UNIVERSITA DEGLI STUDI DI GENOVA

RRT* for robot path planning

Alberto Remus
Simone Maccio'

February 25, 2018

Abstract

Rapidly Random exploring Tree star is an efficient algorithm developed in order to find the optimal path from a certain source region to a goal one inside a workspace by avoiding the obstacles in it. Our objective was to use it in order to make the Baxter robot move an object grasped with one of its arm among the workspace filled with some obstacles.

1 RRT* IN BRIEF

BASICALLY the algorithm exploits random sampling to progressively explore the overall workspace paying attention to avoid collision and computing the optimal path once the goal region has been reached. It has been demonstrated to be asymptotically optimal from a probabilistic point of view, however its implementation is slightly more complex than its "father", the RRT.

The evolution of RRT, introduced by RRT*, consists in considering at each iteration the minimum cost path among the neighborhood, a set of vertices of the tree at a certain distance from the new vertex selected after the sampling.

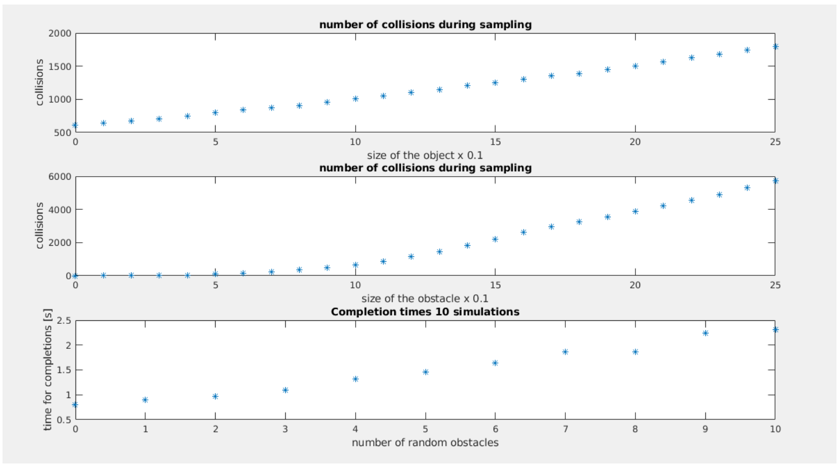
2 THE STUFF IN THEORY

WE DEVELOPED OUR C++ PROJECT modifying the code so that vertices of the RRT* have a size which must be kept into account by the collision avoidance routine, moreover some modifications were introduced in order to grant the sampling occured inside the workspace.

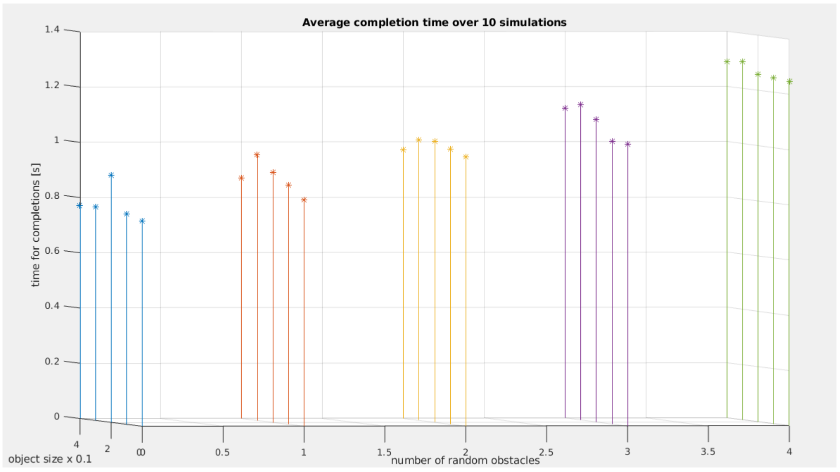
Once we completed this preliminar work, we analyzed the impacts of using RRT* with non zero dimensions nodes, finding out some interesting linear relationships by tuning some parameters:

- size of the obstacles
- number of obstacles
- size of the grasped object

2.1 Simulation results



(a) Cost in terms of computational time and number of collisions



(b) Relationships obtained by increasing number and size of obstacles

The comparison made between the original code and the one which keeps into account the size showed an increment of 25-30 percent over 20000 samplings on the latter version due only to very few lines of code for checking collision with non zero size objects.

