Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2021) 29: 1539 – 1553
© TÜBİTAK
doi:10.3906/elk-2005-208

Obround trees: sparsity enhanced feedback motion planning of differential drive robotic systems

M. Mert ANKARALI
Department of Electrical and Electronics Engineering, Faculty of Engineering, Middle East Technical University, Ankara, Turkey

Received: 30.05.2020 • **Accepted/Published Online:** 19.10.2020 • **Final Version:** 31.05.2021

Abstract: Robot motion planning & control is one of the most critical and prevalent problems in the robotics community. Even though original motion planning algorithms had relied on "open-loop" strategies and policies, researchers and engineers have been focusing on feedback motion planning and control algorithms due to the uncertainties, such as process and sensor noise of autonomous robotic applications. Recently, several studies proposed some robust feedback motion planning strategies based on sparsely connected safe zones. In this class of planning and control policies, local control policy inside a single zone computes and feeds the control actions that can drive the robot to a different connected region while guaranteeing that the robot never exceeds the boundaries of the active area until convergence. While most of these studies apply only to holonomic robotic models, a recent motion planning method (RCT) can solve the motion planning and navigation problems for unicycle like robotic systems based on a randomly connected circular region tree. In this paper, we propose a new/updated feedback motion planning algorithm that substantially enhances the sparsity, computational feasibility, and input effort compared to their methodology. The new algorithm generates a sparse neighborhood tree as a set of connected *obround* zones. Obround regions cover larger areas inside the environment, thus leads to a more sparse tree structure. During navigation, we modify the nonlinear control policy adopted in RCT method to handle the obround shaped zones. The feedback control policy navigates the robot model from one obround zone to the adjacent area in the tree structure, ensuring it stays inside the active region's boundaries and asymptotically reaches the connected obround. We demonstrate the effectiveness and validity of the algorithm on simulation studies. Our Monte Carlo simulations show that our enhancement to the original algorithm probabilistically improves the sparsity, and produces smoother trajectories compared to two motion planning algorithms that rely on sampling based neighborhood structures.

Key words: Path planning, feedback motion planning, robot navigation, sampling based neighbhood graph/tree

1. Introduction

Motion planning & control of robotic platforms is one of the most critical and highest priority tasks in robotic applications. The fundamental goal of the motion planner module in a robotic system is to compute the set of control actions such that the robot (or robots) can safely execute the given motion task without colliding with the static and dynamic obstacles in the environment. The practical duty of the planner can be to drive the robot to a final goal configuration (e.g., autonomous parking [1]), to follow the desired trajectory (e.g., welding robots [2]), to execute a complex mission which is composed of several subtasks [3], etc. Early stages of motion planning algorithms had mainly relied on generating offline open-loop trajectories (or paths) [4–7] as dominant application domain was industrial robotic application. Industrial robotic systems generally operate in highly controlled environments where modeling uncertainties are minimal compared to other robotic applications. Once the offline motion planner generates the open-loop trajectories, high bandwidth industrial controllers and nonback-drivable actuators track these paths in real-time via kinematic control principles.

Recently, the robotic application domain has been rapidly shifting from industrial robotics (controlled environments) to mobile robotic applications, such as autonomous cars, air drones, unmanned underwater vehicles that have to operate in high process and sensor noise conditions. Since original open-loop motion planning techniques can potentially fail even in a mild level of uncertainties, the field moved towards reactive feedback control based motion planning strategies [8–11]. However, due to the nonlinear and nonholonomic robot motion models and nonconvex nature of environments, the majority of feedback motion planning algorithms still relies on generating (online or offline) time-dependent trajectories [8, 12–14]. In these strategies, the feedback control policies compute real-time reactive control actions that can track these trajectories and supply some error-bounds and safety guarantees for the trajectory and controller pairs.

In addition to the existing trajectory-based feedback motion control algorithms, other researchers developed trajectory-free reactive feedback motion planners based on a network of sparsely connected obstacle-free regions [15–19]. In these studies, a discrete motion planner first generates an (online or offline) graph or tree structure that is composed of connected, safe zones inside the environment. After that, local feedback policy inside a single-zone generates the necessary control actions that navigate the robot to a different (connected) area while strictly satisfying that the robot never exceeds the boundaries of the current active region until it reaches the borders of the next zone. Indeed, the significant majority of these connected region-based motion planning methods apply only to holonomic robotic systems [15–18]. Recently Ege & Ankarali [19] developed a motion control strategy for differential drive unmanned surface vehicles based on a network tree of connected circular regions. They showed that their algorithm produces computationally feasible and relatively robust solutions for "simple" circular environments. Later, Ozcan & Ankarali [20] adopted the method to develop a motion planner for a dynamic car model and tested the method on polygonal environments.

In this paper, we propose a new/updated feedback motion planning algorithm, which substantially enhances the sparsity and control smoothness performances compared to their methodologies presented in [19, 20] and an extended version of the SNG algorithm introduced in [16]. Our study's main contribution is that in the discrete planning phase, our algorithm generates a sparse neighborhood tree as a set of connected *obround* zones, which substantially improve the sparsity of the original algorithm. Since the expanded obround regions cover larger areas inside the environment, we can cover the area with fewer nodes, thus potentially solving the motion planning problem relatively more sparse than the methods based on circular regions. Since obround zones enforce a different geometric constraint than the circular areas, we extend the nonlinear control policy proposed in [19] such that the model can safely navigate inside the obround zone and asymptotically converge to the connected obround region or the final goal position.

2. Method

Analogous to the methods presented in [18–20], our approach relies on generating a random tree structure based on connected sparse regions. In the future, we are also planning to expand our work to probabilistic graph-based region networks. Similar to the mentioned studies, our algorithm composed of two phases.

We generate a random tree connected collision-free obround zones using the geometry of the obstacles and boundary in the first phase. The method creates the primary (master) obround region around the goal location similar to other studies. After that, we start the random tree extension process, and at each iteration, we generate a "random" obround connected to one of the existing nodes (i.e., obround zones). Random exploration of the area is inspired by the fundamental RRT algorithm [21]. We repeat the iterations until the last generated node covers the initial position of the robot.

