**Using the RAMP framework**

**Running RAMP with a real robot**

1. roscore
2. Run Rviz, and display the following:
   * Fixed frame: “map”
   * Laser Scan: either “scan” or “scan\_filtered”, the costmap uses “scan\_filtered”
   * Map: either “/costmap\_node/costmap/costmap” or “/accumulated\_costmap”, the accumulated costmap is the one that ramp\_sensing builds and uses to detect obstacles on.
   * MarkerArray: “/visualization\_marker\_array”, this is the topic that ramp\_sensing will publish the circles on for you to view in Rviz
3. Start the robot
   * For a Turtlebot 2, run “roslaunch turtlebot\_bring minimal.launch” and “roslaunch turtlebot\_bringup 3dsensor.launch” **on the robot**
4. Start RAMP on the workstation computer:
   * roslaunch ramp\_launch planner\_full\_costmap.launch
5. Start ramp\_control **on the robot**
   * rosrun ramp\_control ramp\_control
6. Start the planner:
   * rosparam set /ramp/start\_planner true

**Running RAMP in Gazebo simulator**

1. roscore
2. Run Rviz, and display the following:
   * Fixed frame: “map”
   * Laser Scan: either “scan” or “scan\_filtered”, the costmap uses “scan\_filtered”
   * Map: either “/costmap\_node/costmap/costmap” or “/accumulated\_costmap”, the accumulated costmap is the one that ramp\_sensing builds and uses to detect obstacles on.
   * MarkerArray: “/visualization\_marker\_array”, this is the topic that ramp\_sensing will publish the circles on for you to view in Rviz
3. Load planner parameters
   * roslaunch ramp\_launch planner\_parameters.launch
4. Publish map→odom transform
   * rosrun ramp\_planner pub\_map\_odom
5. Start Gazebo and costmap node:
6. Start RAMP:
   * roslaunch ramp\_launch planner\_full\_costmap\_simulation.launch
   * If you want to move obstacles in the Gazebo environment, go to ramp\_planner/config.yaml and set the model names in the gazebo\_model\_names variable (array of strings).
7. Start the planner:
   * rosparam set /ramp/start\_planner true

**Notes about the planner\_full\_costmap.launch file**

This will run the following nodes: ramp\_planner, ramp\_control, trajectory\_generation, trajectory\_evaluation, ramp\_sensing, costmap\_node, laser\_filter, reset\_odometry, pub\_map\_odom

The costmap node depends on the “map” and “base\_footprint” transforms. The node pub\_map\_odom will publish the transform map→odom at a fixed rate. The transform values are based on the robot’s starting pose. The odom frame will be translated by the starting location, and the frame will be rotated by the robot’s starting orientation. These parameters are loaded in automatically based on the “robot\_info/start” rosparam. The transform is published at a rate of 20Hz, and they are published 7 seconds into the future. These two values can be changed in the file ramp\_planner/src/main\_pub\_map\_odom.cpp.

The future time to publish the transform is based on other transform future times. I run “rosrun tf view\_frames && evince frames.pdf”, and match whatever time the other transforms are being published at. Otherwise, the laser data will have some errors about extrapolating into the future. I am not very familiar with the rostf package so there may be a better way to solve this, but matching other transform times solves this for me.

The “pub\_map\_odom” node will crash if the “robot\_info/start” rosparam is not loaded in the server before the node runs. This happens if ROS starts the pub\_map\_odom node before any other nodes. If this happens, you can just run the launch file again.

The planner works with Rviz. In the ramp\_launch package, there is a file “robot\_costmap.rviz” that can be used to configure your Rviz window. If you want to use your own configuration file, make sure the following are shown:

1) Fixed frame: “map”

2) Laser Scan: either “scan” or “scan\_filtered”, the costmap uses “scan\_filtered”

3) Map: either “/costmap\_node/costmap/costmap” or “/accumulated\_costmap”, the accumulated costmap is the one that ramp\_sensing builds and uses to detect obstacles on.

4) MarkerArray: “/visualization\_marker\_array”, this is the topic that ramp\_sensing will publish the circles on for you to view in Rviz

The planner will not run without user input. If the “ramp/use\_start\_param” parameter is true (described below), then the planner will wait until the rosparam “ramp/start\_planner” is set to true. Otherwise, it will wait until the Enter key is pressed.

**RAMP Dependencies**

**Reflexxes Library**

The trajectory\_generator package depends on the Reflexxes Motion Library (http://www.reflexxes.ws/index.html). We use the Type II version. I have included a custom build of this library in the ramp root directory. The custom build fixes a few bugs, and uses a different include structure. Navigate to the ramp root directory, and run the “install\_reflexxes.sh” script:

sudo chmod +x install\_reflexxes.sh

sudo . install\_reflexxes.sh

That script will move the include files to /usr/include, and the executable to /usr/lib. Afterwards, the trajectory\_generator package should compile.

