Motion Planning with Dynamic Obstacles

ITCS 6152: Robot Motion Planning - Course Project Report

Spring 2016

Aswathi Radhakrishnan (800936796)
aradhak2@uncc.edu

1 Table of Contents

1 INTRODUCTION

This project is a naïve attempt to implement the real-time path planning algorithm RT-RRT* [Naderi et. al] based on Rapidly exploring Random Tree (RRT*) approach. The goal of this project is to find a collision-free path for a robot in an environment consisting of both static and dynamic, moving obstacles.

In the first stage, a roadmap is created for the configuration space of the robot that is collision-free with the obstacles in the environment. In the second stage, as the robot traverses along the path, if it senses a collision with any of the dynamic or moving obstacles, a new path will be generated from the point at which the robot is currently at to the goal to avoid collision with the obstacle.

2 RT-RRT*

2.1 Algorithm

Algorithm 1 RT-RRT*: Our Real-Time Path Planning

Input: $\mathbf{x}_a$, $\mathcal{X}_{obs}$, $\mathbf{x}_{goal}$
Initialize $\mathcal{T}$ with $\mathbf{x}_a$, $\mathcal{Q}_r$, $\mathcal{Q}_s$
loop
  `Update` $\mathbf{x}_{goal}$, $\mathbf{x}_a$, $\mathcal{X}_{free}$ and $\mathcal{X}_{obs}$
  while time is left for `Expansion` and `Rewiring` **do**
    `Expand` and `Rewire` $\mathcal{T}$ using Algorithm 2
  Plan $(\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_k)$ to the goal using Algorithm 6
  if $\mathbf{x}_a$ is close to $\mathbf{x}_0$ **then**
    $\mathbf{x}_0 \leftarrow \mathbf{x}_1$
  `Move` the agent toward $\mathbf{x}_0$ for a limited time
end loop

Algorithm 2 Tree Expansion-and-Rewiring

Input: $\mathcal{T}, \mathcal{Q}_r, \mathcal{Q}_s, k_{max}, r_s$
`Sample` $\mathbf{x}_{rand}$ using (1)
$\mathbf{x}_{closest} = \arg\min_{\mathbf{x} \in \mathcal{X}_{SI}} \texttt{dist}(\mathbf{x}, \mathbf{x}_{rand})$
if $\texttt{line}(\mathbf{x}_{closest}, \mathbf{x}_{rand}) \subset \mathcal{X}_{free}$ **then**
  $\mathcal{X}_{near} = \texttt{FindNodesNear}(\mathbf{x}_{rand}, \mathcal{X}_{SI})$
  if $|\mathcal{X}_{near}| < k_{max}$ or $|\mathbf{x}_{closest} - \mathbf{x}_{rand}| > r_s$ **then**
    $\texttt{AddNodeToTree}(\mathcal{T}, \mathbf{x}_{rand}, \mathbf{x}_{closest}, \mathcal{X}_{near})$
    `Push` $\mathbf{x}_{rand}$ to the first of $\mathcal{Q}_r$
  else
    `Push` $\mathbf{x}_{closest}$ to the first of $\mathcal{Q}_r$
  $\texttt{RewireRandomNode}(\mathcal{Q}_r, \mathcal{T})$
$\texttt{RewireFromRoot}(\mathcal{Q}_s, \mathcal{T})$

Algorithm 3 Add Node To Tree

Input: $\mathcal{T}, \mathbf{x}_{new}, \mathbf{x}_{closest}, \mathcal{X}_{near}$
$\mathbf{x}_{min} = \mathbf{x}_{closest}$, $c_{min} = \texttt{cost}(\mathbf{x}_{closest}) + \texttt{dist}(\mathbf{x}_{closest}, \mathbf{x}_{new})$
for $\mathbf{x}_{near} \in \mathcal{X}_{near}$ **do**
  $c_{new} = \texttt{cost}(\mathbf{x}_{near}) + \texttt{dist}(\mathbf{x}_{near}, \mathbf{x}_{new})$
  if $c_{new} < c_{min}$ and $\texttt{line}(\mathbf{x}_{near}, \mathbf{x}_{new}) \in \mathcal{X}_{free}$ **then**
    $c_{min} = c_{new}$, $\mathbf{x}_{min} = \mathbf{x}_{near}$
$V_{\mathcal{T}} \leftarrow V_{\mathcal{T}} \cup \{\mathbf{x}_{new}\}$, $E_{\mathcal{T}} \leftarrow E_{\mathcal{T}} \cup \{\mathbf{x}_{min}, \mathbf{x}_{new}\}$