The second phase of the method is the navigation stage. Inside each obround region, the navigation algorithm's task is to drive the robot connected subsequent node inside the tree asymptotically while strictly ensuring that the robot stays inside the boundaries of the active zone.

The previous studies [19, 20] utilize circular connected regions proposed similar control algorithms that can safely (no boundary violation) and asymptotically (in the sense of Lyapunov) drives to the robot to its equilibrium point, located inside another region. In our work, we modify and extend the proposed control policy [19] such that the robot can perform safe and asymptotic navigation inside the obround regions. To achieve this, we propose adding a reference governer type rule to the control policy developed in [19]. Specifically, based on the robot's location inside the obround, we generate "the" largest circle inside the obround that intersects with the robot's current position. We then use the center of this circle as the updated reference position (instead of the global reference) and apply the feedback rule proposed in[19]. This local policy both moves the robot and local reference toward global reference inside the obround shape.

2.1. Random obround tree generation

The random tree generation stage aims to (fully or partially) cover the obstacle-free space with a set of connected obround shape zones (nodes). Similar to previous studies [18, 19], we first generate a zone around the goal location and name it as the root (or master) node. Figure 1 illustrates the node generation processes around a random point, q_g. After that, we start the random tree generation processes by drawing a random sample location, q_{rnd}, inside the boundaries of the environment. If one of the existing nodes cover this random location, we discard the point and resample a different location. Then, we compute the distance, d_i between this sample and each existing node by projecting q_{rnd} to the boundary of the obround zone. Figure 2 illustrates the projection phase and distance computation based on different possible configurations. We then find the index of the closest node to the sample

$$i^* = \arg\min_i d_i. \tag{1}$$

Finally, we generate a new point of $\bar{q}$ by projecting q_{rand} to a smaller interior obround which has a radius of $\bar{r} = \gamma r$ where $\gamma \in (0, 1)$. Figure 2 also visualizes this final process. In this context, we locate $\bar{q}$ inside the region of attraction of $Node_{i^*}$. Finally, we generate a new node around $\bar{q}$ based on the process illustrated in Figure 1 and add it to the node stack, which stores the sparse tree structure. We repeat the whole process until the union of all regions covers the initial condition(s). Algorithm 1 presents the algorithmic steps of our method in detail. Figure 3 provides a comparative illustration of the obround tree generation algorithm and circle based trees adopted in previous studies [19, 20] on a simple environment.

2.2. Motion Model & Control Policy

In this paper, we focus on developing an improved feedback motion planning algorithm for non-holonomic differential drive robotic systems. A significant amount of autonomous ground and water (surface and underwater) vehicles rely on differential drive steering for motion control. Under some assumptions regarding the

