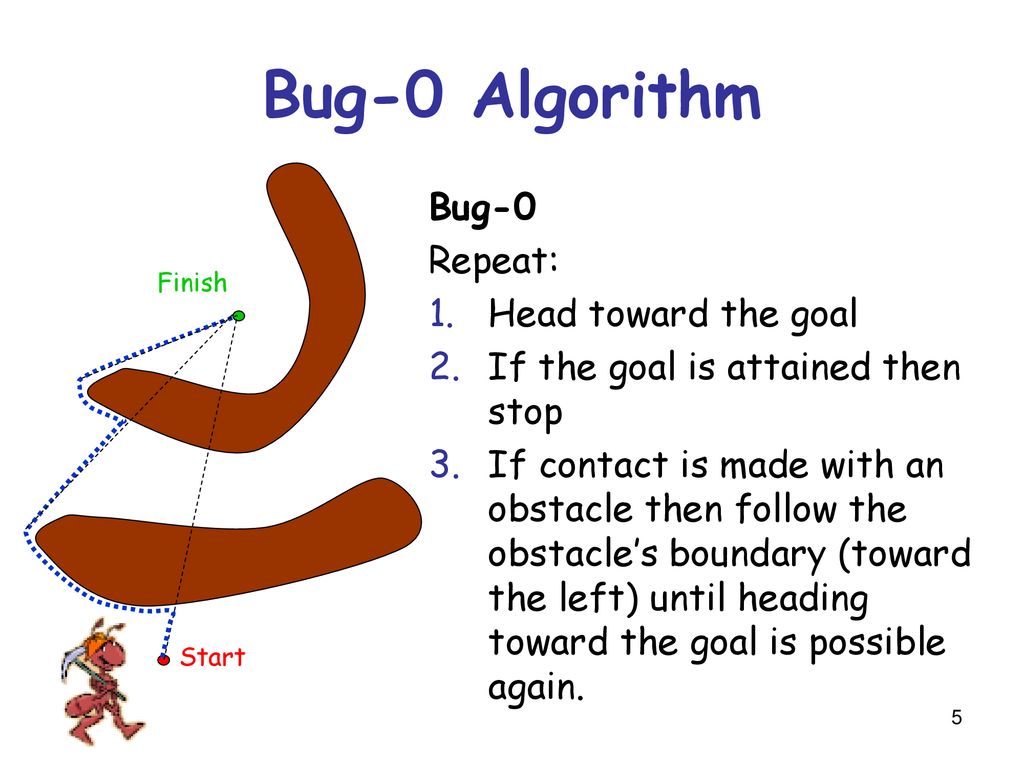
**Obstacle Avoidance and Path Planning**

Ihit R Acharya 16/02/2025

Many obstacle avoidance algorithms are there which utilize the plethora of sensors available for the project. Obstacle avoidance algorithms can be used for building of autonomous traversal vehicles to prevent the vehicle from crashing/damaging itself and for path planning reasons. The algorithms include bug algorithms like bug0, bug1, bug2 or tangent bug or can be Artificial Potential Fields (APF), Dynamic Window Approach (DWA) methods.

**BUG 0:**

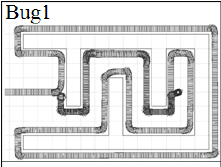
Bug 0 algorithm works by moving towards goal,if an obstacle is avoided it turns leftward while following the boundary of the wall. It continues to follow the wall until it is oriented in direction of the goal and has no obstacle in front of it:



The bug algorithms are simple requiring minimalistic sensors that utilize a lidar for obstacle detection and avoidance. Bug0 tries to circumnavigate the whole obstacle at a time and is an incomplete/unreliable algorithm since plane obstacles could result in a permanent loop for the rover which can cause the failure of the algorithm.