As one may notice in the latter figure we have a linear relationship of computational time tuning the sizes of obstacles and object, however the number of obstacles is the one which affects the most the time as shown in the second figure

3 The stuff in practise

Although the results provided by the simulations have been interesting the standalone version is not enough to make the Baxter robot moving over the workspace performing collision avoidance, that's now that ROS comes into play

We implemented the RRT* algorithm as a service, what we needed was to introduce a new node whose objective is to provide all the inputs necessary for the service extracting the optimal path trajectory as output to be sent to the control node responsible for moving the Baxter, in the next pages you can have a look to a matlab simulation inside some real workspaces and our software architecture.

4 What's next

So far our obstacle avoidance algorithm is performed only on Baxter without changing the orientation, actually our work is conceived to be extended for other kind of robots with little modifications

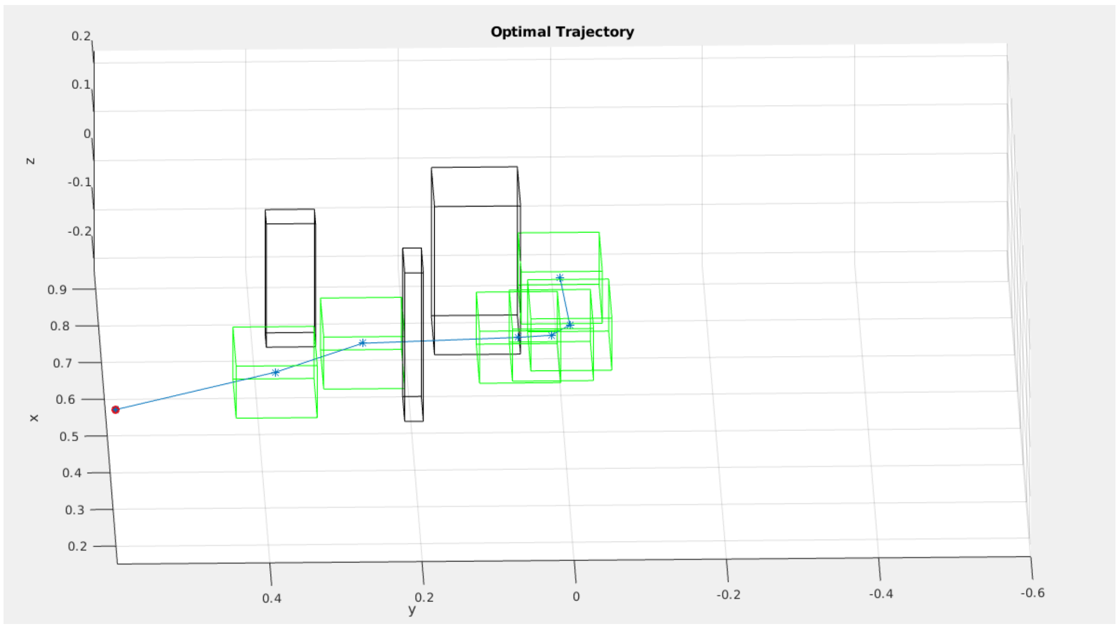
In real world we cannot think not to exploit the degrees of freedom that a human or robotic arm provides, therefore even if the algorithm gets more complicated, the capability of changing orientation of the arm can lead to more cost-effective trajectories, not trivial things once our work is extended to quadricopters or mobile robots in which energy consumption matters.

