CO-PRM: Consensus Algorithm based Optimization of Probabilistic Roadmap Planner

Tomoki Koike

I. Introduction and Motivation

Motion planning and obstacle avoidance has been a mainstream topic in the field of robotics for many years. The task of going outside and walking on the streets seems to be simple for human beings, however, for autonomous robots to perform this requires considerations in many aspects such as sensors, computer vision, object identification, path planning, object avoidance, trajectory tracking, etc. and can be significantly complex to accomplish in reality. Recent applications range broadly as unmanned aerial vehicles (UAV), home appliances (home cleaners), logistics, autonomous cars, and satellites. Many of which have gained more attention for becoming more interactive and adjacent with people's lives in the shape of consumer products and technology that benefit the population [1].

In particular, motion/path planning has been researched to find many distinct methods to optimize the mobility of robots while avoiding static or dynamic obstacles. Apart from the characteristics of the obstacles there are other concerns to acknowledge, for example, the nature of the environment in which the robot traverses and computational expensiveness. Considerations from many facets give rise to countless types of motion planning algorithms and each of them are catered towards specific circumstances and bound by assumptions. In general, motion planning will have a starting point and goal point or some node and waypoint to connect the path to desired locations, however they diverge depending on the conditions as aforementioned. In this section, a brief introduction of existing motion planning algorithms will be discussed in detail and eventually narrowed down to one specific algorithm referred to as the *Probabilistic Roadmap Planning* (PRM).

A. Dijkstra

This is an algorithm which finds the optimum shortest path within a given field with known starting and goal points. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later [2]. The search region will expand concentrically from the starting point due to the algorithms nature of searching paths based on a minimum-cost finding strategy [3].

Algorithm 1 Dijkstra's Shortest Path First Algorithm

```
Input: Graph $G$, Source $S$
Output: Set of Distances $D$, Previously Stored Vertices $P$
Function Dijkstra$(G, S)$
    initialize set of vertex Q
    for each vertex $v$ in $G$ do
        $D[v] \leftarrow$ INF
        $P[v] \leftarrow$ NULL
        add $v$ to $Q$
    $D[S] \leftarrow 0$
    while $Q$ is not empty do
        $u \leftarrow$ store vertex in $Q$ with MIN $D[u]$
        remove $u$ from $Q$
        for each neighbor $v$ (still in $Q$) of $u$ do
            $alt \leftarrow D[u] + length(u, v)$
            if $alt < D[v]$ then
                $D[v] \leftarrow alt$
                $P[v] \leftarrow u$
    return $D$, $P$
```

The algorithm is broken down into 4 steps

1) Designate the starting point (Source S as in above)
2) Create an empty set for the vertices
3) Assign a distance value to all vertices of the graph G in which they are initialized as infinity. Assign 0 for the source vertex.
4) While Q is not empty
   - Calculate distances (movement cost) from starting point to vertex that is not stored in Q and pick the vertex u with the shortest distance.
   - remove u in Q.
   - Update distance of all neighbor vertices of u by iterating through all neighboring vertices. For all neighbor vertex v, if the sum of

distance of u and length of the edge joining of u and v is less then the distance of v we update the distance of v with this sum value. Then we update the previous history of v with u.

The Dijkstra algorithm conducts a holistic search through all the possible paths with cost calculations and this results in poor efficiency and long search time depending on the distance from the starting point to the destination [3].

*B. A**

The A* algorithm is a algorithm that is built on top of the Dijkstra algorithm but has a special feature that the predecessor does not. It is considered one of the best techniques for path-finding and graph traversals. This algorithm was originally presented by Hart *et al* (1968) and what it made it stand out from the Dijkstra was the Heuristic approach taken by estimating the length to complete the trajectory to the desired goal [4] [5]. For each iteration within the Dijkstra algorithm, the A* algorithm computes the distance value $d(i)$ which consists of two parts. One being the approximate shortest distance from the origin to node *i* ($g(i)$) and the other being the approximate shortest distance from node *i* to the goal ($h(i)$. Then the total distance estimate $d(i) = g(i)+h(i)$ is employed as a cost/loss function that determines which path to take in order to minimize the total cost $d(*)$. The $g(i)$ distance is straightforward, however, considering multiple possibilities of reaching the goal point amidst the process of planning the optimum path, a heuristic is adopted to approximate the cost of $h(i)$. There are many methods to this and the three major ones are the Manhattan distance, diagonal distance, and Euclidean distance.

