2019 International Conference on Robotics and Automation (ICRA)
Palais des congres de Montreal, Montreal, Canada, May 20-24, 2019

Kinematic Constraints Based Bi-directional RRT (KB-RRT) with Parameterized Trajectories for Robot Path Planning in Cluttered Environment

Dibyendu Ghosh[1],Ganeshram Nandakumar[1],Karthik Narayanan[1],Vinayak Honkote[1]*and* Sidharth Sharma[2]

Abstract— **Optimal path planning and smooth trajectory planning are critical for effective navigation of mobile robots working towards accomplishing complex missions. For autonomous, real time and extended operations of mobile robots, the navigation capability needs to be executed at the edge. Thus, efficient compute, minimum memory utilization and smooth trajectory are the key parameters that drive the successful operation of autonomous mobile robots. Traditionally, navigation solutions focus on developing robust path planning algorithms which are complex and compute/memory intensive. Bidirectional-RRT(Bi-RRT) based path planning algorithms have gained increased attention due to their effectiveness and computational efficiency in generating feasible paths. However, these algorithms neither optimize memory nor guarantee smooth trajectories. To this end, we propose a kinematically constrained Bi-RRT (KB-RRT) algorithm, which restricts the number of nodes generated without compromising on the accuracy and incorporates kinodynamic constraints for generating smooth trajectories, together resulting in efficient navigation of autonomous mobile robots. The proposed algorithm is tested in a highly cluttered environment on an Ackermann-steering vehicle model with severe kinematic constraints. The experimental results demonstrate that KB-RRT achieves three times ($3X$) better performance in terms of convergence rate and memory utilization compared to a standard Bi-RRT algorithm.**

I. INTRODUCTION

Efficient navigation is a critical capability required for mobile robots operating in unknown and cluttered environments. Efficient navigation demands optimal path planning and smooth trajectory planning. *Autonomous* navigation involves self steering of a mobile robot from a start location to a target location using the compute and memory resources on-board at the edge. Further, to accomplish complex missions and to enable time critical functionalities on *autonomous mobile robots*, the navigation capabilities need to run in *real time, at the edge*. These mobile robots need to operate for extended period of time (at least, for a few hours) to accomplish complex missions such as, search and rescue, surveillance, mapping unknown area etc. The real time operation and extended run time requirements demand energy efficient compute, minimum memory utilization and smooth navigation on each autonomous mobile robot.

Over the last two decades, several flavors of path planning techniques have been developed, to account for constraints in sensors, environmental effects, and motion capabilities. Among them, sampling based path planners (such as, RRT [7]), are in general considered more suitable for planning in continuous domains as they do not require specific discretization of the environments. However, these planners generally focus on finding a feasible trajectory, rather than minimizing the path cost. Thus, the paths computed are often sub-optimal and inconsistent [13]. In recent years, different variants of RRT that give solutions for asymptotically near-optimal motion planning have been developed. These solutions generally exhibit a trade-off between computational time and path cost. Closed-loop rapidly-exploring random tree (CL-RRT) [9] proposed an extension to RRT that samples an input to a stable closed-loop system consisting of the vehicle and a controller, allowing the robot to traverse the path with kinematic constraints [9]. Other variants of RRT such as, RRT* [6] and RRT-connect, help in optimizing the path length, however, fail to give optimal paths in cluttered environments when taking kinematic constraints into consideration. Trajectory parameter-space (TP-space) RRT [1] is a new RRT variant that inherently incorporates the exact vehicle shape and vehicle kinematic constraints. However, this method is based on trajectory based space configuration and does not optimize the compute (number of RRT nodes) and memory utilization on each robot. Separately, many researchers investigated the use of kinodynamic constraints in sampling-based algorithms owing to their effectiveness with high dimensional spaces ([4] [5] [10] [12] [16] [8]). In general, for kinodynamic planning, the main focus is on developing efficient steering functions, which correspond to a local trajectory generator neither taking into account the obstacles nor considering RRT node generation. In this paper, we present a planning technique based on Bi-RRT, taking into consideration the kinematic model of the robot to find an asymptotically near-optimal path [2] in a highly cluttered environment. Contrary to the conventional methods of selecting purely random nodes, the addition of kinematic constraints restricts the tree expansion to feasible regions of the workspace. Such a design not only enhances path optimization but also decreases the overall compute requirement and memory utilization. Finally, inclusion of kinodynamic constraints in cost function based on asymptotically near-optimal path guarantees smooth trajectory for navigation.