**Bayesian Filtering Library (BFL)**

The ramp\_sensing package depends on the BFL software. There aren’t any custom builds required so refer to the official website for installation: <https://people.mech.kuleuven.be/~tdelaet/bfl_doc/installation_guide/>

**OpenCV**

The ramp\_sensing package depends on the OpenCV library. The official installation guide can be used for this.

**Planner parameters:**

**File: ramp\_planner/config.yaml**

This file sets various parameters for the RAMP algorithm:

1) population\_size – this sets the number of trajectories available to the robot. I keep this between 5-12. Increasing it has significant effects on the system’s performance.

2) sub\_populations – this turns on sub-populations, but that code has not been updated in a long time so it is best to leave this as false.

3) evaluations – if this is false, then trajectories will not be evaluated. This is sometimes useful when debugging the system.

4) seed\_population – if this is true, then the population is initialized with specific trajectories, rather than being initializing completely with random trajectories. The specific trajectories need to be manually coded, so it is best to leave this as false.

5) error\_reduction – if this is true, then the system will account for motion error during the planning cycles, ie the Adjustment procedure will run. This should be left as true.

6) only\_sensing – if this is true, then both planning and control cycles do not run. This is useful when debugging the sensing module, but otherwise should be false.

7) moving\_robot – if this is false, then the control cycles will not run. This is useful for debugging, but otherwise should be true.

8) gens\_before\_control\_cycle – this is the number of generations (or planning cycles) to run before the control cycles begin running

9) pop\_traj\_type – this sets the type of trajectory for the population. This is useful for debugging purposes, but otherwise it should be set to 1 (hybrid trajectories).

10) fixed\_control\_cycle\_rate – this is the rate of control cycles. Control cycles will actually occur at dynamic times (details are in our IROS 2016 paper), but this value sets the maximum time.

11) sensing\_cycle\_rate – this sets the Hz that the ramp\_sensing package will publish the latest obstacle information.

12) eval\_weight\_T – this sets the weight for the time term in the evaluation function.

13) eval\_weight\_D – this sets the weight for the minimum obstacle distance term in the evaluation function.

14) eval\_weight\_A – this sets the weight for the change in orientation term in the evaluation function.

15) global\_frame – this sets the frame that the RAMP system will use as the global frame

16) update\_topic – this sets the topic that RAMP will use to get updates about the robot’s pose

17) shrink\_ranges – if this is true, the environment bounds are reduced based on the robot’s radius. This is necessary if the environment is surrounded by walls.

18) use\_start\_param – if this is true, then the planner will not start until the rosparam ramp/start\_planner is true.

19) obstacle\_vels – this is an array to set the topics that are used to publish obstacle velocities on. This is only needed when we are using other robots as obstacles and we want the obstacle to move on specific trajectories.

20) obstacle\_delays – this is an array of times to delay obstacle movement. This is only used in the obstacle\_twist\_all node in ramp\_planner.

21) gazebo\_model\_names – this is an array of Gazebo model names. This is only used to move obstacles in Gazebo.

**Changing RAMP values**

**Changing the environment bounds**

File: ramp\_launch/launch/config/robot\_0.yaml

Edit the “DOF\_min” and “DOF\_max” variables. They are both 3-element arrays in the form [x\_value, y\_value, orientation\_value]. Make sure to change the costmap width and height to match the new bounds (described below).

**Changing the start and/or goal**

File: ramp\_launch/launch/config/robot\_0.yaml

Edit the “start” and/or “goal” variables. They are both 3-element arrays in the form [x\_value, y\_value, orientation\_value].

**Changing the maximum speeds**

File: ramp\_launch/launch/config/robot\_0.yaml

Edit the “max\_speed\_linear” and/or “max\_speed\_angular” variables.

**Changing the robot’s radius**

File: ramp\_launch/launch/config/robot\_0.yaml

Edit the “radius” variable. The unit is meters.

**Changing the costmap parameters**

File: ramp\_launch/launch/costmap\_param/my\_rolling\_costmap.yaml

All of the costmap parameters are defined in this file. You will have to change some parameters in there when you run RAMP in a new environment. Specifically, make sure that the width and height are **double** the environment bounds. They need to be double because the costmap is centered on the robot, so we must increase the costmap bounds to extend far enough in front of the robot to match the environment bounds.

**Connecting to a robot**

The ramp\_control node will publish geometry\_msgs/Twist messages on topics “twist” and “cmd\_vel”. You must have a node that subscribes to one of these topics and drives the robot based on Twist messages.

**Changing robot model**

If you want to change what type of robot you use (e.g. switching from a Turtlebot to Cooky), edit the ramp\_launch/launch/config/robot\_0.yaml file. In this file, you can set a new radius, new maximum speeds, and new DOF ranges. The parameters in this file are loaded in the planner\_parameters.launch file with the line:

<rosparam file="$(find ramp\_launch)/launch/config/robot\_0.yaml" command="load" ns="robot\_info/" />

If you want to use a different configuration file for the robot, create the file and replace the file argument in the above line in planner\_parameters.launch.