- Manhattan distance: The very right in Fig. 1. The sum of absolute values of x- and y-coordinates.
- Diagonal distance: The middle image in Fig. 1. The MAX$\{|x|, |y|\}$.
- Euclidean distance: The left in Fig. 1. The exact distance from a point to the destination.

This improvement made eliminates the futile searches that cause low efficiency with high loads of computation.

C. Rapidly Exploring Tree (RRT)

The third algorithm is a prestigious randomized planning method adopted in many path planning applications. This algorithm is effective for nonholonomic constraints with few heuristics and arbitrary parameters. The algorithm was originally postulated by Steven LaValle (1998) and is famous for being practical for versatile conditions involving obstacles in a controlled environment which either could be 2- or 3-dimensional.

The RRT first instigates by generating a myriad of random points in the defined search space. Then a connection is attempted between the starting point to the nearest point while conducting obstacle clearance. When the connection is made that means the newly generated point is within the free space and not in the obstacle region. The state of the tree is then updated with a state transition equation, and the RRT expands its state proportional to the Voronoi region [6]. The growth of the tree is controlled by a growth factor or rate, and furthermore, the tree extends in a biased manner in order to be guided towards the destination with the probability of sampling. The higher the probability is the more greedily the expansion maneuvers towards to planned goal. Once a route from the starting point to the goal is connected, the algorithm back tracks the optimal path that has already determined the closest path and back propagates the trajectory nodes as the final result like in the example of Fig. 2. The parameter ρ denotes a state space distance metric which defines the termination condition for the iterative process. Additionally, there can be many other constraints aside from obstacles like path angles. The algorithm can be described as follows [7].

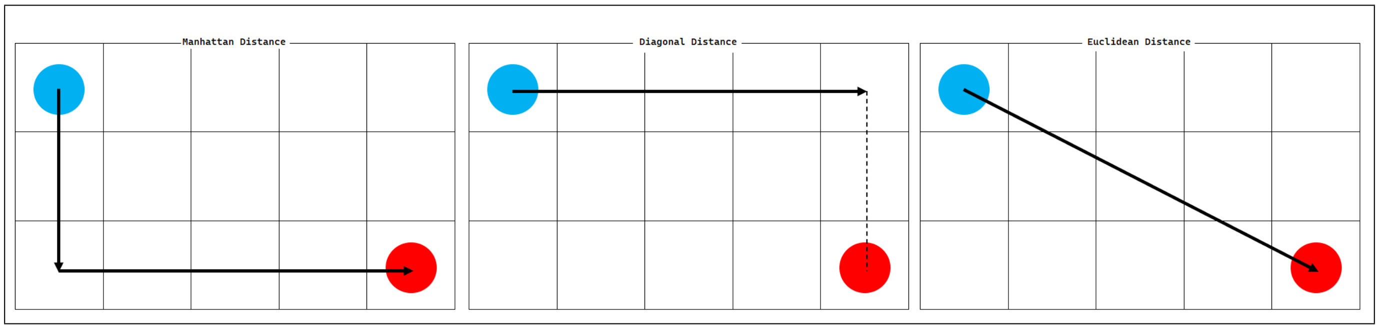

Fig. 1: Three methods of distance approximation of A^*.

Algorithm 2 RRT Path Planning

```
Input: Initial configuration $q_{init}$, number of vertices $K$,
       incremental distance $\Delta q$
Output: RRT Graph $T$
Function RRT($q_{init}, K, \Delta q$)
    $T$.add($q_{init}$)
    for $k$ = 1 to $K$ do
        $q_{rand}$ = random configuration
        $q_{near}$ = nearest neighbor in tree $T$ to $q_{rand}$
        $q_{new}$ = extend $q_{near}$ toward $q_{rand}$ with increment
          of $\Delta q$
    if $q_{new}$ can connect to $q_{near}$ then
        $T$.addVertex($q_{new}$)
        $T$.addEdge($q_{near}$,$q_{new}$)
    if $\rho(q_{new}, q_{goal}) <$ distance to Goal then
        BREAK
    return $T$
```

The disadvantage of the RRT derives from the nature of how the graph expands. Intrinsically, the tree branch expansion has randomness in their length as well as direction. Although the structural nature contributes the brevity of the computation, the randomness as well as the number of newly generated vertices are the limiting factor of whether the path planner obtains a optimum solution.