Algorithm 4 Rewire Random Nodes

Input: $\mathcal{Q}_r, \mathcal{T}$
repeat
  $\mathbf{x}_r$=PopFirst($\mathcal{Q}_r$), $\mathcal{X}_{near}$=FindNodesNear($\mathbf{x}_r, \mathcal{X}_{SI}$)
  for $\mathbf{x}_{near} \in \mathcal{X}_{near}$ **do**
    c_{old}=cost($\mathbf{x}_{near}$), c_{new}=cost($\mathbf{x}_r$)+dist($\mathbf{x}_r, \mathbf{x}_{near}$)
    if $c_{new} < c_{old}$ and $\texttt{line}(\mathbf{x}_r, \mathbf{x}_{near}) \in \mathcal{X}_{free}$ **then**
      $E_{\mathcal{T}} \leftarrow (E_{\mathcal{T}} \setminus \{\texttt{Parent}(\mathbf{x}_{near}), \mathbf{x}_{near}\}) \cup \{\mathbf{x}_r, \mathbf{x}_{near}\}$
      Push $\mathbf{x}_{near}$ to the end of $\mathcal{Q}_r$
until Time is up or $\mathcal{Q}_r$ is empty.

Algorithm 5 Rewire From the Tree Root

Input: $\mathcal{Q}_s, \mathcal{T}$
if $\mathcal{Q}_s$ is empty **then**
  Push $\mathbf{x}_0$ to $\mathcal{Q}_s$
repeat
  $\mathbf{x}_s$=PopFirst($\mathcal{Q}_s$), $\mathcal{X}_{near}$=FindNodesNear($\mathbf{x}_s, \mathcal{X}_{SI}$)
  for $\mathbf{x}_{near} \in \mathcal{X}_{near}$ **do**
    c_{old}=cost($\mathbf{x}_{near}$), c_{new}=cost($\mathbf{x}_s$)+dist($\mathbf{x}_s, \mathbf{x}_{near}$)
    if $c_{new} < c_{old}$ and $\texttt{line}(\mathbf{x}_s, \mathbf{x}_{near}) \in \mathcal{X}_{free}$ **then**
      $E_{\mathcal{T}} \leftarrow (E_{\mathcal{T}} \setminus \{\texttt{Parent}(\mathbf{x}_{near}), \mathbf{x}_{near}\}) \cup \{\mathbf{x}_s, \mathbf{x}_{near}\}$
    if $\mathbf{x}_{near}$ is not pushed to $\mathcal{Q}_s$ after restarting $\mathcal{Q}_s$ **then**
      Push $\mathbf{x}_{near}$ to the end of $\mathcal{Q}_s$
until Time is up or $\mathcal{Q}_s$ is empty.

2.2 Analysis

Running time and space complexity of RRT* is as follows:

	Algorithm	Probabilistic Completeness	Asymptotic Optimality	Monotone Convergence	Time Complexity		Space Complexity
					Processing	Query	
Proposed Algorithms	PRM*	Yes	Yes	No	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
	k-PRM*						
	RRG	Yes	Yes	Yes	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
	k-RRG						
	RRT*	Yes	Yes	Yes	$O(n \log n)$	$O(n)$	$O(n)$
	k-RRT*						

Source: http://www.cs.berkeley.edu/~pabbeel/cs287-fa15/optreadings/rrtstar.pdf

Although RT-RRT* is designed for dynamic environments, it is also viable in static environments—where it is asymptotically optimal with a runtime of Θ (log n) per iteration for graphs with n nodes. This is similar to RRT and RRT* Θ (log n).

The expected Θ (log n) time is achieved even if tree rewiring is done whenever an obstacle is encountered. This is because rewiring is done only for a small area for a predefined $\varepsilon > 0$ and adjacent node information is maintained at all nodes which improves the graph over time.

Thus, RT-RRT* has time complexity similar to RRT* and its other variants.

2.3 Tools and Environment Setup

The implementation has been done in Python using Pygame library. The project has primarily been tested on Ubuntu. Pygame is a library designed for multimedia software development in Python.