5 References

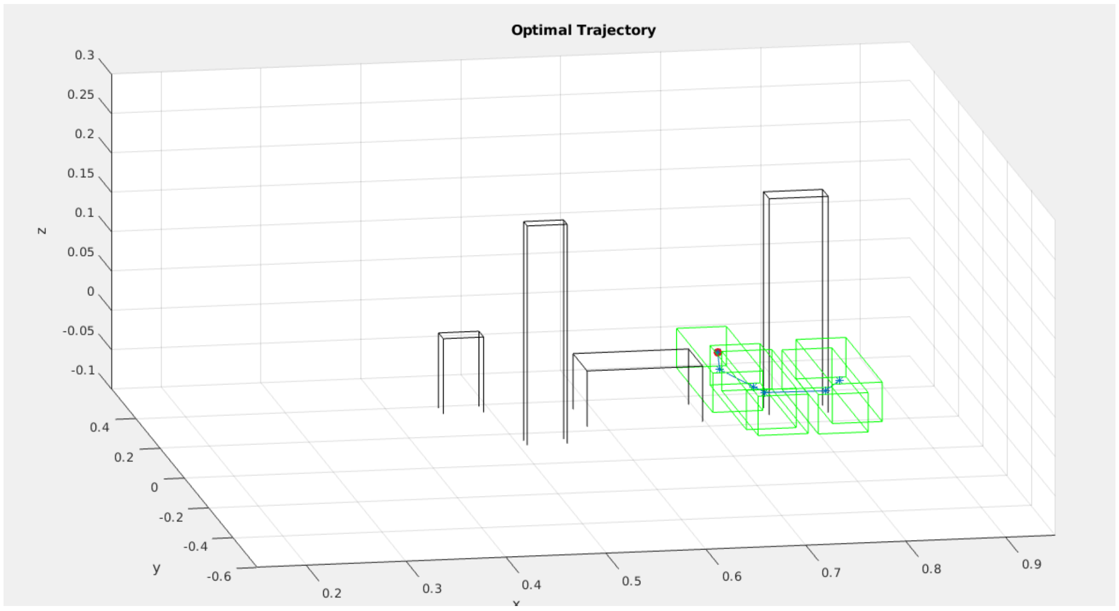
Our work can be found in the github repository `https://github.com/AlbertBit/rrtstar_remus_maccio`
Videos of the Baxter in action can be found at `https://drive.google.com/drive/my-drive`

Our reference material has been the paper Sampling-based algorithms for optimal motion planning - Sertac Karaman and Emilio Frazzoli

A special thank to our supervisor Kourosh Darvish.

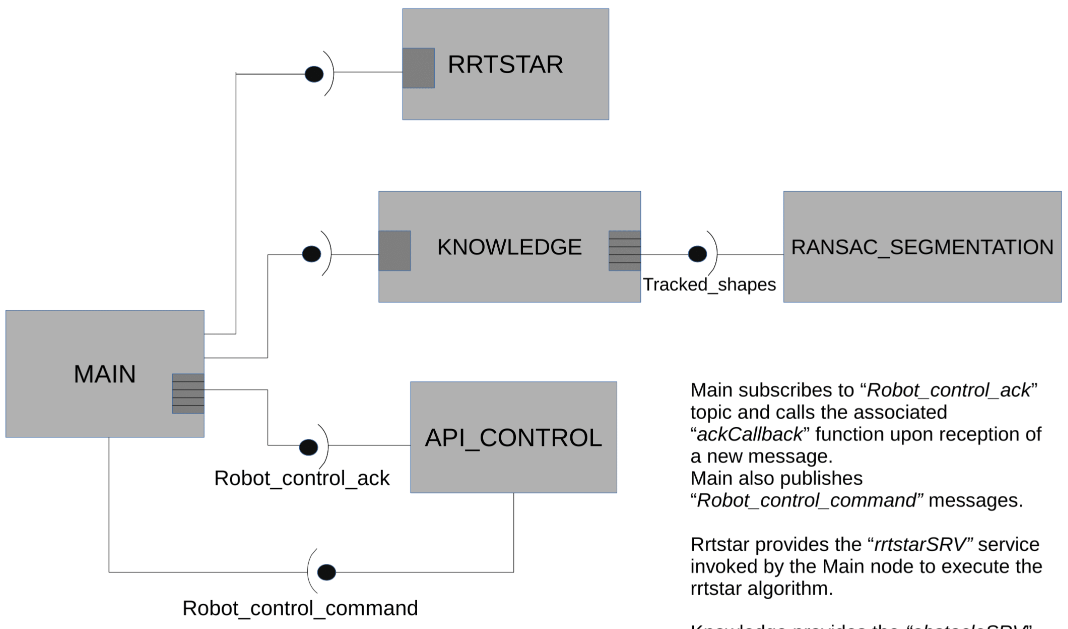


(a) Workspace with 3 cylinders



(b) Workspace with 3 cylinders and 1 sphere

Figure 3.1: The Baxter passing through the obstacles with different workspaces



Main subscribes to "*Robot_control_ack*" topic and calls the associated "*ackCallback*" function upon reception of a new message.
Main also publishes "*Robot_control_command"* messages.

Rrtstar provides the "*rrtstarSRV"* service invoked by the Main node to execute the rrtstar algorithm.

Knowledge provides the *"obstacleSRV*" service invoked by the Main node to get the obstacles in the workspace.
Knowledge also subscribes to the "*Tracked_shapes"* topic and calls the "*CallbackShapes"* function upon reception of a new message.

Figure 3.2: Our ros architecture