Provided the overview of Shortest Path Algorithms (SPA), it is evident that the all of the discussed algorithms combine the task of generating possible nodes, check to see if nodes satisfy the constraints, compute heuristics, and expand the graph. These step-by-step tasks are stacked inside an iteration and conflate the calculations leading to a failure of obtaining a solution. This issue is tackled with the method of introducing an auxiliary path generator like the Probabilistic Roadmap (PRM) planner which constructs a graph with collision free points and edges [7]. Then with a preconstructed graph, the computations to find an optimal path through that it will be highly efficient using preexisting and simple path planners such as Dijkstra and A*. However, PRM has its own flaws and will require an optimization to acquire a higher performance from splitting the conventional tasks of SPA into two separate steps of constructing the graph and generating the optimum path.

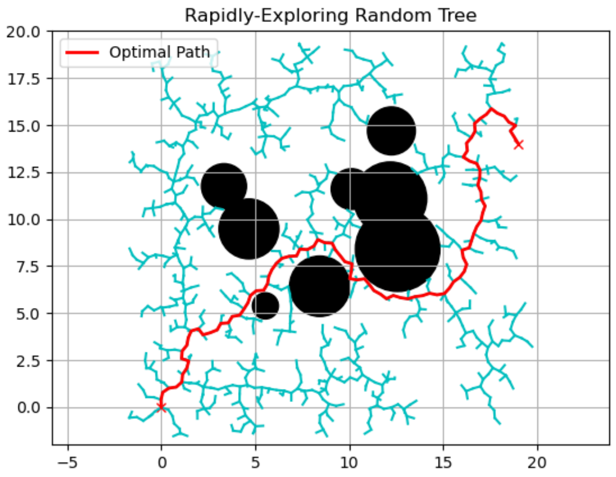

Fig. 2: RRT path planning example.

II. Problem Formulation

To establish the premise of utilizing PRM Planner for a reformed path planning method, it is vital to elucidate the theories that will be involved. The two essential theories are the PRM Planner algorithm and consensus theory. The former is the foundation of the problem formulation, and therefore, is inevitable to expound. The latter will be applied to optimize the graph construction of PRM. In the following sections we will give an overview of PRM Planner, the refined PRM algorithm known as PRM*, and the proposal of a consensus optimized PRM* algorithm (CO-PRM*).

A. *Probabilistic Roadmap Planning (PRM)*

Similar to the RRT algorithm and other path planners, the PRM is based on the notion of a configuration space or search space $\mathcal{C}$. The dimension of this space is defined by the degree of freedom of the objects and the object in the space is transformed into an obstacle. This obstacle indicates the forbidden part of the configuration space $\mathcal{C}_{forb}$. A generated path in the space is collision free if there is no intersection with the $\mathcal{C}_{forb}$. This is equivalent for having all the nodes of the constructed path be within the completely free part of the configuration space denoted as $\mathcal{C}_{free}$ [8].

Many PRM methods have been proposed but all follow the same concept of sampling the configuration space for a collision free space and attempts to connect the configurations into a roadmap. In the free space $\mathcal{C}_{free}$, the PRM selects a series of configurations which consist of a graph $G = (V, E)$ where V are the vertices and E are the edges of the graph. The planner samples a pair of nodes with a local algorithm and connects the configurations with a path. When this is successful, a new edge is appended to the graph. Conventionally, the network structuring between configurations will be done with an interpolation method and will be checked depending on an intersection with $\mathcal{C}_{forb}$. The node sampler is strategic in that it is rewarded when the sampler generates a useful point and is penalized otherwise [9].