2.3.1 Installation of Pygame

Pygame can be downloaded from http://packages.ubuntu.com/trusty/python-pygame. It can be installed using synaptic package manager.

2.3.2 Directory Structure

The working directory would consist of following files:

1. rrt.py
2. Node.py - Defines nodes or vertices of RRT
3. Object.py - Defines obstacles
4. obstacles.txt - x, y, width, height dimensions of the obstacles
5. round.png - image used to represent obstacles

In a terminal, the program can be run as follows:

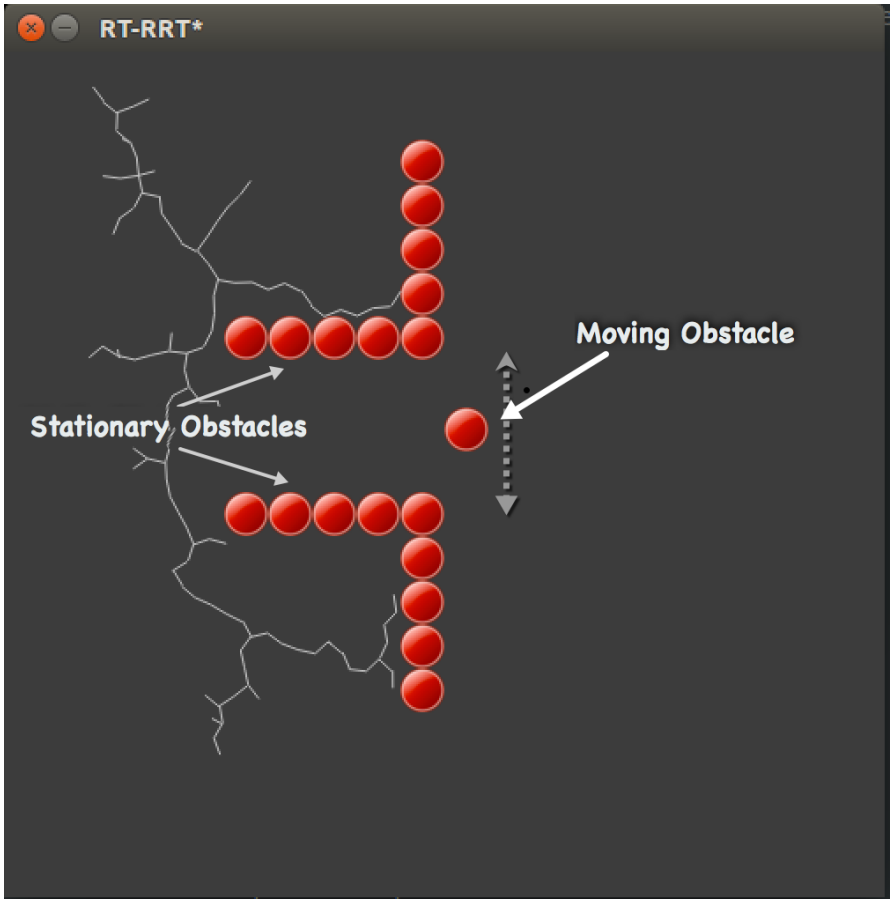
```
>$ python rrt.py
```