The remainder of the paper is organized as below. The preliminaries including space configuration and robot motion models are introduced in Section II. Proposed methodology

[1]Silicon Technology Prototyping Lab, Intel Labs, Bangalore, India `dibyendu.ghosh@intel.com`, `ganeshram.nandakumar@intel.com`, `karthik1.narayanan@intel.com`, `vinayak.honkote@intel.com`
[2]IIIT, Delhi, India `sidharth14163@iitd.ac.in`

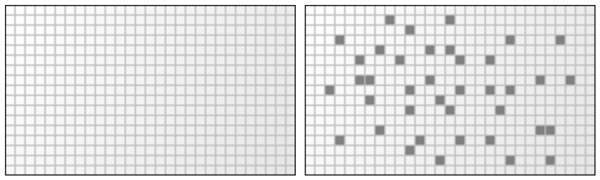


Fig. 1: (a) Grid map structure, (b) Occupancy grid map where black color indicates the obstacles present in particular cells

for KB-RRT is described in Section III. Experimental setup and results are discussed in Section IV. Conclusions are presented in Section V.

II. Preliminaries

In this section, the preliminaries are described. In Sections II-A and II-B, the space configuration (description of environment in C-space ([2], [11])) and robot motion model concepts are introduced.

A. Space Configuration

The occupancy grid mapping algorithm works on the grid map, which is essentially a multidimensional random field representing the occupancy state of the cells in a spatial lattice [3]. This map is generated by dividing the real environment into many equal-sized small cells (each cell is equivalent to robot size). Each cell holds the probability values of occupancy. Occupancy values range from zero to one. An occupancy value of zero (i.e., $P(Cell = OCC) = 0$), corresponds to an open cell and indicates that the region is free from any obstacle. An occupancy value of one (i.e., $P(Cell = OCC) = 1$), indicates the occupied cell. In the absence of any prior knowledge about the obstacle, the occupancy value is set to 0.5 indicating the equally likely probability of presence of an obstacle in the corresponding cell. The grid cells are then updated iteratively using sensor data fusion to construct a map of the environment with the probability of presence or absence of obstacles in the corresponding region. Fig. 1(a) and 1(b) present an example case of generating occupancy grid map using occupancy grid mapping algorithm from a normal grid map. The black cells in Fig. 1(b) represent the occupied cells while the white ones are empty cells.

B. Robot Motion Model

The vehicle kinematic model approximates the mobility of a car. The non-holonomic nature of the car is related to the assumption that the car wheels roll without slipping. We consider an agent model whose state evolves as per a non-linear discreet time dynamical system:

$$x(t+1) = f(x(t), u(t)) \tag{1}$$

where $x \in R^n$ is the state at time t, $f : R^n \times R^n \rightarrow R_n$ describes the state transition map of the system and $u = \begin{bmatrix} v \\ \omega \end{bmatrix}$ is the control input exercised for the time δt, where v and ω are the instantaneous linear and angular velocities, respectively. Further, we define δs as the total displacement of the robot while applying the control input u. We assume that the vehicle is moving on a level surface and can be modeled as an Ackermann drive with kinematics as:

$$\begin{aligned} \dot{x} &= dx/dt = v * cos(\theta) \\ \dot{y} &= dy/dt = v * sin(\theta) \\ dx\sin(\theta) &- dy\cos(\theta) = 0 \\ \dot{\theta} &= d\theta/dt = (v/L) * tan(\phi) \end{aligned} \tag{2}$$

where $X = (x, y, \theta)$ represents the vehicle pose and L is the distance between the rear and front wheel axles. We consider a continuous control space to exploit the maximum knowledge of the non-holonomic constraints. In Fig. 2, Ackerman-steering model of car-like robots is depicted, where ϕ and W are the instantaneous steering angle and width of the robot, respectively.

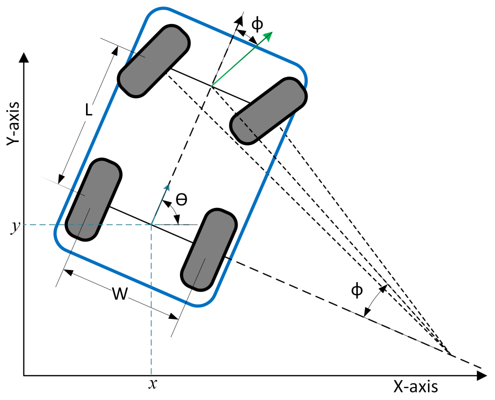


Fig. 2: Ackerman-steering model of car-like robots

III. Methodology

In this section, the proposed methodology for path planning is described. The path planning algorithm builds on the motion model described in Section II-B. These are interdependent and need to work hand-in-hand for determining an optimal path. In Section III-A, the algorithmic details of KB-RRT are presented. In Section III-B, the trajectory generation aspects are described.