To be clear, the connection of configurations are, in other words, a cluster of nodes that can be joined together depending on a certain metric such as distance and other design costs. If the metric is particularly the distance, the nodes within a certain radius will be clustered together. The pseudo-code is presented below [8].

Algorithm 3 Probabilistic Roadmap Construction

```
Input: $\mathcal{C}_{\{\nabla\rceil\rceil}$, $\mathcal{C}_{forb}$, limit of number of nodes $N$
Output: $G$
Function $\mathcal{C}_{free}$, $\mathcal{C}_{forb}$, $N$
    $G(V,E) \leftarrow$ = NULL
    for iteration in $[0...N]$ do
        $c \leftarrow$ a configuration in $\mathcal{C}_{free}$ that satisfies the metric
        $V \leftarrow V \cup \{c\}$
        $N_c \leftarrow$ a set of nodes from $V$ that satisfy a metric
        for all $c' \in N_c$, in increasing distance order from $c$ do
            if $c'$ and $c$ are not connected already in $G$ then
                if the local planner identifies a path for $c'c$ then
                    $E \leftarrow c'c$
                    $G(V,E) \leftarrow N_c$ and $E$
    return $G(V,E)$
```

The disadvantages of this algorithm is that having too few configurations result in a fractured graph and clusters may remain disconnected especially in high density obstacle regions. With few number of nodes generated, paths are sparsely formed and include crucial level of irregularity in the paths which is undesirable in realistic conditions.

B. PRM*

The PRM* algorithm is a refined version of PRM which implements a different local planner or clustering method that depends on the number of nodes generated. In the original PRM algorithm, the clustering is determined by a simple radial distance metric. However, in PRM* the distance metric is based on a logarithmic function metric. That is, the connection radius is scaled by a specific logarithmic function with the variable being the number of nodes.

$$R(n) := \gamma_{PRM}\Big(\frac{log(n)}{n}\Big)^{1/d} \tag{1}$$

where

$$\gamma_{PRM} > \gamma^*_{PRM} = 2\Big(1+\frac{1}{d}\Big)^{1/d}\Big(\frac{\mu(\mathcal{C}_{free})}{\zeta_d}\Big)^{1/d}.$$

In this equation, n is the number of nodes, d denotes the dimension of the search space $\mathcal{C}$, $\mu(\mathcal{C}_{free})$ is the Lebesgue measure (i.e. volume) of $\mathcal{C}_{free}$, and ζ_d indicates the volume of the unit ball in the d-dimensional Euclidean space [10]. The radius function, $r(n)$ is visually illustrated in Fig. 3.

C. Algorithm Proposal

From the analysis of Karaman and Frazzoli, the PRM and PRM* algorithms are categorized based on 5 components: probabilistic completeness, asymptotic optimality, monotone convergence, time complexity, and space complexity. Probabilistic completeness is the property that quantifies the probability of the planner failing to find an optimum path based on the expensiveness of the computation. If a path exists, it asymptotically approaches zero. A probabilistically complete method has its performance measured by the rate of convergence. The second component, asymptotic optimality defines that for a given algorithm *ALG*, if the worst case running time is slower than a general lower bound of comparative algorithms then the *ALG* is not optimal. Whereas, if it performs better than the lower bound it is optimal.

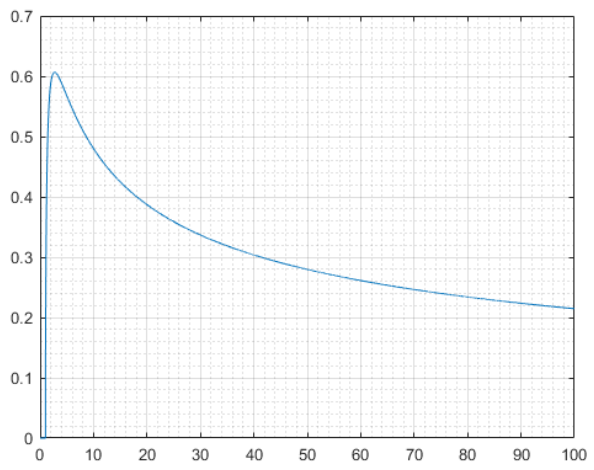

Fig. 3: Demonstration of log(n)/n in 2D search space

Monotone convergence is defined to be when the *ALG* is increasing its sequence by iteration and is bounded above by a supremum, then it will converge to the supremum. Similarly, when the *ALG* sequence decreases and bounded below by a infimum, then it will converge to the infimum. The remaining two can be interpreted literally as the time- and dimesion-wise computation complexity of the *ALG*.