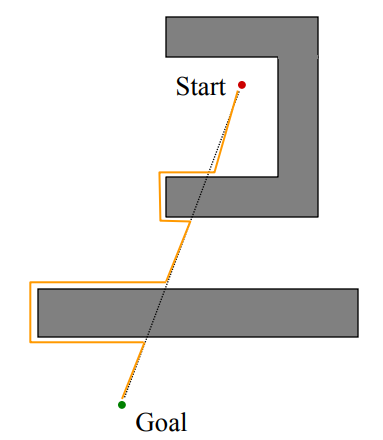
**BUG 1:**

Bug1 circumnavigates the obstacles till it finds the point where it is closest to the goal and stops circumnavigating the obstacle and continues to move towards the goal. To find the point where it is closest to the goal it is required to make a complete rotation about the obstacle before detaching at the point closest to the goal. Bug1 is more effi



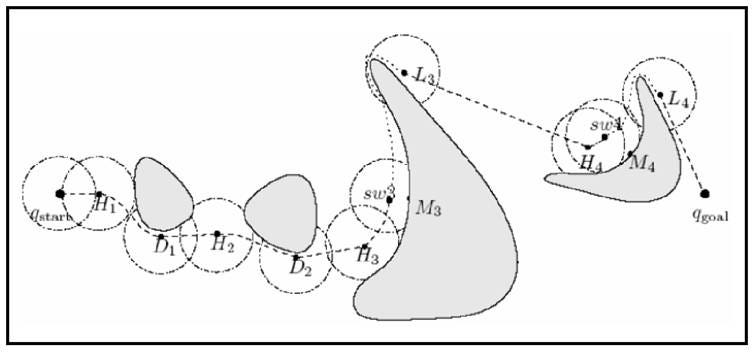
**BUG 2:**

Bug2 is a complete algorithm that moves closely to the wall of the obstacle till it has freedom to move towards its goal. It is an algorithm that is simple and used in obstacle avoidance. It works by first creating a line joining the current coordinate to the end goal coordinate and then moving along this line, if an obstacle encountered it tries to circumnavigate this obstacle by following its wall until it reaches the line again and detaches from the wall, continuing navigation towards the end goal repeating the process..



**TANGENT BUG:**

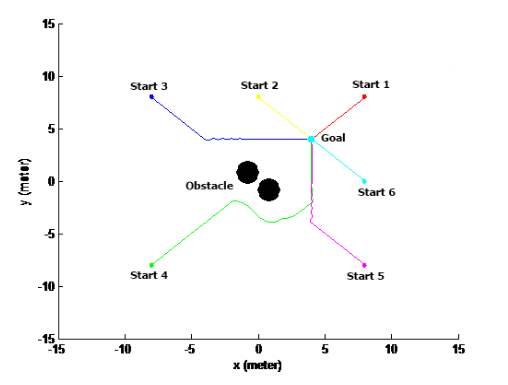
Tangent bug is a bug algorithm that uses the lidar ranges to identify edges/tangents of the obstacle being maneuvered which makes it efficient in obstacle avoidance since it only makes necessary movements while maneuvering the obstacles .



It finds these edges by finding the two adjacent indices of lidar angle where one is very small in comparison to another, and follows by adding an extra buffer angle to compensate for rover body dimensions and makes movement towards the point after which it tries to navigate to the goal repeating the process. This method can face issues with small gaps and perforations in obstacles hence an additional improvement to the code has to be made by making sure there is a long consecutive length of lidar indices long enogh for rover dimension.

**ARTIFICIAL POTENTIAL FIELDS(APF):**

APF is an obstacle avoidance algorithm that marks obstacles with a repulsive force and the goal as attractive force allowing it to dodge obstacles while simultaneously moving towards the goal. However, APF has a disadvantage being that its inability to deal with the local minima problem. APF works by setting markers on the global cost map and accordingly assigning different magnitude of forces and maneuvering the global cost map.



