Assignment 3

The purpose of this assignment is to learn using the TF package and to investigate driving a robot along a path, generated by a global path planner.

# Getting ready

Study TF (<http://wiki.ros.org/tf> ), the ROS package that lets the user keep track of multiple coordinate frames over time. From the TF tutorials at <http://wiki.ros.org/tf/Tutorials>, do at least the tutorials “*Introduction to tf*” and “*Writing a tf listener*”.

Study path following (path tracking) algorithms. See slides and following pointers:

* <http://www8.cs.umu.se/education/examina/Rapporter/465.pdf>
* <http://cradpdf.drdc.gc.ca/PDFS/unc45/p524913.pdf>
* <http://www.red3d.com/cwr/steer/>

# The Task

In this task you are going to develop various ‘global planners’ and a ‘local\_planner’. A ‘*global\_planner’* calculates a path which it publishes as a ‘*nav\_msgs/Path*’ message on the topic *‘/plan*’. A path (<http://www.ros.org/doc/api/nav_msgs/html/msg/Path.html>) is an array of poses in the /map coordinate frame that represents the path for a robot to follow. A ‘*local\_planner*’ makes the robot follow that path. It uses a *TF listener* to lookup the various poses and steers the robot via ‘*Twist*’ messages which it publishes on *‘/cmd\_vel’*.

/plan

/cmd\_vel

/tf

Use the same setting as in previous assignment: Stage simulator with empty.world.

In *rqt* you can activate the TF plugin to see the TF tree. Note that the Stage simulator provides the transformation */odom -> /base\_link*, but not */map -> /odom*. Normally this transform is published by a node which performs localization, a subject we will be dealing with later in this course. For now you can however simulate such localization node using the following command: *rosrun tf static\_transform\_publisher 0 0 0 0 0 0 map odom 100.* This node can of course also be started from a launch file: *<node pkg="tf" type="static\_transform\_publisher" name="localizer" args="0 0 0 0 0 0 map odom 100" />*. If you want to inspect the pose of the robot on the map: *rosrun tf tf\_echo /map /base\_link*

Create a new package “*assignment3*”. In this package write the following local\_planner node:

* ‘*follow\_carrot’*
  + Implement some sort of ‘Follow-the-Carrot’ algorithm.

Also write the following global planner nodes:

* ‘*rectangle\_planner*’
  + This nodes generates a rectangular path and publishes it on the /plan topic.
* *‘triangle\_planner’*
  + This nodes generates a triangular path and publishes it on the /plan topic.

Requirements:

* You must implement a lookahead distance
* There are various ways of calculating the position of the carrot (see hints below). You have the freedom to select one which you think is best or easiest to calculate. *Explain your choice in your documentation!*
* When within lookahead distance of the final waypoint you must decrease speed proportionally as in servoing in previous assignment.
* Each combination of global\_planner and local\_planner must have a separate launch file.
* Preferably parameterize each type of node. Parameters can be published to the Parameter Server via the launch file.

Now do the following research:

* Investigate the performance of various combinations of global planner and local\_planner nodes. In particular *investigate the influence of the lookahead distance*, i.e. try various distances and show the effect of making it smaller or larger.

Hints:

* One way of calculating the position of the carrot is to intersect the path with a lookahead circle, i.e. a circle with radius equal to the lookahead distance. How to intersect a line (path segment) and a circle is described for instance here:
  + <https://www.google.com/search?q=line+circle+intersection>
  + <http://paulbourke.net/geometry/circlesphere/>
* Another way of calculating the position of the carrot is to find the closest point on the path. From this point “walk the path” for a distance equal to the lookahead distance to find the carrot point. Finding the shortest distance to a line and/or closest point on a line is described for example here:
  + <http://paulbourke.net/geometry/pointlineplane/>
  + <https://en.wikipedia.org/wiki/Distance_from_a_point_to_a_line>
  + <https://stackoverflow.com/questions/3120357/get-closest-point-to-a-line>
* There are other ideas I have seen in the past (I like to learn from you!) :
  + real robot following the carrot that perfectly follows the path
  + robot changing attention to following waypoint when close enough to current waypoint

# What to Submit

Your submission for this assignment will have two parts:

1. The source code, i.e. a zip file containing the package
2. A document which should:
   1. explain how everything works and include computation graphs,
   2. describe test scenario’s and how to launch them
   3. describe the results of your tests and include screendumps of trails in Stage.