Based on these measures, Karaman and Frazzoli postulate that PRM satisfies probabilistic completeness and monotone convergence but not asymptotic optimality, and it has a time complexity of $O(n\log n)$ and space complexity of $O(n)$. On the contrary, PRM* is probabilistically complete and asymptotically optimal but not monotonously convergent with the same time complexity as PRM and different space complexity of $O(n\log n)$ [10]. For algorithms that are probabilistically complete but not asymptotically optimal, the number of samples generated will not have the solution cost, if one exists, converge to the optimal cost. This is applicable to the PRM algorithm. Whereas the PRM* overcomes the deficiency of being asymptotically optimal; however, the scaling techniques are limiting and is debatable in terms of computational efficiency.

From the nature of PRM algorithms and others like RRT, it is interesting to observe the common characteristic of random geometric graphs. Such graphs show phase transition phenomena including percolation and connectivity. Percolation refers to the detachment of edges between nodes and the reformation of clusters during the network construction. Karaman and Frazzoli concluded that percolation and connectivity are a crucial factor of probabilistic completeness in that for a sampling-based path planning algorithm, it is only probabilistically complete if the random geometric graph percolates and is asymptotically optimal if and only if the graph is connected [10].

The main objective of this paper emanates from the idea of convolving consensus theory with PRM-based path planning to construct an algorithm that guarantees probabilistic completeness and asymptotic optimality with consensus based theories revolving around graph theory, connectivity, consensus algorithm and so forth. The paper organization for following sections will be noted below for clarity.

D. Paper Organization

1) Implement graph theory to define the connectivity of random and dynamic geometric graphs generated by point sampling.
2) Develop the heuristics with loss/cost function that becomes the objective of convergence for the random geometric graph.
3) Devise an algorithmic method of performing percolation and connection with (node) state updates using consensus algorithms.
4) Simulate the algorithm and retrieve results.
5) Analyze the results and investigate further improvements.

References

[1] Yu Wu, Jinzhan Gou, Xinting Hu, and Yanting Huang. A new consensus theory-based method for formation control and obstacle avoidance of uavs. *Aerospace Science and Technology*, 107:106332, 2020.

[2] E.W. Dijkstra. A note on two problems in connexion with graphs, journal =.

[3] M. Noto and H. Sato. A method for the shortest path search by extended dijkstra algorithm. In *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics. 'cybernetics evolving to systems, humans, organizations, and their complex interactions' (cat. no.0*, volume 3, pages 2316–2320 vol.3, 2000.

[4] Nilsson N.J. Hart, E.P. and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on System Science and Cybernetics*, 4:100–107, 1968.

[5] W. Zeng and R. L. Church. Finding shortest paths on real road networks: the case for a*. *International Journal of Geographical Information Science*, 23(4):531–543, 2009.

[6] Steven M. LaValle. Rapidly-exploring random trees: A new tool for path planning. *Iowa State tech report*, 1998.

[7] Vojtěch Vonásek, Jan Faigl, Tomáš Krajník, and Libor Přeučil. Rrt-path – a guided rapidly exploring random tree. In Krzysztof R. Kozłowski, editor, *Robot Motion and Control 2009*, pages 307–316, London, 2009. Springer London.

[8] Roland Geraerts and Mark H. Overmars. *A Comparative Study of Probabilistic Roadmap Planners*, pages 43–57. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[9] Xiaowen Yu, Yu Zhao, Cong Wang, and Masayoshi Tomizuka. Trajectory planning for robot manipulators considering kinematic constraints using probabilistic roadmap approach. *ASME. J. Dyn. Sys., Meas., Control*, 139(2):846–894, 2016.

[10] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.