Comparison of Several Optimal and Suboptimal Algorithms for Motion Planning in Various Environment

By : Kritika Iyer, WPI, Motion planning project proposal

***Abstract*—So many algorithms have been introduced during the lectures and each algorithm has it's own merits and demerits. Most algorithms are a modified version of A* search as the base of this search returns an optimal solution. The modifications however, have suboptimal solutions but have improved the algorithm in other ways by trading off optimality.The project compares A* , ARA*,and RRT* algorithms in various environments and maps. The results will include a detailed assessment of when which algorithm is best to use and why. The project was carried out in Gazebo using ROS and visualized in Rviz.**

I. Introduction and related work

In the world of motion planning, there are a lot of algorithms that are either novel or derived from base algorithms. To understand them, there is no better way than to implement them yourself and test them for cases. This gives insight into the algorithms and where they are lacking and where they are efficient. In this project, 3 algorithms are taken into consideration and assessed for their properties. The 3 algorithms are A*, ARA* and RRT*. Some of these have been compared previously as mentioned in the following subsections.

A. comparisons of various algorithms

The R*, A*, ARA* algorithms have been tested and compared for humanoid footstep planning [1]. This resulted in ARA* being slow and giving suboptimal solutions. Comparison was also made between ARA*, A* and Dijkstra's algorithm [2].This project focuses on 3 algorithms for wheeled robots. The non-holonomic constraints are not taken into consideration.

1) A Algorithm:* A* is generally used for lower dimensional problems and searches only the most promising paths [3].But it can be modified to be used for higher dimensions as well [5]. A* is used to find the trajectory with least cost and returns an optimal solution [4].

2) ARA:* This algorithm finds a suboptimal path very fast and then keeps updating the path that it has returned to finally return the optimal path [6]. This is based on an epsilon value which makes the huesitic inadmissible at first and then lowers its value till the algorithm is the astar algorithm.

3) RRT:* RRT algorithms have many variations as A* does. Any one of these variations can be implemented. The basic one is a simple tree expansion and node extension system [7]. The other variations include RRT*, Dynamic RRTs, Anytime RRTs and Anytime Dynamic RRTs [8]. Though RRT* is used for high dimensional problems, this project explores how RRT* works with lower dimensions.

B. ROS, Gazebo, and Rviz

ROS is robot operating system. It is an interface by which robots and simulations can connect with each other through ROS nodes. This allows the simulation to provide data to the program dynamically.
Gazebo is a simulation environment which contains physics libraries and other useful plugins. This gives very real simulations of occurrences in real life.
Rviz is a visualization package in which markers can be created through ROS and robot models can be visualized.

II. Methods

The project is an attempt to have a comprehensive understanding of the different algorithms introduced in class and present a cumulative chart to help people find out which algorithm is the best for the environments faced in this project. This also helps understand the nuances between different algorithms. The algorithms were written in C++ and implemented in Gazebo simulation usiing ROS . The car model used is a wheeled car with hokuio sensor as shown in figure 1.

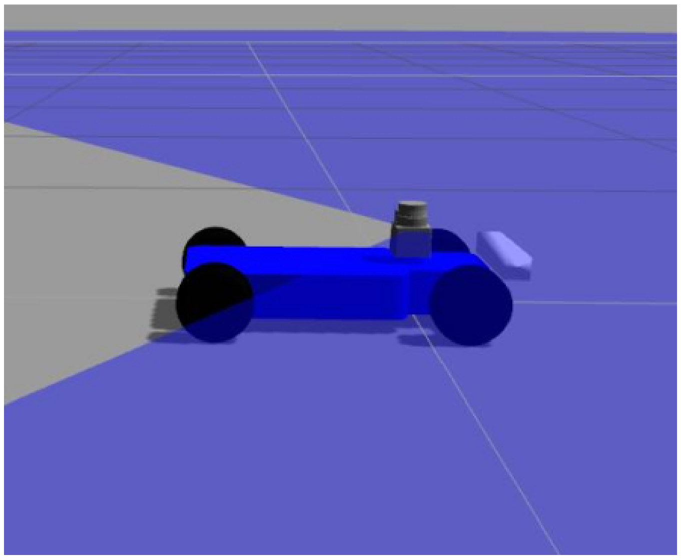


Fig. 1: car model in Gazebo

These algorithms were all tested in the 3 environments. These environments were built by me. The environments are considered to have static obstacles. One environment has a narrow passage like in figure 2a, another has a plain environment with one single obstacle in the middle of the area as shown in figure 2b and finally the last environment is plain with no obstacles2c. The next few subsections talk in detail about the algorithms used and execution of the project.