2.4 Sample Run with Output

2.4.1 Screenshots

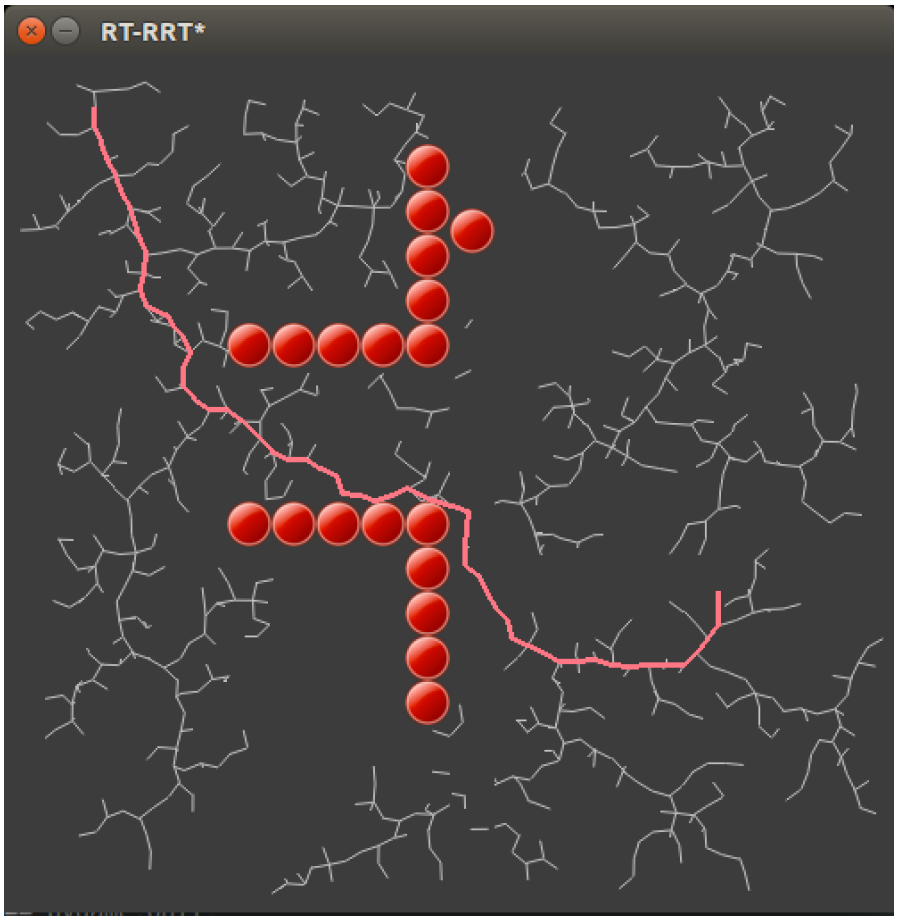
Configuration Space

The Configuration Space of the point robot consists of a few stationary obstacles and a moving obstacle. The direction of motion of the moving obstacle has been shown in the figure. The root node of the tree denotes the Start node.



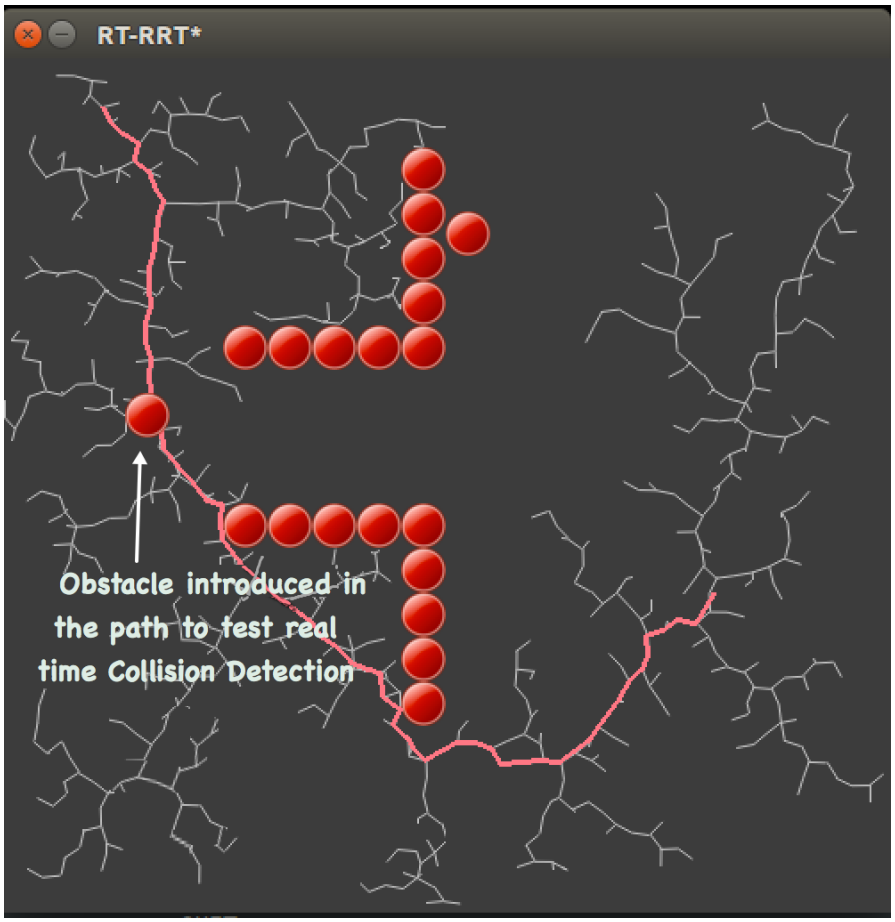
Roadmap from Start to Goal after Stage One

In stage 1, the robot finds a roadmap from its initial configuration to goal configuration while avoiding the stationary and the moving obstacles.