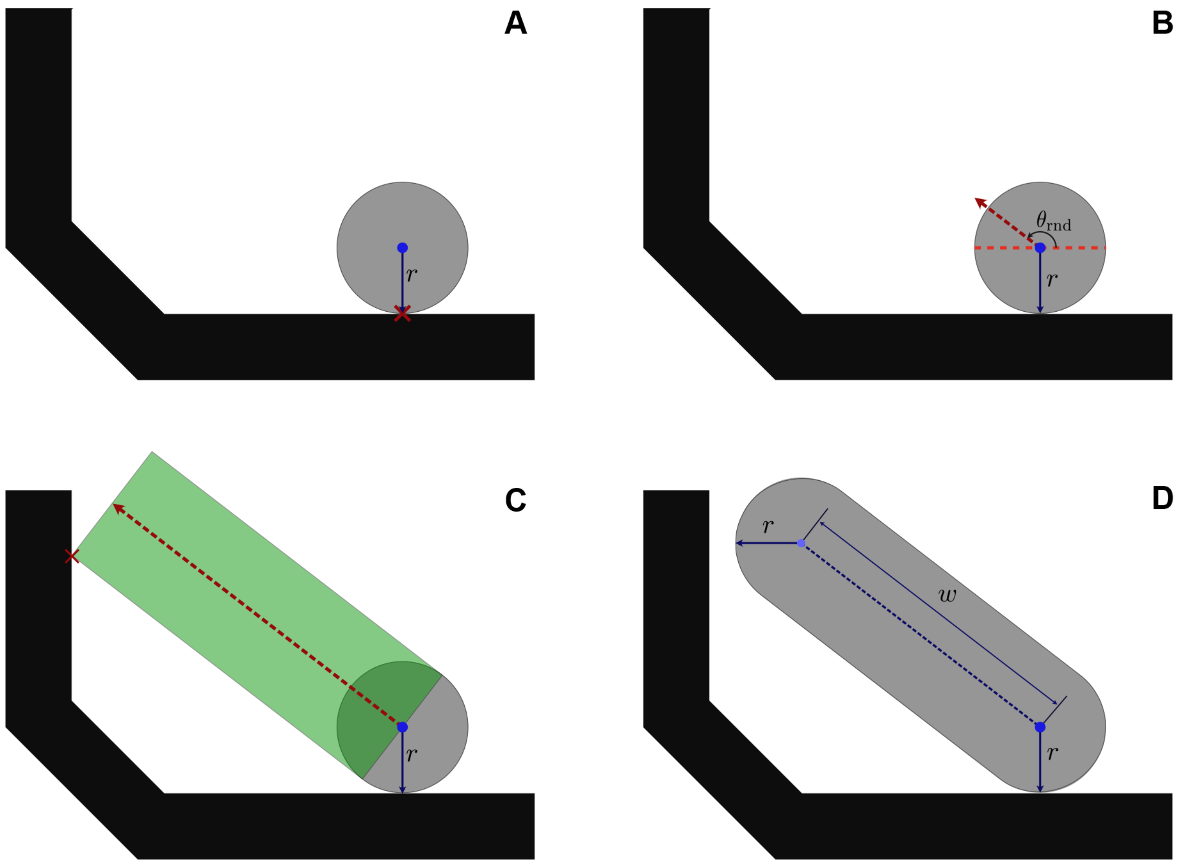

Figure 1. Illustration of the node generation process around a sample point, q_g (blue filled circle) (A) The method first computes the minimum distance of q_g to the obstacles and boundaries and finds associated the closest point q_{min} (red cross) . We then generate a circle with a radius of $r = ||q_g - q_{min}||_2$ which is the largest obstacle-free circle around the pint q_g. (B) We draw a line (orange dashed line) passing through q_c and perpendicular to the normal vector $q_g - q_{min}$. We then generate a random extension vector v_{ext} (red dashed) from q_g such that angle between the vector and the previously generated line is $\theta_{rand} \in (0, \pi)$. (C) We extend a rectangular region (transparent light green), with a height of $2r$, from q_c along the extension vector, v_{ext}, until the extended rectangular hits an obstacle. (D) Finally, we generate an obround shape by "rounding" the corners of the extension rectangle with the original circle's radius r. One can treat this final obround shape as a union of two circles and one rectangle. The first circle is centered around q_g and has a radius of r. The radius of the other circle is also equal to r and center q_{ext} located along the extension direction. Note that the rectangular shape has a height of $2r$ and length of $\omega = ||q_{ext} - q_g||_2$

environmental friction and high-bandwidth velocity controllers, the motion model of these robotics systems can be accurately captured with a unicycle model (illustrated in Figure 4) [19]. The state-space of the unicycle model is composed of cartesian position (x, y), and body orientation, θ measured with respect to a world fixed reference frame, $\mathcal{W}$. The system's driving inputs are forward (and backward) speed, ν, and angular velocity, ω. Since there exists a kinematic constraint on velocity (not position), the system model becomes non-holonomic. In this context, the equations of motion of the system in the state-space form takes the following form.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \nu \cos\theta \\ \nu \sin\theta \\ \omega \end{bmatrix} \tag{2}$$

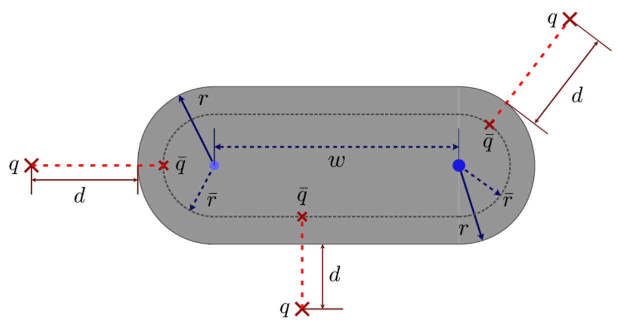

Figure 2. Here, we illustrate the (minimum) distance computation process between a random sample and a selected node (obround zone) and projection operation of "the" sample to the inner obround based on different possible configurations. Blue maker depicts the goal location, q_g of the node, and its the center of the goal circle. The light blue marker is the center point, q_{ext} of the extended circle. r and ω are the radius and length of the obround zone, respectively, whereas $\bar{r} > r$ is the radius of the inner obround.

Algorithm 1 Obround tree generation.

```
1:  $Node_0 \leftarrow$ GenerateObroundRegion($q_{goal}$)
2:  $T \leftarrow$ InitializeTree($GoalNode$)
3:  for $k = 1$ to $K_{max}$ do
4:      $q_{rnd} \leftarrow$ Sample()
5:      $Node_{i^*} \leftarrow$ Nearest($T, q_{rnd}$)
6:      $\bar{q} \leftarrow$ Project($Node_{i^*}, q_{rnd}$)
7:      $Node_k \leftarrow$ GenerateObroundRegion($\bar{q}$)
8:      $T$.InsertNode($Node_k$)
9:      if $q_{init} \in G$ then
10:         return T
11:     end if
12: end for
13: Return T
```

In our study and similar studies based on connected region tree-based motion-planning algorithms, the task of the local control policy for each active node is

- asymptotically driving the model towards the node's goal location, q_g,
- ensuring that the robot's trajectories strictly remains inside the convex boundaries of the zone.

In our paper, the boundary of the obround shape imposes the convex positional constraints that the robot should obey until convergence. The goal location is the center of the central circle. In the previous work based on connected circular regions [19], the authors proposed a control policy that can asymptotically drive the unicycle dynamics towards the center of the circle while also ensuring that the trajectories never leave the circular boundary. They first reformulate the dynamics around the goal location (considering it as the equilibrium point) with respect to the instantaneous body-fixed reference frame, $\mathcal{B}$, using polar coordinates, $[\rho\ ,\ \phi]$. ρ depicts the distance between the unicycle and q_g, and ϕ represents the angle between the robot's heading direction and line connecting the q_g to the body center. Figure 4 illustrates these variables.

Based on this change of variables operation, Ege & Ankarali [19] propose the following nonlinear control