A. KB-RRT with Robot Motion Model

A conventional RRT planner employs a sample based approach. It was originally devised to handle systems with large dimensional configuration spaces. In this procedure, a random state is selected from the state space, where these states are assumed to be independent and identically distributed (i.i.d.) [14]. Since the return state is purely random, the number of *nodes* in the state space can increase exponentially in a cluttered environment. Fig. 3 represents the node generation using a conventional RRT without considering any kinematic constraints..

In the proposed Kinematic Bi-Directional RRT (KB-RRT) algorithm, the expansion of RRT is restricted to feasible regions of the state space optimizing the path itself and improving the computation required to generate the path.

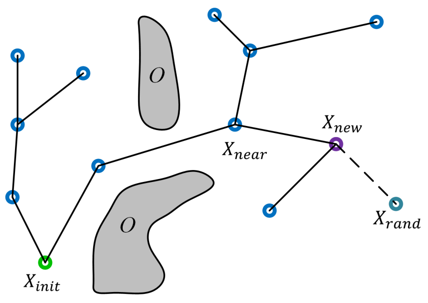
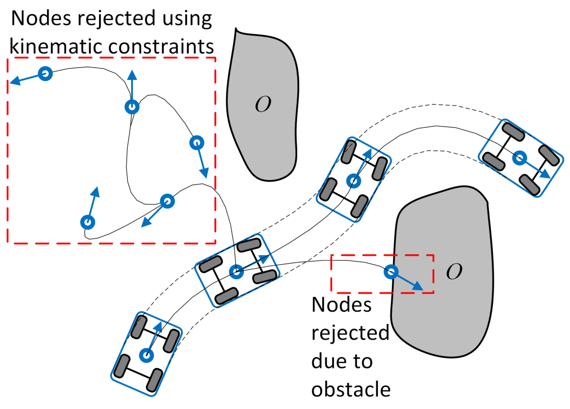


Fig. 3: Node generation using generic RRT

The process of incorporating the kinematic constraints of the vehicle to the node generation process is termed as Kinematic Constraints based Random State Search (KCRSS). We delve into details of both these algorithms in the subsequent sections. In this approach, the path planning problem is formulated using the starting state of the robot (x_s, y_s, θ_s), the end state (x_g, y_g, θ_g), the obstacle co-ordinates (O_x and O_y), the instantaneous vehicle velocities and the dimensions of the vehicle. We assume an underlying grid where each cell is a square of a predefined size. A cell is assumed to be navigable if there are no obstacles inside its boundary, else, it is termed blocked. The obstacle set O includes all such blocked cells. Including kinematic constraints on the random node generation lets us avoid unnecessary growth of the RRT. In Fig. 4, the node selection using KB-RRT algorithm is depicted.

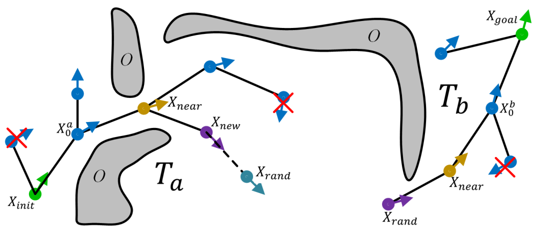


Nodes rejected using kinematic constraints
O
O
Nodes rejected due to obstacle

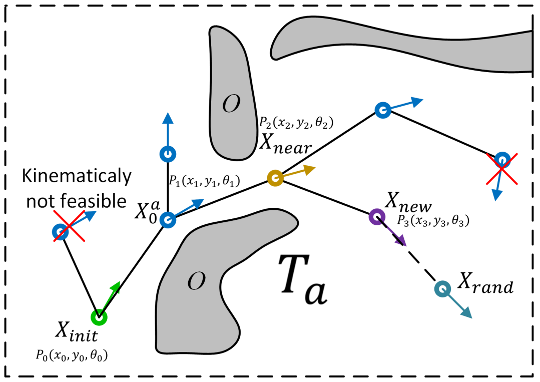
Fig. 4: Representation of node selection using KB-RRT

The KB-RRT algorithm starts by growing two RRTs, one from X_{init} and another from X_{goal}, where X_{init} is the start state and X_{goal}, the destination state of the robot. This is depicted in Fig. 5a. The two trees T_a and T_b (starting from X_{init} and X_{goal} respectively) are maintained at all times until they become connected and a solution is found. In each iteration, a random point X_{rand}, is generated and the nearest neighbor from the tree to X_{rand} is termed as X_{near}. Further the kinematic feasibility of the state transition from X_{near} to X_{rand} is verified using KCRSS as described next.