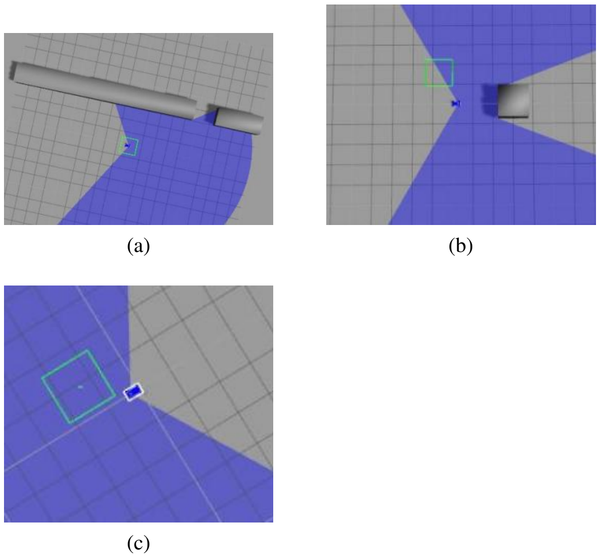
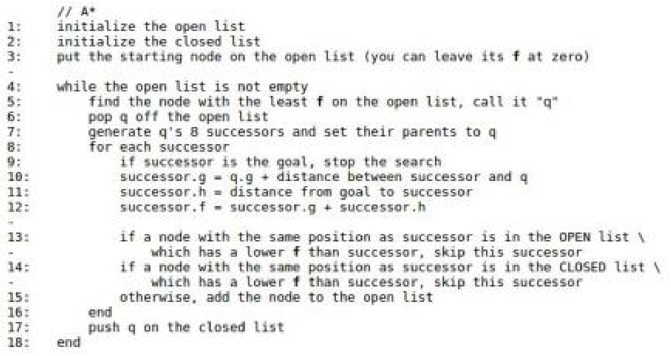

(a) (b) (c)

Fig. 2: Environments

A. *A* algorithm*

The A* algorithm forms the base for many algorithms. It is based on the movement towards a goal position in a slightly biased manner covering nodes at each step. In this project the A* algorithm has step size of 0.1 and neighbors are 8 connected euclidean. The algorithm used is shown in figure 3. The algorithm is optimal as the heuristic function is considered admissible.

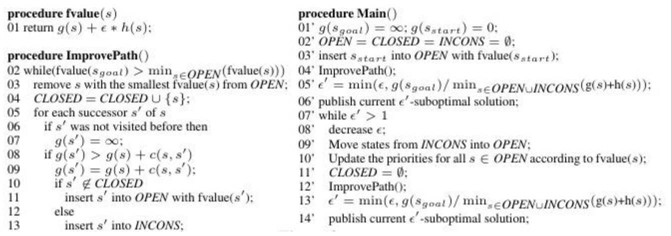


```
// A*
initialize the open list
initialize the closed list
put the starting node on the open list (you can leave its f at zero)

while the open list is not empty
    find the node with the least f on the open list, call it "q"
    pop q off the open list
    generate q's 8 successors and set their parents to q
    for each successor
        if successor is the goal, stop the search
        successor.g = q.g + distance between successor and q
        successor.h = distance from goal to successor
        successor.f = successor.g + successor.h

        if a node with the same position as successor is in the OPEN list \
            which has a lower f than successor, skip this successor
        if a node with the same position as successor is in the CLOSED list \
            which has a lower f than successor, skip this successor
        otherwise, add the node to the open list
    end
    push q on the closed list
end
```

Fig. 3: A* algorithm

B. *ARA**

The ARA* algorithm is a spin off the A* algorithm where the heuristic is inadmissible. It is an iterative process where with each iteration and epsilon value which is multiplied with the heuristic is decreased till the optimal path is found. In this algorithm a sub-optimal path is first returned and then improved upon through iterations. It uses the predetermined paths data to improve upon the path. The algorithm is given in figure 4.