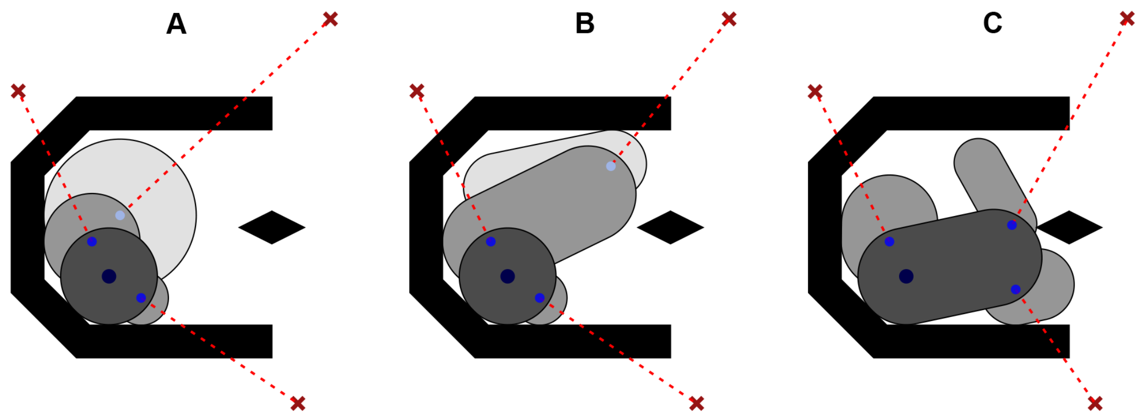

Figure 3. Comparative illustration of the circle trees and obround trees. Here, we compare the three iterations of tree generation based on circular regions [19] (A) and the obround areas (proposed approach) (B & C) where for both tree generation algorithms we use the sample "random" sample locations. Since in the random obround tree generation algorithm, we also sample a random direction of growth, one can obtain different tree structures even with the same random locations. Illustrations (B) and (C) proposes two various tree examples based on the same random points. In all figures, dark grey, grey, and light gray regions illustrate the area of the goal nodes, nodes in the second stage of the tree structure, and nodes of the third stage of the tree structure, respectively. The dark blue marker is the master goal location of the environment. Blue and light blue markers are the "goal" center points of each (non-master) node.

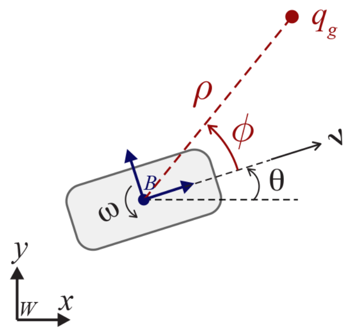

Figure 4. Illustration of the unicycle model and variables. $\mathcal{W}$ and $\mathcal{B}$ denote the world fixed and body-fixed reference frames, respectively. θ is the heading angle of the robot measured with respect to $\mathcal{W}$. $q_g = [x_g \ , \ y_g]^T$ is the goal/target position measured with respect to $\mathcal{W}$. ρ and ϕ denote the polar (radius and angle respectively) coordinates of the q_g with respect to the body frame $\mathcal{B}$.

policy:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} K_v \rho \cos\phi \\ K_\phi \phi \end{bmatrix} \tag{3}$$

and prove using Lyapunov's second method that if one selects the gain parameters, $[K_v \ , \ K_\phi]$, such that $\frac{K_\phi}{K_v} > 1$, trajectories of the model asymptotically reach the equilibrium point, i.e. the goal location and never exceeds the boundries of the circle centered around the goal location.

We adopt the same motion model with [19] in this paper, and if we apply the proposed controller, it can asymptotically drive the robot towards the goal location inside the zone. However, it can not guarantee that trajectories stay inside the convex boundaries of the obround area. In this context, we developed a reference governor algorithm specific to our problem, which strictly (and possibly conservatively) avoids the constraint violation while satisfying the asymptotic convergence.

The reference governor is the name of a class of add-on controllers that modify the reference command of a stable (linear or non-linear) feedback control system such that it can impose some constraints on state variables [22, 23]. In a control system topology enhanced with a reference governer, the first step is to design a stable controller ignoring (all or some) constraints. In this context, we will adopt the control policy in (3), since the algorithm can asymptotically drive the robot to a goal point while also ensuring that the trajectories never exceeds the boundaries of the circle centered around a goal location, i.e. "global" equilibrium of the active node.

Note that inside an active node (obround region), we treat the center of the master circle (illustrated with a dark blue marker in Figure 5) as the primary target/reference location, q_g. If the robot's cartesian position locates inside the master circle (i.e., the circle centered around q_g and has a radius of r), we directly apply the control policy in (3). Note that this control policy can both asymptotically stabilize the motion around q_g while also satisfying that trajectories strictly remains inside the master circle, which is entirely located inside the obround zone, ensuring that no constraint violation occurs inside the obround zone. Figure 5 illustrates the primary reference location and the master circle, together with the essential variables used by the control policy.

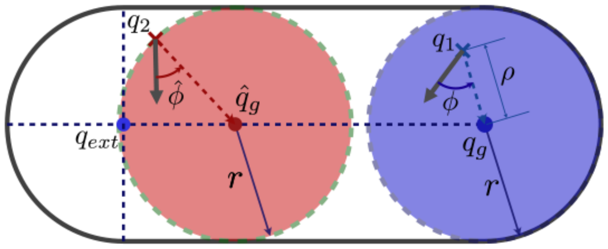

Figure 5. Illustration of the control policy based on two possible configurations. q_1 and q_2 visualize the two categorically different possible robot locations. Grey arrow illustrates the direction of the robot in both cases. Master circle (blue transparent region) covers q_1, i.e., $||q_1 - q_g||_2 = \rho \leq r$. In this scenario, we adopt the control policy in (3) directly where the reference location is the center, q_g. q_2 is outside of the master circle boundary (light blue dashed line), i.e. $||q_2 - q_g||_2 > r$. In this scenario, the method first generates the circle (inside the obround) such that its radius is equal to r, its center, $\hat{q}_g$ resides on the axis connecting q_g, q_{ext}, its boundary crosses q_2, and $(q_g - q)^T(\hat{q}_g - q) \geq 0$. Then the governor selects $\hat{q}_g$ as the instantaneous reference location and updates the control policy. Note that in this scenario $\hat{\rho} = r$ for all possible locations. Algorithm computes $\hat{\phi}$ based on the robot's orientation, q_2 and $\hat{q}_g$.

However, if the robot's cartesian position locates outside the boundaries of the master circle, i.e. when $||q - q_g||_2 > r$, we can not apply the same local control policy since trajectories may deviate from the obround zone. In this case, the control algorithm generate an instantaneous modified reference location, $\hat{q}_g$, which is located on the axis connecting q_g and q_{ext} of the active node.

