14.3.2 B. Navigate with given route points and simple recovery actions

The following tree structure can be used for taking multiple goals from the user via a given route.yaml (type) file. This tree never returns that the action has finished successfully, but will return FAILURE after all the goals have been reached. However, until the system is shut down, the tree will continue to take new goals (if any) from the user and pursue them.

1. To launch the behavior tree for navigation with route points and simple recovery actions, execute:

   ```
   roslaunch behavior_tree sim_bt_nav_set_route_mode.launch
   ```

2. This behavior tree is contained in the set_route_tree.xml file inside the tree folder. The tree folder is alongside the CMakeLists.txt and package.xml files.



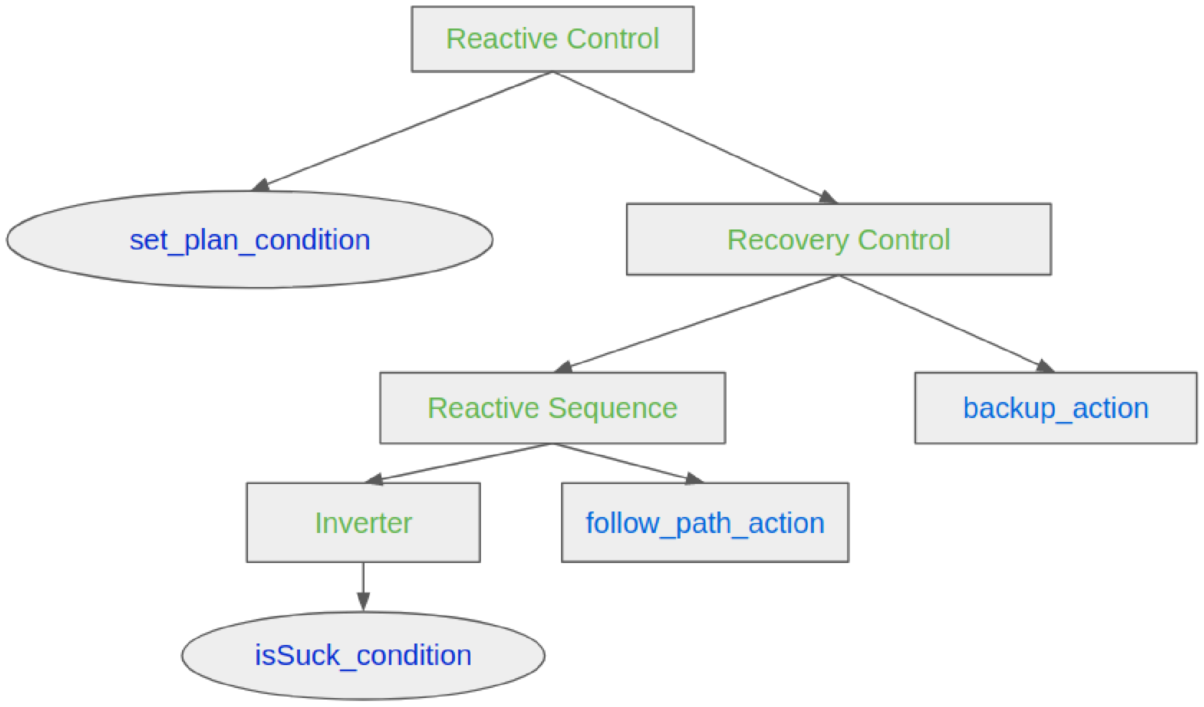
14.3.3 C. Navigate with a given plan and simple recovery actions

The following tree structure can be used for taking multiple plans from the user via a plan.txt (type) file. This tree never returns that the action has finished successfully, but will return FAILURE after all the plans have been reached. However, until the system is shut down, the tree will continue to take new plans (if any) from the user and pursue them.

1. To launch the behavior tree for navigation with a given plan and simple recovery actions, execute:

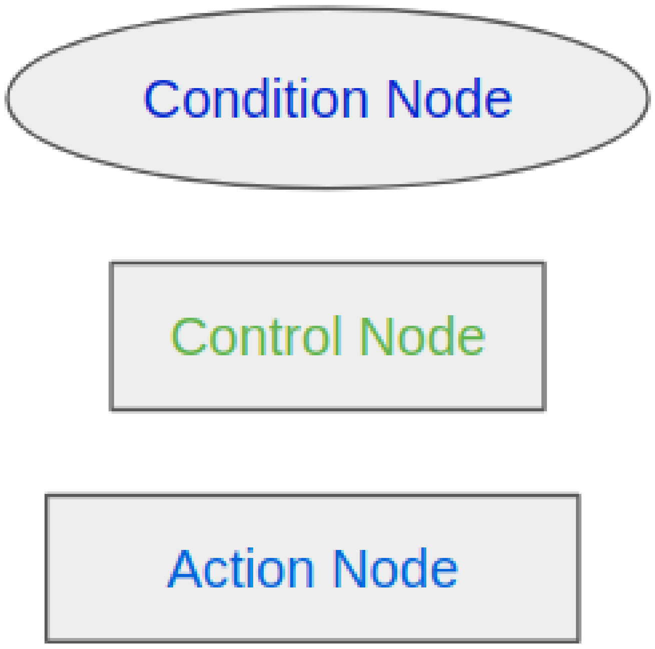
   ```
   roslaunch behavior_tree sim_bt_nav_set_plan_mode.launch
   ```

2. This behavior tree is contained in the set_plan_tree.xml file inside the tree folder. The tree folder is alongside the CMakeLists.txt and package.xml files.



14.4 Legend

Legend for the behavior tree diagrams:



For more information about the behavior tree nodes that are available in the default BehaviorTreeCPP library, see documentation here: https://www.behaviortree.dev/bt_basics/