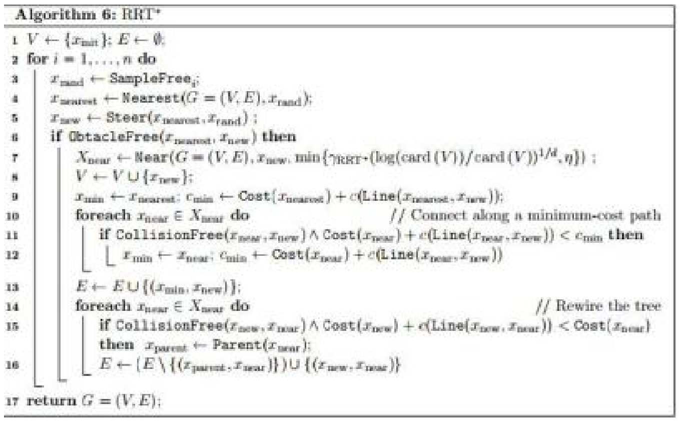


```
procedure fvalue(s)
return $g(s) + \epsilon * h(s)$;

procedure ImprovePath()
while(fvalue($s_{goal}$) > $\min_{s \in OPEN}$(fvalue(s)))
  remove s with the smallest fvalue(s) from OPEN;
  $CLOSED = CLOSED \cup \{s\}$;
  for each successor $s'$ of $s$
    if $s'$ was not visited before then
      $g(s') = \infty$;
    if $g(s') > g(s) + c(s, s')$
      $g(s') = g(s) + c(s, s')$;
      if $s' \notin CLOSED$
        insert $s'$ into OPEN with fvalue($s'$);
      else
        insert $s'$ into INCONS;

procedure Main()
$g(s_{goal}) = \infty; g(s_{start}) = 0$;
$OPEN = CLOSED = INCONS = \emptyset$;
insert $s_{start}$ into OPEN with fvalue($s_{start}$);
ImprovePath();
$\epsilon' = \min(\epsilon, g(s_{goal}) / \min_{s \in OPEN \cup INCONS}(g(s)+h(s)))$;
publish current $\epsilon'$-suboptimal solution;
while $\epsilon' > 1$
  decrease $\epsilon$;
  Move states from INCONS into OPEN;
  Update the priorities for all $s \in OPEN$ according to fvalue(s);
  $CLOSED = \emptyset$;
  ImprovePath();
  $\epsilon' = \min(\epsilon, g(s_{goal}) / \min_{s \in OPEN \cup INCONS}(g(s)+h(s)))$;
  publish current $\epsilon'$-suboptimal solution;
```

Fig. 4: ARA* algorithm [6]

C. *RRT**

The RRT* differs from the A* and ARA* as it is a sampling based algorithm. The nodes are sampled and added to the path. In RRT* there is a goal bias (generally 10-15 percent). Every few loops the goal is sampled to check if the goal can be reached or if a node can be extended in that direction. After a node is sampled, rewiring takes place where the hearest node is taken and nodes in the neighborhood of the node are also taken. The neighborhood is larger that the step size. The node with the least cost is connected to the sampled node other connections are cut. This algorithm is probabalistic complete. In this project i have used RRT extend to grow the tree. The algorithm can be seen in figure 5.



```
Algorithm 6: RRT*
$V \leftarrow \{x_{init}\}; E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ do
    $x_{rand} \leftarrow$ SampleFree$_i$;
    $x_{nearest} \leftarrow$ Nearest($G = (V, E), x_{rand}$);
    $x_{new} \leftarrow$ Steer($x_{nearest}, x_{rand}$) ;
    if ObtacleFree($x_{nearest}, x_{new}$) then
        $X_{near} \leftarrow$ Near($G = (V, E), x_{new}, \min\{\gamma_{RRT^*}(\log(card(V))/card(V))^{1/d}, \eta\}$) ;
        $V \leftarrow V \cup \{x_{new}\}$;
        $x_{min} \leftarrow x_{nearest}; c_{min} \leftarrow$ Cost($x_{nearest}$) + $c$(Line($x_{nearest}, x_{new}$));
        foreach $x_{near} \in X_{near}$ do                // Connect along a minimum-cost path
            if CollisionFree($x_{near}, x_{new}$) $\wedge$ Cost($x_{near}$) + $c$(Line($x_{near}, x_{new}$)) $< c_{min}$ then
                $x_{min} \leftarrow x_{near}; c_{min} \leftarrow$ Cost($x_{near}$) + $c$(Line($x_{near}, x_{new}$))
        $E \leftarrow E \cup \{(x_{min}, x_{new})\}$;
        foreach $x_{near} \in X_{near}$ do                // Rewire the tree
            if CollisionFree($x_{new}, x_{near}$) $\wedge$ Cost($x_{new}$) + $c$(Line($x_{new}, x_{near}$)) $<$ Cost($x_{near}$)
            then $x_{parent} \leftarrow$ Parent($x_{near}$);
                $E \leftarrow (E \setminus \{(x_{parent}, x_{near})\}) \cup \{(x_{new}, x_{near})\}$
return $G = (V, E)$;
```

Fig. 5: RRT* algorithm

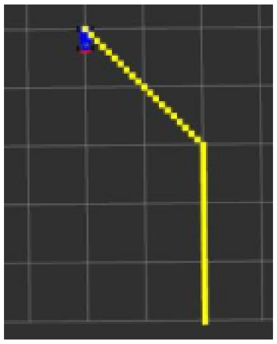
D. *Experiments*

The Gazebo Environment is launched along with the Rviz application. The iterations of each algorithm are done and readings are taken. The algorithms were tested in 3 environments and assessed based on space complexity, time complexity, optimality and completeness. These were then compared to find out which is most suitable.