Let q be the robots current location and $||q - q_g||_2 > r$. We find the circle with following features

- Center, q_g, is on the axis connecting q_g and q_{ext},
- Radius is equal to r (same with the master circle),
- Robot's location q is on its boundary, i.e. $||q - \hat{q}_g||_2 = r$
- $(q_g - q)^T(\hat{q}_g - q) \geq 0$

Then, we choose $\hat{q}_g$ as the instantaneous goal location and feed it to the controller. The controller computes the modified polar coordinate variables, i.e. $[\hat{\rho}\ ,\ \hat{\phi}]$ and then generates the control actions. Figure 5 illustrates the generated circle, modified reference location, and updated polar coordinate variables for an example robot location q_2 that is outside the master circle boundary. The equations below detail the mathematical procedure of the derivation of $\hat{q}_g$:

$$\begin{aligned}
e &= \frac{q_{ext} - q_g}{||q_{ext} - q_g||_2}\ , \\
\kappa &= (q - q_g)^T e\ , \\
\alpha &= \kappa - \sqrt{r^2 - (||q - q_g||_2^2 - \kappa^2)}\ , \\
\hat{q}_g &= q_g + \alpha \cdot v\ .
\end{aligned} \tag{4}$$

A well defined reference governor [22] has to be constructed such that

- if $\hat{q}_g$ is kept constant, no constraint violation occurs in the subsequent motion,
- $\hat{q}_g$ is as close to q_g as possible

Note that if $\hat{q}_g$ kept constant by the construction of the control policy, we guarantee that the robot will remain inside the circle (red region in Figure 5) for which the center and radius are equal to $\hat{q}_g$ and r, respectively. Since the obround shape of the node entirely covers this circular region, the algorithm satisfies the first condition (most probably conservatively). If $\hat{q}_g$ is between q_g and the projection of q on the axis connecting q_g to q_ext, $\hat{q}_g$ continuously approaches to q_g that automatically satisfy the second condition. Note that in the formulation (4), $\kappa > \alpha$ for all possible configurations and this guarantees the condition on the location of $\hat{q}_g$.

Fig. 6 illustrates the block diagram topology of the whole closed system.

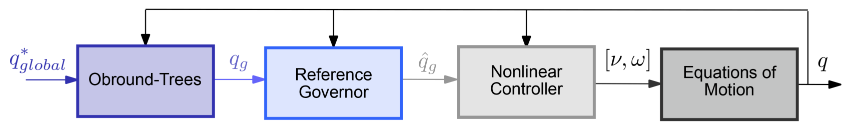

Figure 6. Illustration of the block diagram topology of the whole closed-loop system. "Obround-trees" block generates the connected region tree stucture based on the global goal location q^*_{global} and feeds the active node's parameters to the "Reference governer" block. "Reference governer" generates a modified $\hat{q}_g$ command considering the geometric constraints of the abround shape and feeds this command to the controller. "Nonlinear controller" block generates the input actions and feed them to the plant dynamics. Equations of motion dynamics are executed based on the unicycle model parameters.

2.3. Directed sampling based neighborhood graph (D-SNG)

To compare the effectiveness of our algorithm, we extended and implemented a sampling-based neighborhood graph (SNG) algorithm [16], one of the first and most cited sparse feedback motion strategies in the literature. Similar to our approach, the core idea in [16] is to cover the environment's obstacle-free spaces with intersecting random zones and create some feedback control policies that can navigate the robot models in between these overlapping regions. Unlike the directed tree-based approach adopted in this paper, Yang & Lavalle generates a *undirected* graph structure by analyzing the intersections between circular neighborhoods.

However, it is not possible to directly implement the methods in [16] and compare the results with our algorithm, since Yang & Lavalle [16] only considers robot models with strictly holonomic constraints. In this context, we modified their methodology to suit for navigating (nonholonomic) differential drive systems. Because the SNG algorithm considers holonomic only robot models, when there exists an intersection between two circular regions, SNG algorithm can generate a bidirectional edge between two vertices (regions). After that, the SNG creates two different navigation functions (holonomic feedback control policies); each can steer the robot from one circular zone to the next. As we discussed previously, RCT [19] method also relies on a connectivity structure (particularly tree) based on randomly generated overlapping circles; however, by construction, they only generate a connection where center of one circle is located inside the other region. The feedback control policy introduced in [19] can asymptotically guide the (nonholonomic) robot to a connected region since its equilibrium point locates inside the connected region. In this context, our first extension to the SNG algorithm is that we create a *directed* graph, $G = \{V, E\}$, by analyzing the types of connectivity structures. Suppose two circles associated with two vertices, (v_i, v_j) intersect each other, and without loss of generality, assume that $r_i \geq r_j$, where r stands for the radius of the associated regions. Based on the distance between centers, $d_{i,j} = ||q_i - q_j||_2$, there are three possible outcomes in terms of connectivity structures.

- If $d_{i,j} > r_i$, then there exist no edge connecting the vertice pair (v_i, v_j)
- If $r_i \geq d_{i,j} > r_j$, then there exist a unidirectional connection and one edge, $e_{j \mapsto i}$, connecting the vertice pair (v_i, v_j)
- If $r_j \geq d_{i,j}$, then there exist a bidirectional connection and two edges, $e_{j \mapsto i}$ & $e_{i \mapsto j}$, connecting the vertice pair (v_i, v_j)

Fig. 7 illustrates these connectivity types and associated details. We also adopted a modified sampling strategy to handle the new connectivity structure and nonholonomic navigation. At each iteration, the SNG algorithm draws a random sample, q_{rnd}, from the configuration space. If this random location is located inside the collision-free space, $q_{rnd} \in \mathcal{C}_{free}$ and unexplored regions, $q_{rnd} \notin \mathcal{B}$, SNG treat this sample as *success* and generate a vertex (circular region) v_{new} around this point. This strategy ensures that no neighborhood is a subset of another neighborhood and enhances sparsity. However, due to the constraints associated with nonholonomic navigation, if we adopt such a sampling strategy, it would not be possible to establish unidirectional connections between vertices. Moreover, since the new zone's circle would be outside of the existing regions, it would not be possible to add an edge that directs towards the new edge. This means that it won't be possible to connect existing vertices with this sampling strategy, and we can never find a solution (graph) that connects start location to the goal zone. In order solve this problem, we adopt a slightly different sampling strategy. At each iteration, D-SNG algorithm draws a random sample q_{rnd} from the configuration space:

- If $q_{rnd} \notin \mathcal{C}_{free}$ then the sample is treated as "failure",
- else if $q_{rnd} \notin \mathcal{B}$ then the sample is treated as "success" ,
- else if $q_{rnd} \in \mathcal{B}$ then instead of considering the sample as "failure", we draw a random binary variable from the Bernoulli distribution with a probability of P, i.e. $b \sim \text{Ber}(P)$, then based on the value of b,

D-SNG algorithm takes the following actions

$$b = \begin{cases} 0 & \rightarrow \text{ failure} \\ 1 & \rightarrow \text{ success} \end{cases}$$

In other words, when a sample is covered by at least of one the regions, we discard the sample with a probability of $1 - P$ and we keep the sample and generate a vertex around the point with a probability of P. If we select $P = 0$, then sampling strategy will be same with the original SNG method. This approach allows generating zones whose centers are located inside existing nodes and eliminates the problems discussed above with the possible cost of reducing the sparsity performance. D-SNG re-iterates the sampling process until the start location covered by a node (region) which is "connected" to the graph group that contains the goal zone. However, unlike tree-based methods, we need to run a search algorithm to find a path (sequence of zones) that connects the node that covers the start location to the goal zone. D-SNG adopts the A^* search algorithm to find the path due to its completeness, optimality, and optimal efficiency features [24]. Once A^* finds a discrete path from start vertex to goal vertex, we terminate the search and discrad the unopened zones to enhance the sparsity. One should also note the fact that, A^* finds the optimal path to the goal location for each of the explored nodes.

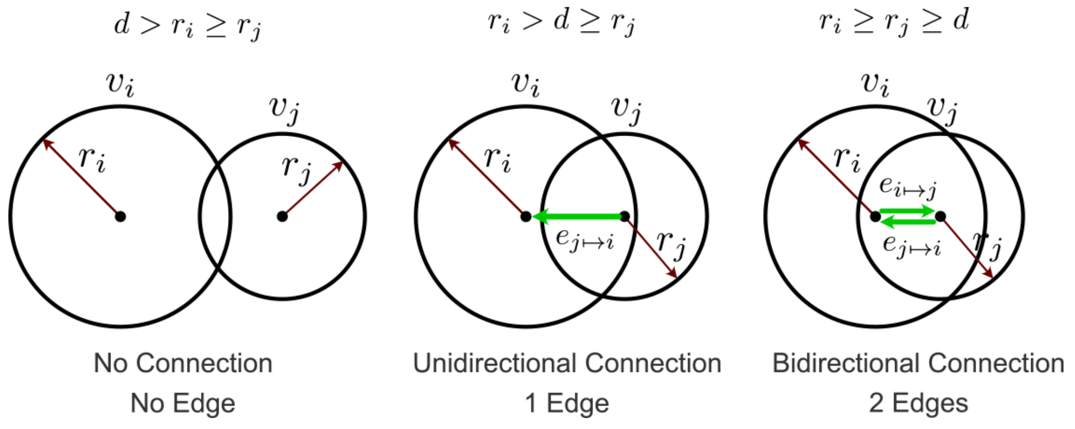

Figure 7. Illustration of the connectivity structures between two *intersecting* vertices in D-SNG algorithm

3. Results

We performed systematic comparative simulation experiments to demonstrate the effectiveness and performance of our new feedback motion planning methodology. For each experimental simulation scenario, we both implemented our algorithm, the motion planning method in [19], as well as D-SNG algorithm using MATLAB to illustrate the relative performance of the new enhanced approach. We will refer the motion planning algorithm proposed in [19], as RCT (a.ka. random-circle-trees), to present the comparison results clearly. In our simulation experiment, we compared both methods based on sparsity performance, measured in terms of $\#Nodes$, control smoothness, measured in terms of root mean square value of forward acceleration, a_{rms}, and total trajectory length.

Figure 8 shows the arena that we used to test the effectiveness of both algorithms. The arena's boundary is rectangular, with a width of $16m$ and a height of $8m$. The map consists of six polygonal obstacles and compromises important benchmark features such as shot passages, dead ends, and local minima for a fair and effective comparative analysis. In all simulations, we adopted the same control and algorithmic parameters for both algorithms, $\gamma = 0.9$, $K_\nu = 4$, and $K_{phi} = 2$. In D-SNG algorithm we also selected $P = 0.1$ (The algorithm produced similar performances, when $P \in (0, 0.5)$).