New Dynamic Obstacle Introduced

An obstacle has been introduced into the robot's path to test whether it would sense potential collision.

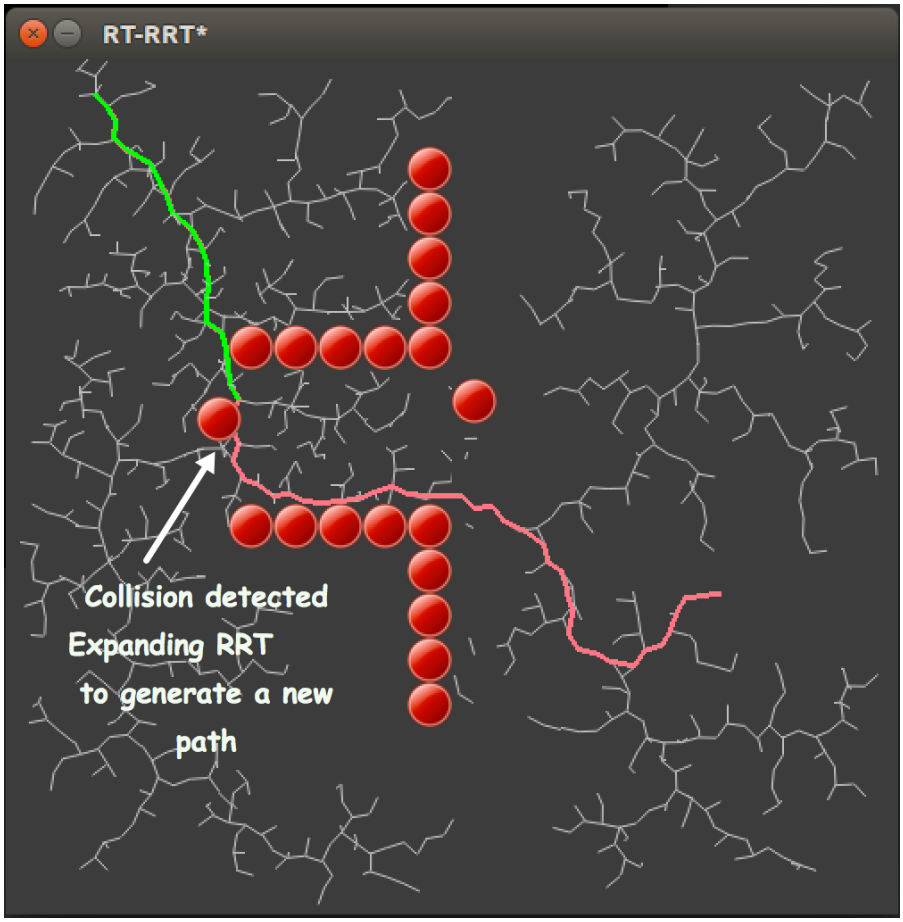


Collision Detected

Collision has been detected and RRT is expanded by adding new nodes. Once the duration of time allocated for tree expansion is exhausted, a new path is generated avoiding the obstacle. Since the number of nodes around the obstacle has now increased, it results in a shorter path to the goal configuration.