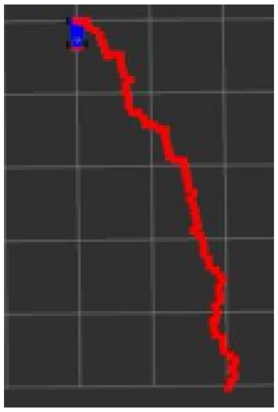
III. RESULTS

The results obtained can be seen in the figures below. The red paths are generated by rrt and yellow paths are generated using astar.

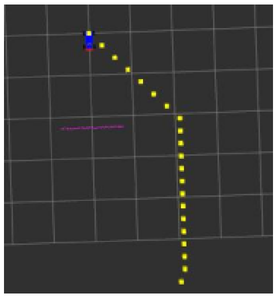
There were some issues with ARA* execution. The program runs well on its own but once interfaced with ROS the ROS node fails to initialize which doesn't happen with the other programs. Attached with this report is a .txt file of the nodes obtained from the program.



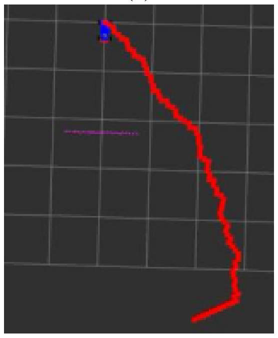
(a)



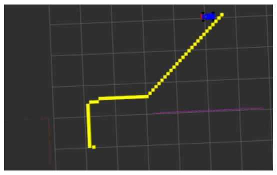
(b)



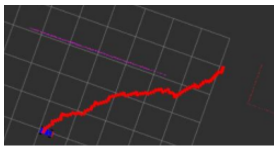
(c)



(d)



(e)



(f)

Fig. 6: results

A. *completeness*

The A* and ARA* guarantee completeness where as the RRT* is only probabalistically complete. The RRT* is a sampling based algorithm and hence will keep sampling nodes until a path is found. It does not return a statement to indicate a path not being there.

B. *optimality*

Due to the fact that RRT extend has been used, it is sub-optimal. the results may have varied if RRT connect had been used. A* is inherently optimal and ARA* becomes optimal once epsilon is 1 as it becomes the A* algorithm.

C. *space complexity*

In general the A* sampled the least nodes followed by the RRT* and then the ARA*. The ARA* returns a path and improves upon it with iterations and hence it is expected to be more space complex.

D. *time complexity*

The time complexity of these was not properly measured as the goals seemed very easy to reach. The ARA* however did take longer to converge to the optimal path but returned a suboptimal path fast.

IV. Conclusion and Discussions

The project gave a very good understanding of different types of algorithms. Since the coding was done from scratch I learned a lot about these algorithms. The A* is the best in terms of most cases but when maps have narrow passages it would be more advisable to go with RRT* as the passage can be reached faster. These algorithms can be further modified to suit specific environments.

V. Future Work

There are many improvements which can be done with this project like

- Building an Octomap to scan the environment and check for collision more efficiently.
- Trying the algorithm for more environments and more goal configurations
- Incorporating non-holonomic constraints. and,
- Using MoveIt! and fcl libraries for more robust collision checking.

References

[1] Hornung, Armin, et al. ”Anytime search-based footstep planning with suboptimality bounds.” Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on. IEEE, 2012.

[2] Arain, Muhammad Asif, et al. ”A comparison of search-based planners for a legged robot.” Robot Motion and Control (RoMoCo), 2013 9th Workshop on. IEEE, 2013.

[3] Nilsson, Nils J. Principles of artificial intelligence. Morgan Kaufmann, 2014.

[4] Hart, Peter E., Nils J. Nilsson, and Bertram Raphael. ”A formal basis for the heuristic determination of minimum cost paths.” IEEE transactions on Systems Science and Cybernetics 4.2 (1968): 100-107.

[5] Islam, Fahad, Venkatraman Narayanan, and Maxim Likhachev. ”Dynamic Multi-Heuristic A.” Robotics and Automation (ICRA), 2015 IEEE International Conference on. IEEE, 2015.

[6] Likhachev, Maxim, Geoffrey J. Gordon, and Sebastian Thrun. ”ARA*: Anytime A* with provable bounds on sub-optimality.” NIPS. 2003.

[7] Kuffner, James J., and Steven M. LaValle. ”RRT-connect: An efficient approach to single-query path planning.” Robotics and Automation, 2000. Proceedings. ICRA’00. IEEE International Conference on. Vol. 2. IEEE, 2000.

[8] Ferguson, Dave, and Anthony Stentz. ”Anytime, dynamic planning in high-dimensional search spaces.” Robotics and Automation, 2007 IEEE International Conference on. IEEE, 2007.

[9] Likhachev, Maxim, and Anthony Stentz. ”R* search.” Lab Papers (GRASP) (2008): 23.