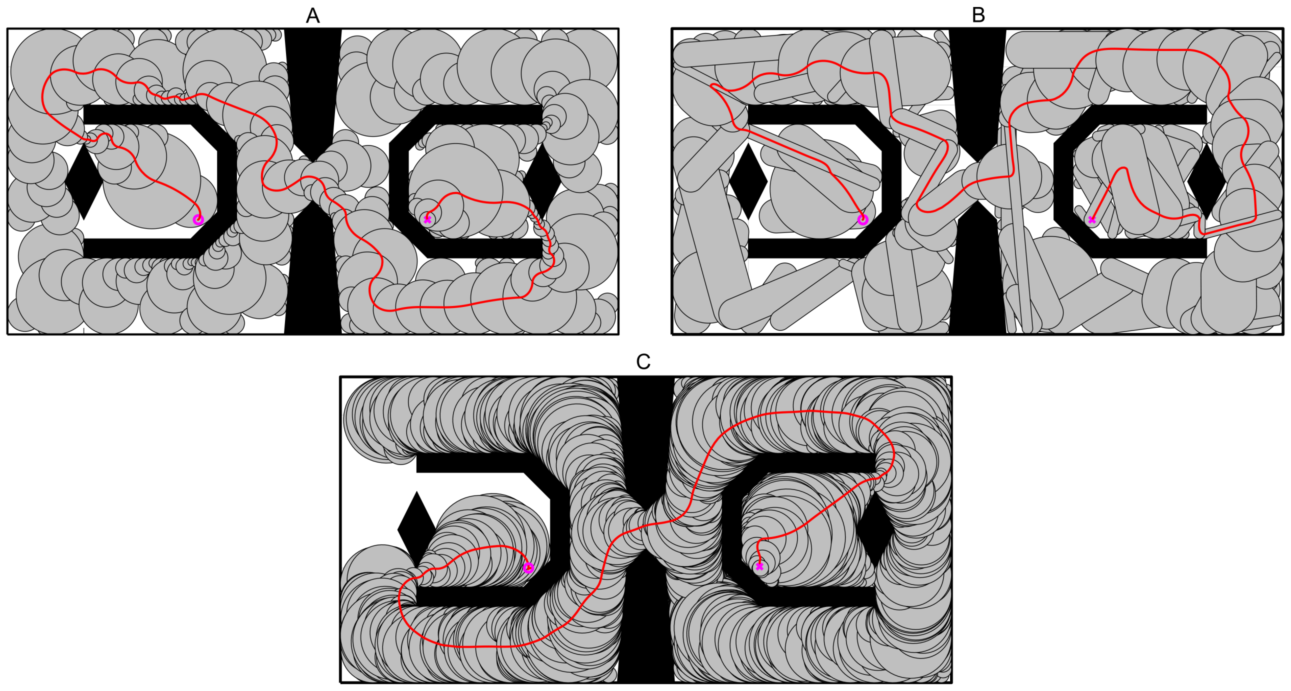

Figure 8. This figure illustrates an example simulation result where we tested all three algorithms: obround-trees, RCT, and D-SNG. The maps boundary is rectangular (black box), with a width of $16m$ and a height of $8m$, and it consists of six polygonal obstacles (black). For all methods, we adopted the same parameters: $\gamma = 0.9$, $K_\nu = 4$, $K_{phi} = 2$, $q_{goal} = [11, 3]m$ (magenta cross marker), and $q_{initial} = [5, 3]m$ (magenta circle marker). Column (A), (B), & (C) belong to the simulations performed with RCT, obround-trees, & D-SNG, respectively. Red curves illustrate the resultant robot trajectory in both scenarios. Light gray regions (circle in A & C, and obround in B) with dark grey boundaries illustrate the zones of each node in each algorithm. Each active zone is responsible from navigating the initial conditions that is visible in the illustration. In other words the sequence of zones acts like a ladder which illustrates the priority queue.

Figure 8 illustrates a sample result from our simulations in Figure 8. In this example, initial and goal positions are equal to $q_{initial} = [5, 3]m$, and $q_{goal} = [11, 3]$ respectively. In these samples, D-SNG and RCT algorithms found solutions with 247 and 1769 # nodes, respectively, where as obround-trees reached the solution with only 122 # nodes.We can see that for these specific examples, obround-trees is the best algorithm in terms of sparsity. Even more surprisingly, D-SNG's sparsity metric is the order of magnitude higher (worse) than the other two algorithms. We speculate that the relaxation of the holonomic assumption in the original SNG algorithm [16] categorically changes the sparsity feature. It may not even be possible to refer D-SNG as a sparse motion planning algorithm.

On the other hand, control-smoothness for RCT, D-SNG, and obround-trees algorithms are 3.4, 3.7 & $2.9m/s^2$, respectively. These sample results show that our method has the potential to increase control-smoothness performances compared to the other two algorithms. Finally, if we compute the resultant trajectory lengths for RCT, D-SNG, and obround-trees, we find the following values, respectively; 30.2, 26.7 & $34.6m$. D-SNG method provides the trajectory with minimum length. Since D-SNG covers the work-space with a relatively dense graph structure and then finds the optimal discrete path using A^* search algorithm, we expected to see a similar result.

To systematically compare the performances of both algorithms, we performed 500 Monte Carlo simulations using the same environment, and the same initial & goal configuration set shown in Figure 8 and compared all methods based on the three performance metrics.

Table 1 presents the results of the Monte Carlo simulations. Based on these results RCT and D-SNG solve the motion problem with 292 and 3340 number of nodes on average. On the other hand, the mean number of nodes produced with the obround-trees algorithm is equal to 165, which corresponds to a more than 40% reduction compared to RCT and order of magnitude smaller than the D-SNG method. In terms of control smoothness metric, RCT and D-SNG produce trajectories for which mean RMS values of acceleration data are $\mu_{a_{\text{rms}}} = 3.4m/s^2$ and $\mu_{a_{\text{rms}}} = 3.8m/s^2$ respectively. The Obround-Trees algorithm reduces this cost metrics to $\mu_{a_{\text{rms}}} = 1.9m/s^2$. These results imply that not only the Obround-Trees method is significantly better in terms of sparsity performance, but also it produces significantly smoother trajectories that can be critical for physical robotic platforms. Finally, when we observe the trajectory length perfomances, quite expectedly D-SNG provides the shortest path on average thanks to its graph structure and optimal discrete serach stage. Moreover, it seems that average trajectory length (30.3m) in RCT approach is also shorter than the one produced by our algorithm (34.6m). It seems that there is a trade-off in improving sparsity and control smoothness metrics. In the future, we will concentrate on the optimality aspect and improve obround-trees algorithm in that regard.

Table 1. Monte-Carlo simulation results. μ and σ denote mean and standard deviation of the corresponding metric, respectively.

Algorithm	# Nodes	a_{rms} [m/s^2]	Path Length $[m]$
Obround-Trees	$\mu = 165$ $\sigma = 36$	$\mu = 1.9$ $\sigma = 0.4$	$\mu = 34.6$ $\sigma = 7.9$
RCT	$\mu = 292$ $\sigma = 35$	$\mu = 3.4$ $\sigma = 0.3$	$\mu = 30.3$ $\sigma = 1.8$
D-SNG	$\mu = 3340$ $\sigma = 1164$	$\mu = 3.8$ $\sigma = 0.4$	$\mu = 26.1$ $\sigma = 0.6$

In addition to these simulations, we also tested the robustness of the obround-trees feedback motion planning algorithm under relatively extreme actuator/process noise conditions. In these tests, we contaminated the actuator inputs, (v, ω), with zero mean white Gaussian noise. We performed 300 Monte-Carlo simulations with different SNR (signal-to-noise-ratio) levels, $\text{SNR} \in \{0.25, 0.5, 0.75\}$. SNR level of 0.25 corresponds to an average of %25 RMS error between the computed and commanded actuator inputs, which is quite radical even for physical robotic platforms. In simulations with $\text{SNR} \in (0.25, 0.5)$, the obround-trees algorithm successfully and safely navigated the robot to the goal position in all of the Monte-Carlo tests. On the other hand, for $SNR = 0.75$, the model converged the goal position in %85 of the simulations, which quite remarkable considering the extreme level of process noise. These results show that the obround-trees method is robust to actuator/process noise conditions, which is a promising result for future experimental applications.