KCRSS is initiated by passing the two states, X_{near} and X_{rand} to the steering function, which in turn applies non-holonomic control strategies to generate a trajectory between



(a) Generated trees, T_a and T_b using KCRSS



O
$P_2(x_2, y_2, \theta_2)$
X_{near}
$P_1(x_1, y_1, \theta_1)$
Kinematicaly not feasible
X_0^a
X_{new}
$P_3(x_3, y_3, \theta_3)$
O
T_a
X_{rand}
X_{init}
$P_0(x_0, y_0, \theta_0)$

(b) Close up diagram of T_a showing node generation

Fig. 5: Node generation using KCRSS

the two states. Algorithm 1 defines the steering function used to determine the feasibility of connecting a path segment from X_{near} to X_{rand}. The steering function starts with the state configuration of X_{near} and generates a series of control steps to steer it towards X_{rand}. The steering functions are utilized to follow the path given by the way-points under non-holonomic constraints. It continuously computes the new steering angle and updates the position and heading of the vehicle for a time δt, which is the resolution used by the steering function.

In addition to the above constraints, the position of the newly generated vehicle state is inspected to assess whether it falls under the obstacle cell set O. If it does fall under O, a path from X_{near} to X_{rand} is not possible and a new random point is generated. The algorithm iterates until either the point is reached or distance equal to the tree step size is traveled. If the control inputs result in successfully reaching the target point, the corresponding node is added to the RRT and its state configuration is stored. Fig. 5b depicts the proposed approach of node generation using KCRSS.

Algorithm 1 depicts the steps involved in KCRSS, where X_c and X_r are the current state and the state of the robot at the immidiate next node, respectively. Also, α is the angular difference between the current robot orientation and the immediate next node. The key steps in KCRSS are:

- Generate a new random state X_{rand} which satisfies the kinematic constraints based on the current state, X_{near}.
- Find control inputs u to steer the robot for a time δt, from X_{near} to X_{rand}.
- If there are no detected collisions during the move from X_{near} to X_{rand}, add X_{rand} to the tree (T_a or T_b) and record u with new edge.

Algorithm 1 KCRSS (X_c, X_r)

```
1: Input: $u, L, \delta t, \delta s$
2: Output: $x_{new}, y_{new}, \theta_{new}$
3: $length = 0$
4: while $\| (x,y)_c - (x,y)_r \| \leq v * \delta t$ and $length \leq \delta s$ do
5:     $\alpha = \theta_c - atan2(y_r - y_c, x_r - x_c)$
6:     if $abs(\alpha) > \phi$ then
7:         $\alpha = \begin{cases} \phi & if \alpha > 0 \\ -\phi & if \alpha < 0 \end{cases}$
8:     end if
9:     $x_{new} = x_c + v * \delta t * cos(\theta_c + \omega * \delta t)$
10:    $y_{new} = y_c + v * \delta t * sin(\theta_c + \omega * \delta t)$
11:    $\theta_{new} = \theta_c - (v/L) * tan(\alpha) * \delta t$
12:    $CurrentCell = Cell(x_{new}, y_{new})$
13:    if $CurrentCell \in O$ then
14:        return false
15:    else
16:        $length = length + |v| * \delta t$
17:    end if
18: end while
```

Algorithm 2 KB-RRT (X_{init}, X_{goal})

```
1: $T_a$.AddNode($X_{init}$)
2: $T_b$.AddNode($X_{goal}$)
3: while $k <= MAXITERATION$ do
4:     $X_{rand} \leftarrow KCRSS()$
5:     if $\sigma_{new} \in O$ then
6:         continue
7:     end if
8:     $T_a$.Extend($X_{rand}$)
9:     $T_b$.Extend($X_{new}$)
10:    if Connected($T_a, T_b$) $== True$ then
11:        return $Path(T_a, T_b)$
12:    end if
13:    Swap($T_a, T_b$)
14: end while
```

In addition, the RRT generated from X_{goal}, T_b, is also allowed to extended towards this new state. The same process is repeated until the two trees, T_a and T_b are connected and a solution is generated.