**LOCAL MINIMA PROBLEM:**

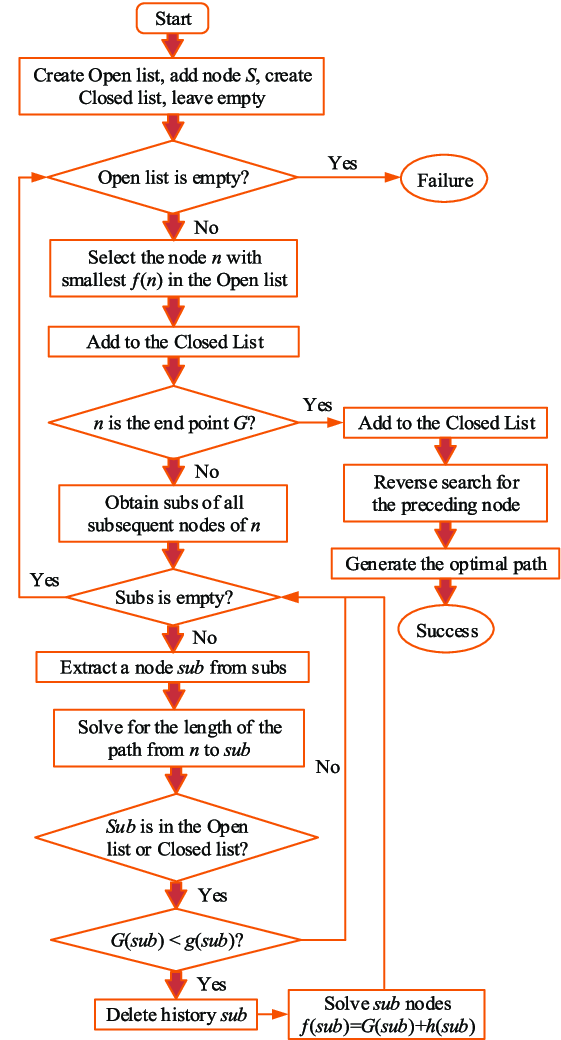
The local minima problem entails a problem where rover is stuck in at the bottom point of a bell curve making it a problem for obstacle avoidance since it can cause rover to be stuck or confused in its movement which can cause failure of the algorithm being utilized.

**Dynamic Window Approach (DWA):**

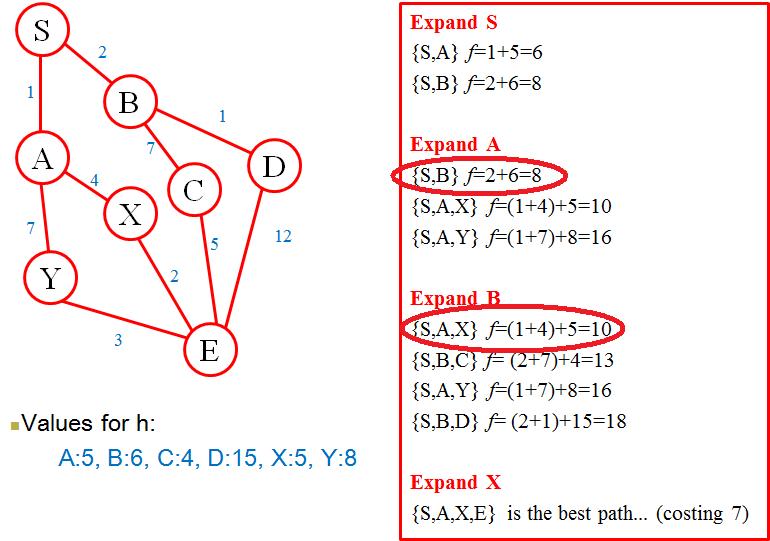
DWA is an algorithm that simulates the possibility of obstacle avoidance by choosing different linear and angular velocity pairs and accordingly choosing the pair which results in no collisions and having heading towards the goal. It allows for dealing with local minima problem since it simulates all the possible velocity pairs and accordingly chooses the fastest and most efficient pair.

**A\* ALGORITHM:**

A\* algorithm finds the shortest path to the goal using the concept of optimization of cost function f(n)=g(n)+h(n) where f(n) is total cost required to reach node ”n” ,g(n) is the cost to reach node “n” from start and h(n) is the approximate cost required to reach end goal. The algorithm chooses the smallest f(n) from the list of possible nodes it can explore (open list) and places the travelled nodes into another list (closed list) after this simulated movement cost is calculated for the remaining nodes that are to be explored.