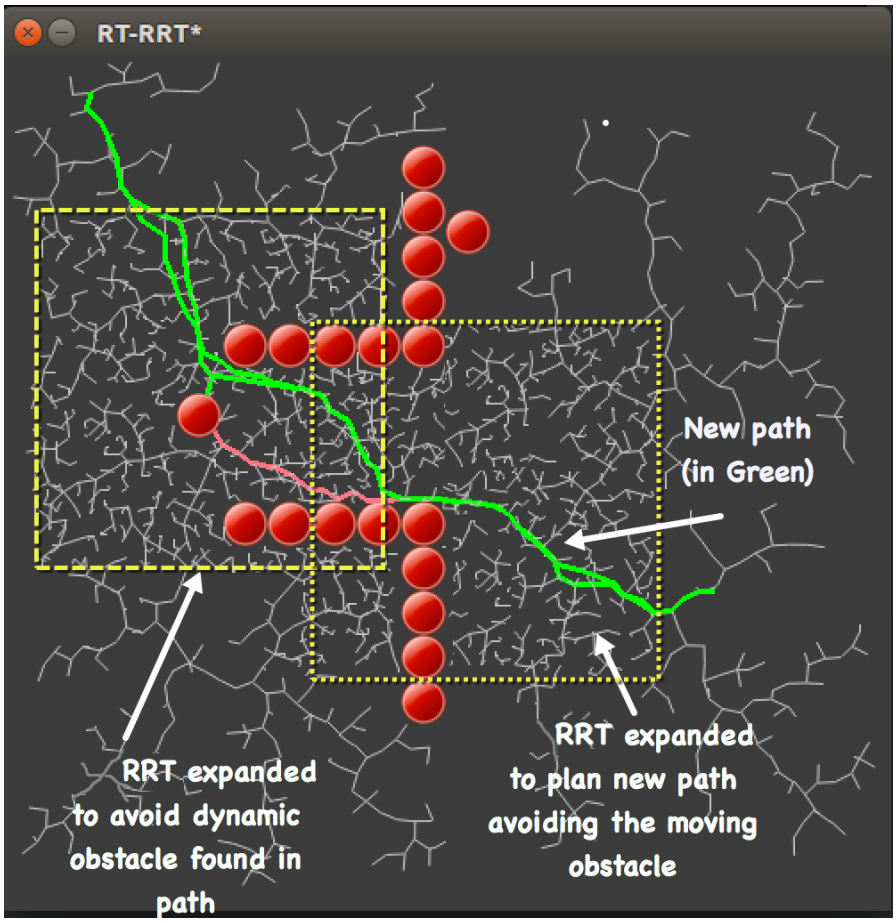
Unlike RRT*, in this stage, RRT is generated only in a small space so as to plan a new path around the colliding obstacle.

The maximum radius within which the random nodes are selected is defined and the values are selected based on random radii values.



New Potential Collision Detected

In the following figure, once the robot found a new path around the first obstacle, it senses a new obstacle (moving) and starts to plan an alternate path in case it faces a collision.



3 COLLISION DETECTION

Bounding box collision detection method has been used to detect collisions with the obstacles.

Code Snippet:

```
def isInCFree(q2):
    for obj in objects:
        if (q2.x > obj.x) and (q2.x + robotRadius < (obj.x + obj.width + robotRadius)) and (q2.y > obj.y) and
(q2.y + robotRadius < (obj.y + obj.height + robotRadius)):
            return True
```

4 DEVIATIONS FROM PROPOSAL

Considering the scope of the project, RT-RRT* hasn't been implemented in its entirety. A few details of the algorithm have been excluded from the implementation.

1. In RT-RRT*, the goal configuration of the robot can be changed on-the-fly which results in instantaneous regeneration and rewiring of the tree. The path to the initial goal is discarded and the robot starts moving to the goal configuration while the path is being generated.
2. RT-RRT* has to be implemented as a multi-query application to realize its full potential.

5 ACHIEVEMENTS

1. Collision detection and path planning avoiding stationary and moving obstacles has been implemented
2. Collision detection has been implemented using Bounding box approach
3. When the robot collides with an obstacle, it plans an alternate path around the obstacle. Expansion of the tree is done locally unlike the original RRT. This enables the robot to find a shorter path to the goal configuration.

6 CHALLENGES

The entire RT-RRT* algorithm couldn't be implemented due to the challenges faced during the initial phase of the project.

1. Setting up the environment took longer than anticipated
2. When generating RRT tree, velocity of moving obstacles influences the coverage of the configuration space. Proper values had to be determined so as not to compromise running time with the complexity of the environment.
3. There was a lot of challenge involved in getting the obstacle to bounce off the edges of the screen.
4. Several graphical issues had to be resolved.

7 REFERENCES

1. Kourosh Naderi, Joose Rajamaki, Perttu Ham al ainen, *RT-RRT*: A Real-Time Path Planning Algorithm Based On RRT** http://webpages.uncc.edu/aradhak2/RT-RRT*.pdf
2. Carlos Hernandez, Jorge A. Baier. *Avoiding and Escaping Depressions in Real-Time Heuristic Search*.
3. Andrea Baumann. *Robot Motion Planning in Time-varying Environments*.
4. http://intelligence.worldofcomputing.net/