Thus T_a and T_b can be defined as,

$$\begin{aligned} T_a &= \{X_{init}, X_a^0, X_a^1, X_a^2, ..., X_a^{n-1}\} \\ T_b &= \{X_{goal}, X_b^0, X_b^1, X_b^2, ..., X_b^{m-1}\} \end{aligned} \tag{3}$$

where m and n are the number of nodes in trees T_a and T_b respectively. Further into the iteration, when the two trees are connected, the path from X_{init} to X_{goal} is returned. The roles of the two trees are reversed at each iteration by swapping them. The tree continues to grow until a path is found or the planner has run for the specified $MAXITERATION$ number. The value of $MAXITERATION$ is user defined based on the envirionment under consideration. Thus KB-RRT constitutes the above described combination of KCRSS and Bi-RRT to generate a feasible path from X_{init} to X_{goal}. KB-RRT does not guarantee a kinematically feasible connectivity between T_a and T_b, however, it assures that they create a connected tree (T_p) where,

$$T_p = \{X_{init}, X_a^0, X_a^1, ..., X_a^{n-1}, X_b^{m-1}, ..., X_b^0, X_{goal}\} \tag{4}$$

The trajectory generation algorithm (described further in Section III-B) calculates a feasible path throughout T_p. Algorithm 2 depicts the algorithmic represenation of the steps involved in KB-RRT.

B. Trajectory Generation

The path generated by KB-RRT T_p (as described in Section III-A), has numerous nodes and does not guarantee an optimal trajectory. Hence, the nodes have to be preprocessed before passing it to the trajectory tracker. The critical nodes ($C = \{c_0, c_1 ....c_n\}$) are identified such that longest possible straight line paths are achieved between the selected nodes without encountering an obstacle. These nodes act as waypoints. Note that, although the way-points do not have obstacles between them, the robot might not be able to take straight line path(depending on the current orientation of the robot). This is due to the constraints imposed by the Ackermann-steering system.

Once the waypoints are generated, a trajectory has to be devised between the waypoints. As the robot is non-holonomic, trajectory generation considers the orientation of the robot and generates trajectories which are kinematically feasible. In this work, a reactive Parametrized Trajectory Generator (PTG$-\alpha$) [2] is considered for generating trajectory between the waypoints using the equation:

$$\begin{aligned} \omega &= \omega_{max}\left(\left(1 + e^{-\frac{\alpha}{k\omega}}\right)^{-1} - \frac{1}{2}\right) \\ v &= v_{max} e^{-(\frac{\alpha}{k_v})^2} \end{aligned} \tag{5}$$

where, ω_{max} and v_{max} are the maximum steering speed and maximum forward velocity respectively.

The parameters k_w and k_v are the parameters which control the slope of the exponential curve as shown in Eq. (5). By changing the values for these variables, the behavior of the trajectory generator can be changed. Large set of trajectories are generated between the waypoints by varying these parameters as shown in Fig. 6.

Trajectories which are not kinematically feasible are dropped due to the reason that the robot is kinematically constrained and cannot make an in-place turn. This is done by using the minimum turning radius (R) constraint given by the equation below:

$$\begin{aligned} R(\alpha) &< R_{min} \\ \text{where,} R(\alpha) &= \frac{v(\alpha)}{\omega(\alpha)} \\ R_{min} &= L * tan\left(\frac{\pi}{2} - \alpha_{max}\right) + \frac{W}{2} \end{aligned} \tag{6}$$

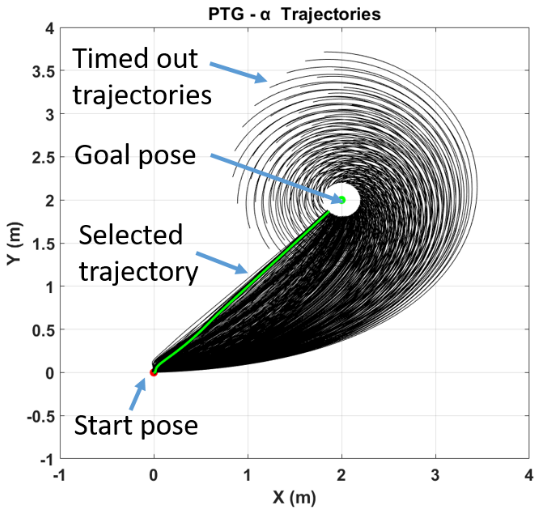


Fig. 6: Trajectories from $(0,0)$ to $(2,2)$ with tolerance of 0.2m

Some of the trajectories are further dropped if they encounter obstacles. The final set of kinematically feasible and obstacle free trajectories are optimized by finding the trajectory with minimum value of $d_i.t_i$ from the set. where, distances $d_i = \{d_0, d_1, .., d_n\}$ and time taken $t_i = \{t_0, t_1, .., t_n\}$. This optimizes the trajectories with respect to distance and time taken.

The parameters k_w and k_v corresponding to the optimal trajectory are implemented using PTG$-\alpha$ to navigate from the current position to the target position as shown in Fig. 6.