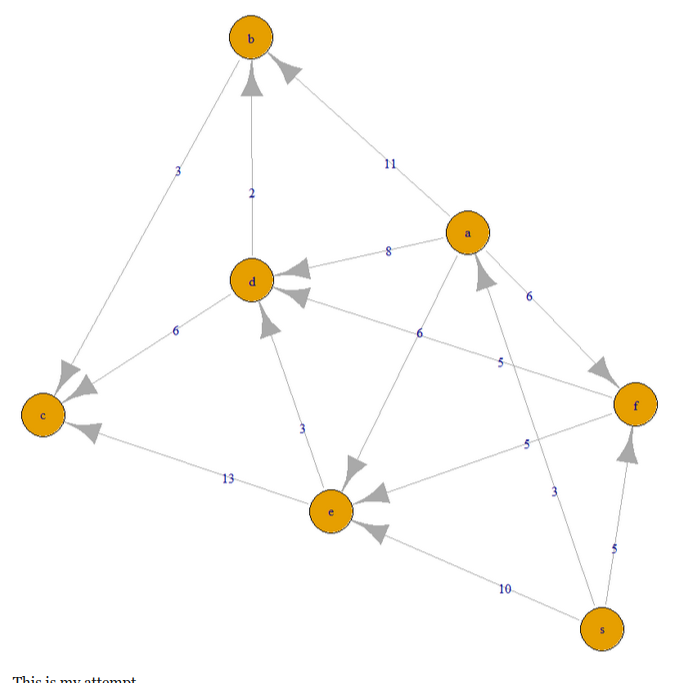


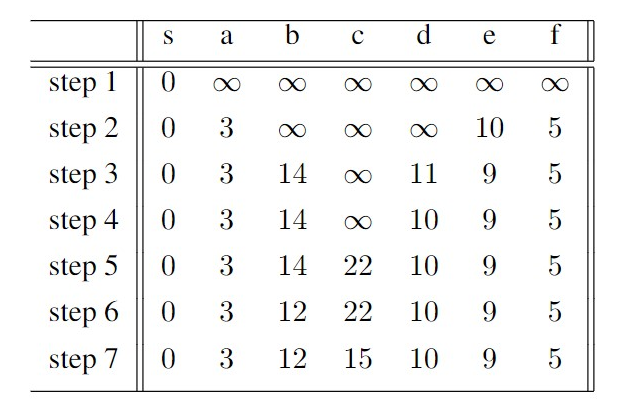
The exploration is then repeatedly continued till the goal has been reached. The h(n) is Euclidian distance between current coords and goal coords. After the goal has been reached, i.e. h(n) becomes zero, the nodes in the closed list are reverse explored to reach the goal in the shortest path possible.

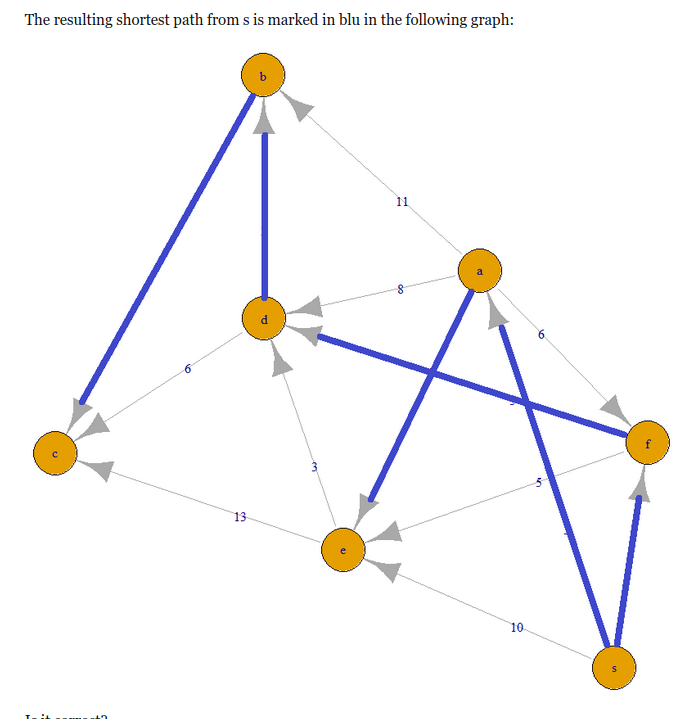


**Dijkstra ALGORITHM:**

Dijkstra works by usage of a priority que where from the current node the shortest distance (weighted edge) in the priority que is picked, after picking of the node it explores its nearest neighbor and computes the shortest path which is then compared with the original distances in the previous que, if a value in the previous que is smaller than the total weighted edge for travelling till the nearest neighbor of the neighbor then that node in the original que is explored , and this process is repeated till it gets the shortest path to the goal node.