IV. Results & Discussions

For testing the performance of the proposed algorithms, computer based simulations were carried out on an Intel Core^{TM} i7 processor with 250GB storage and 8GB RAM. Valgrind tool suite [15] was used for memory analysis. The cluttered environments were simulated by adding adequate number of obstacles. 2D-LiDAR based obstacle map was used for this purpose. Three complicated scenarios were considered to demonstrate the performance of the proposed algorithms. Below is a short description of the scenarios:

- `Scenario 1`: A maze-like structure with only one opening. Source is near the opening and the target is inside the maze at the extreme corner.
- `Scenario 2`: A plane with a tight narrow tunnel space. Source is inside the plane but outside the tunnel. Target is inside the tunnel at the extreme end.
- `Scenario 3`: A complicated maze structure with no-opening. Source and the target both are inside the maze.

In Table I, the details corresponding to the three scenarios considered are captured.

For each scenario, Bi-RRT and KB-RRT were executed to determine the best possible path to reach the target. Trajectory generation was incorporated on KB-RRT generated nodes. Total size of the map considered was $(40m\text{x}40m)$ and the number of obstacle points were varied from 1031 to 2695.

TABLE I: Details of the test scenarios

Sl. No	Start Point (x,y)	Target Point (x,y)	No of Obstacles
`Scenario 1`	$(0,0)$	$(13.50,-7.0)$	1031
`Scenario 2`	$(3.80,18.0)$	$(9.0,15.0)$	2695
`Scenario 3`	$(0,2.0)$	$(27.0,13.0)$	1581

TABLE II: Performance comparison of Bi-RRT and KB-RRT

Sl. No	Algorithm	No. of Nodes	Iterations	Memory (KB)
`Scenario 1`	KB-RRT	3061	16842	1635
	Bi-RRT	6079	57456	3150
`Scenario 2`	KB-RRT	8759	13133	1595
	Bi-RRT	10437	55376	3600
`Scenario 3`	KB-RRT	787	1171	274.3
	Bi-RRT	1707	17196	1521

Both Bi-RRT and KB-RRT were setup with maximum iteration limit ($MAXITERATION$) of 150000 and bias towards the goal was set to 0.7. The results of the Bi-RRT and KB-RRT algorithms were compared based on this setup.

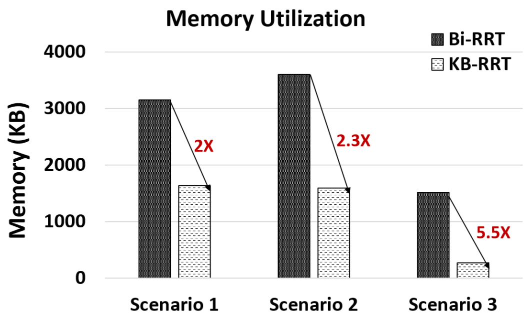
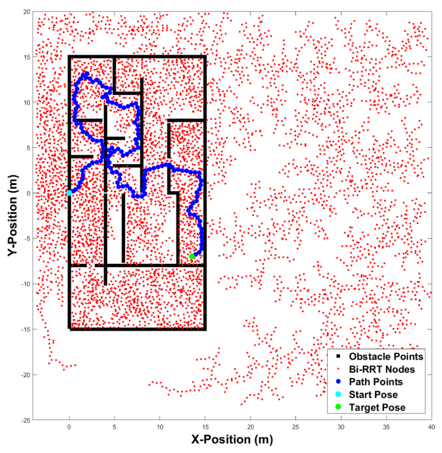


Fig. 8: Comparison of memory utilization

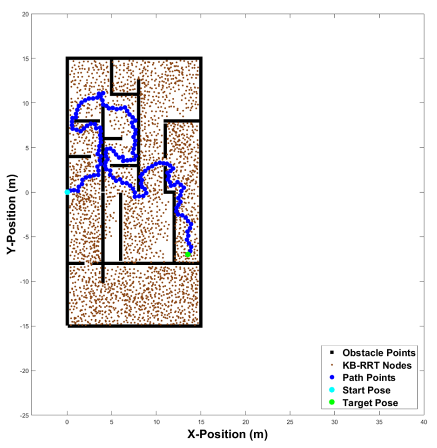
In Fig. 7, the detailed functionality plots for all the scenarios considered are presented. In Figures 7a, 7b, 7c, the results of `Scenario 1` corresponding to Bi-RRT, KB-RRT and trajectory generation are presented, respectively. Note that, the conventional Bi-RRT expansion is uncontrolled and the nodes grow in all possible directions. However, KB-RRT expansion is restricted and the nodes are grown only in the directions which result in kinematically feasible paths, resulting in significant reduction of nodes. Next, smooth, kinematically feasible, optimized (for time and distance) trajectory for the physical motion of mobile robots are generated.

Similarly, the results corresponding to `Scenario 2` and `Scenario 3` are presented in Figures 7d–7f and in Figures 7g–7i, respectively. In `Scenario 2`, as the start point is in open space, the nodes are grown in similar structure between Bi-RRT and KB-RRT. However, the node count is significantly less due to the kinematic constraints imposed on KB-RRT. Even in the tight space such as, `Scenario 3`, the efficient node and trajectory generation using the proposed algorithms are evident from the results.

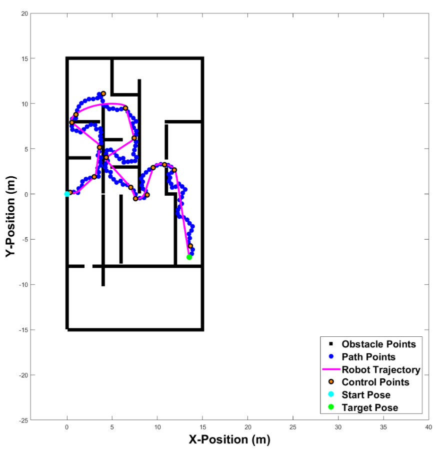
In Table II, the performance summary of Bi-RRT and KB-RRT algorithms operating in the three scenarios considered is presented. Clearly, the number of nodes generated in KB-RRT is significantly less compared to the nodes generated in Bi-RRT resulting in fewer iterations, less compute/execution time, and considerable