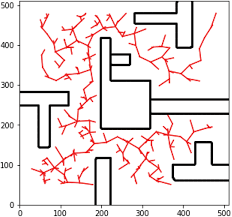
The algorithm can only deal with positively weighted edges but fails with negative weights which are solved using Bellman-Ford Algorithm.

**Bellman-Ford ALGORITHM:**

Bellman-Ford Algorithm can be utilized with negative edge weights (computed cost functions), it works by calculating cost to a neighbor of a neighbor node and tries to find a shorter path to this same node. It is an algorithm that is less efficient than Dijkstra but can deal with negative weights and negative cycles (paths where cost increases by a large margin).

**Rapidly Exploring Random Tree(RRT) and variants:**

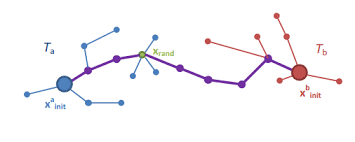
Rapidly exploring Random Tree (RRT) is an n algorithm which is also renowned for its ability to explore a majority of the possible nodes and generate a solution for reaching the goal node from the starting node. It works by first starting at the start node followed by generating random nodes in a specified search vicinity, it then connects the current node to the closest of the random nodes and traverses to it making it part of the branch of the tree, this process is repeated till it reaches the goal node.



There are many variations of RRT algorithm with improvements in the efficiency of the path planning algorithm.

**Hybrid RRT :** Efficient algorithm which is a combination of normal RRT with heuristic algorithms like (A\* or Dijkstra) which act as biases to direct the branching towards the goal node.

**Bidirectional Hybrid RRT : I**nstead of only creating branches from only start it also does so from the goal so that convergence of the two trees would be faster hence leading to a faster algorithm.



**Informed RRT\*** **:** Another variation of normal-RRT but instead of random node sampling around a circular vicinity of the autonomous vehicle it instead has an elliptical sampling area that is biased towards the direction of the goal which can allow it only choosing the optimum random nodes resulting in faster convergence. This variant of RRT can be utilized with Bidirectional-RRT and heuristic-RRT to get even faster convergence of the two trees allowing for even faster and optimized convergence by using insights from heuristic RRT combined with the search region optimization of Informed-RRT. RRT can also be further improved using neural networks to further reduce the list of possible random nodes that are needed to be placed by RRT.

**GREEDY ALGORITHMS:**

Greedy algorithms tend to rely on choosing the minimal cost function nodes with the probability of finding the goal. A\* and Dijkstra are examples of greedy algorithms showing their reliance on minimization of cost function thus finding optimal solution to the goal node. Greedy Best-First Search (GBFS) is an algorithm that is based on only minimalization of the heuristic cost function thus helping them to find the solution till the goal node quickly, however the path generated by this algorithm is highly unoptimized since it’s only taking into consideration the heuristic costs which only signify distance to goal node from current node. Voronoi-Based Greedy Path Planning is an algorithm that chooses the safest path by choosing the safest/largest gaps between obstacles which allows it to utilize the safest path to the goal which can help in certain case scenarios where the autonomous vehicle needs to traverse a highly saturated obstacle course.

**Randomized Depth-First Search (RDFS) and Randomized Breath-First Search (RBFS):**

Randomized Depth-First Search (RDFS) and Randomized Breath-First Search (RBFS) are also methods that are utilized to completely traverse a path till it reaches a dead end and methodically traversing each path by equal amounts respectively. The following algorithms however tend to be very inefficient and time consuming due to their brute force approach without adding modifications to the algorithms.

**Fuzzy Logic ALGORITHM:**

Fuzzy logic algorithm works by converting distance into less specific and general data like close, moderate distance, and far and accordingly giving it kinematic vector commands to traverse the obstacles, this method allows for smoother and gradual movements instead of sudden rotations and movements which can be crucial for real life applications where fast turns and movements are tough to execute in practical case scenario.