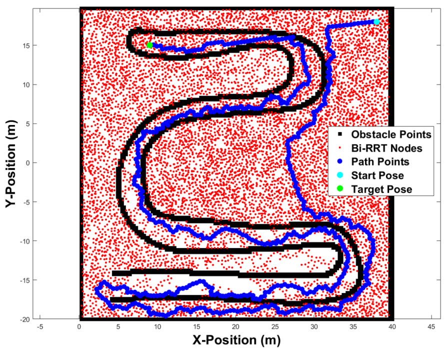
(a) `Scenario 1`: Bi-RRT



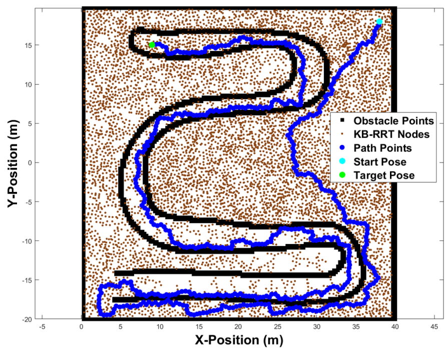
(b) `Scenario 1`: KB-RRT



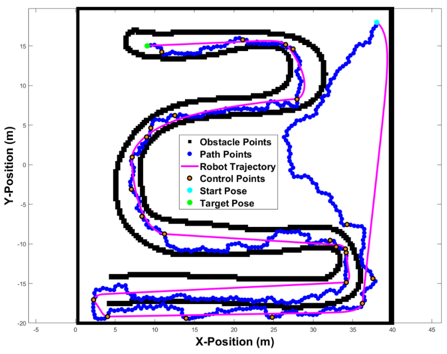
(c) `Scenario 1`: Generated trajectory



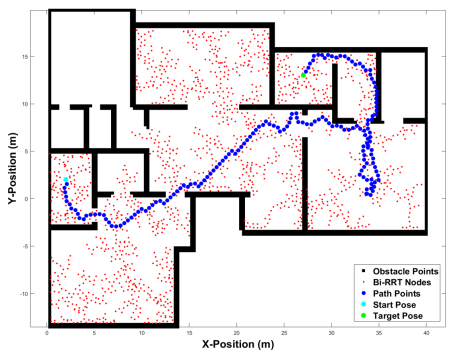
(d) `Scenario 2`: Bi-RRT



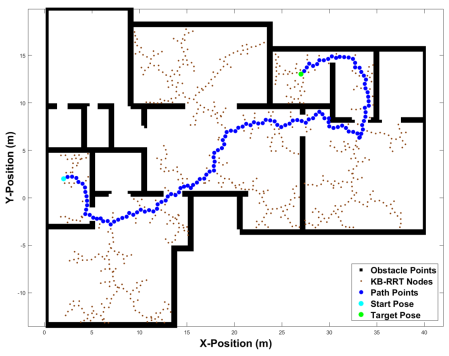
(e) `Scenario 2`: KB-RRT



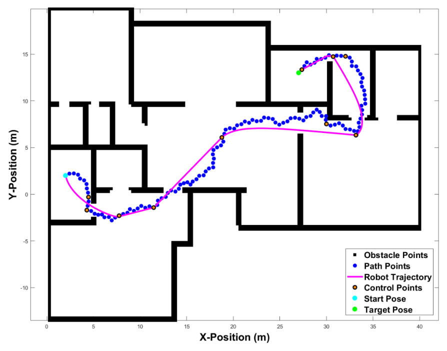
(f) `Scenario 2`: Generated trajectory



(g) `Scenario 3`: Bi-RRT



(h) `Scenario 3`: KB-RRT



(i) `Scenario 3`: Generated trajectory

Fig. 7: Comparison of node points (with Bi-RRT and KB-RRT) and trajectories generated for 3 scenarios considered

reduction in memory utilization. In Fig. 8, the details on memory savings for each scenario are presented. In `Scenario 1`, `Scenario 2`, `Scenario 3`, the memory savings for KB-RRT compared to Bi-RRT are $2X$, $2.3X$, and $5.5X$, respectively. All these results with the proposed methodology demonstrate clear advantage and substantial improvements in key parameters that have direct implications on the hardware including compute, storage and battery life on each mobile robot.

V. Conclusion

In this paper, we present KB-RRT algorithm which improvises over Bi-RRT by incorporating kinematic constraints and trajectory generation - resulting in better performance and smooth navigation in cluttered environments. The algorithm is tested with three different scenarios depicting varied degrees of complexities. The simulations demonstrate reduced number of nodes, reduced iterations/run time, considerable improvement in memory utilization resulting in $3X$ better performance on average for the proposed KB-RRT algorithm compared to the Bi-RRT algorithm operating in identical conditions. Further, in bigger space/dimensions and in all practical situations, the proposed methodology will result in significantly higher savings (in iterations/compute time, memory and overall hardware) thus making it a viable solution for efficient navigation of autonomous mobile robots.

References

[1] Jose Luis Blanco, Mauro Bellone, and Antonio Gimenez-Fernandez. Tp-space rrt–kinematic path planning of non-holonomic any-shape vehicles. *International Journal of Advanced Robotic Systems*, 12(5):55, 2015.

[2] John J Craig. *Introduction to robotics: mechanics and control*, volume 3. Pearson Prentice Hall Upper Saddle River, 2005.

[3] Alberto Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[4] David Hsu, Robert Kindel, Jean-Claude Latombe, and Stephen Rock. Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research*, 21(3):233–255, 2002.

[5] Sertac Karaman and Emilio Frazzoli. Optimal kinodynamic motion planning using incremental sampling-based methods. In *Decision and Control (CDC), 2010 49th IEEE Conference on*, pages 7681–7687. IEEE, 2010.

[6] Sertac Karaman, Matthew R Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. Anytime motion planning using the rrt. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1478–1483. IEEE, 2011.

[7] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE, 2000.

[8] James J Kuffner Jr and Steven M LaValle. *RRT-connect: An efficient approach to single-query path planning*, volume 2. 2000.

[9] Yoshiaki Kuwata, Justin Teo, Gaston Fiore, Sertac Karaman, Emilio Frazzoli, and Jonathan P How. Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on Control Systems Technology*, 17(5):1105–1118, 2009.

[10] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. *The international journal of robotics research*, 20(5):378–400, 2001.

[11] Riccardo Longoni and Paolo Salvatore. Configuration spaces are not homotopy invariant. *Topology*, 44(2):375–380, 2005.

[12] Alejandro Perez, Robert Platt, George Konidaris, Leslie Kaelbling, and Tomas Lozano-Perez. Lqr-rrt*: Optimal sampling-based motion planning with automatically derived extension heuristics. 2012.

[13] Erion Plaku, Kostas E Bekris, Brian Y Chen, Andrew M Ladd, and Lydia E Kavraki. Sampling-based roadmap of trees for parallel motion planning. *IEEE Transactions on Robotics*, 21(4):597–608, 2005.

[14] Basak Sakçak. *Optimal kinodynamic planning for autonomous vehicles*. PhD thesis, Italy, 2018.

[15] Julian Seward, Nicholas Nethercote, and Josef Weidendorfer. *Valgrind 3.3-advanced debugging and profiling for gnu/linux applications*. Network Theory Ltd., 2008.

[16] Dustin J Webb and Jur van den Berg. Kinodynamic rrt*: Asymptotically optimal motion planning for robots with linear dynamics. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5054–5061. IEEE, 2013.