4. Conclusion

Path & motion planning has always been one of the most important and popular topics in the robotics community. Robotics researchers developed numerous algorithms & techniques for different variants of motion planning & control problems [25]. In this paper, we concentrated on one of the most fundamental and practically important motion planning tasks, i.e. computing a set of control policies and actions for a differential drive mobile robotic system such that the robot can be safely driven to a specified goal location from any initial condition (covered by the algorithm) inside the environment (2D) without colliding with the obstacles. Recently, several researchers focused on developing robust feedback motion planning strategies for robotic systems based on generating a sparse random network of safe zones [15–19] inside the environment. These approaches first sparsely partition the obstacle-free areas of the environment with connected random regions and generate a neighborhood graph or tree based on the connection structures between these regions. After that, a discrete planner creates discrete paths inside the neighborhood network from all of the nodes (zones) to the goal node, i.e., the region that hosts the goal location. Finally, local feedback control policy inside a single-zone generates the necessary control actions that navigate the robot to a different (connected) area while strictly satisfying that the robot never exceeds the boundaries of the current active region until it reaches the borders of the next zone.

Recently Ege & Ankarali [19] developed a motion control strategy for the same problem that we focus on based on a network tree of connected circular regions, which we refer to as RCT in this paper. RCT method starts with generating random obstacle-free circular areas and connect then to form a tree structure to obtain a sparse network tree of nodes. Each circle region acts as the basins of attraction of some local feedback control policies [19]. RCT then connects these feedback policies in the essence of the sequential composition idea introduced by Burridge et al. [26]. Each region is associated with a feedback control policy. It is responsible for navigating the robot inside the area/volume to a final location located inside a different but sequentially connected region.

In this paper, we proposed an improved feedback motion planning algorithm, which substantially enhances the sparsity, control effort, and control smoothness performances of the RCT algorithm. During, offline planning phase, our method partitions the obstacle-free region with sparsely connected *obround* zones (instead of circles) and form a tree network structure. Due to the construction methodology of the obround zones, nodes based on obround shapes cover larger areas than the circle. Hence, it dramatically reduces the number of nodes to reach a solution. The Monte-Carlo simulation results clearly show that obround-trees algorithm sparsity performance is substantially better than the RCT and D-SNG algorithms.

In both algorithms (RCT, D-SNG, and Obround-Trees), inside an active node, we define a local feedback control policy. The task is to navigate the robot to a different connected zone while also strictly satisfying that robot will stay inside the boundaries of the active region. RCT algorithm adopts a nonlinear control policy that guarantees that trajectories converge to the circle's center asymptotically and never violates the circle boundary. We adopt the same feedback control policy for D-SNG since both approachs rely on circular zones. However, since obround regions enforce different convex geometric constraints than the circular areas, we can not directly use the control policy of the RCT approach [19]. To solve this problem, we developed a custom add-on reference governor block, that modifies the target location (i.e. reference position) to guarantee that trajectories never violate the geometric constraints imposed by the obround shape. This reference governed enhanced local control policy not only ensured that the robot can safely navigate and reach master goal location, but also it substantially improved the control effort and smoothness performances.

In the future, we would like to extend our algorithm in this paper for higher-order dynamical robot models, i.e., systems driven by forces and/or accelerations, and possibly test the methodology on a physical robotic platform. In addition to these, we also want to adopt the method for robots working in 3D environments, such as drones and unmanned underwater vehicles, by utilizing capsular volumes (3D extension of obround).

Acknowledgments

This work was supported by the Scientific and Technological Research Council of Turkey (TUBITAK) through project number 118E195.

References

[1] Li THS, Yeh YC, Wu JD, Hsiao MY, Chen CY. Multifunctional intelligent autonomous parking controllers for carlike mobile robots. IEEE Transactions on Industrial Electronics 2009; 57 (5): 1687–1700.

[2] Ogbemhe J, Mpofu K. Towards achieving a fully intelligent robotic arc welding: a review. Industrial Robot: An International Journal 2015; 42 (5): 475–484.

[3] Li B, Moridian B, Kamal A, Patankar S, Mahmoudian N. Multi-robot mission planning with static energy replenishment. Journal of Intelligent & Robotic Systems 2019; 95 (2): 745–759.

[4] Kavraki LE, Svestka P, Latombe JC, Overmars MH. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation 1996; 12 (4): 566–580.

[5] Piazzi A, Visioli A. Global minimum-jerk trajectory planning of robot manipulators. IEEE Transactions on Industrial Electronics 2000; 47 (1): 140–149.

[6] Mitsi S, Bouzakis KD, Mansour G, Sagris D, Maliaris G. Off-line programming of an industrial robot for manufacturing. The International Journal of Advanced Manufacturing Technology 2005; 26 (3): 262–267.

[7] LaValle SM, Kuffner Jr JJ. Randomized kinodynamic planning. The International Journal of rRobotics Research 2001; 20 (5): 378–400.

[8] Tedrake R, Manchester IR, Tobenkin M, Roberts JW. LQR-trees: Feedback motion planning via sums-of-squares verification. The International Journal of Robotics Research 2010; 29 (8): 1038–1052.

[9] Karaman S, Frazzoli E. Sampling-based algorithms for optimal motion planning. The International Journal of Robotics Research 2011; 30 (7): 846–894.

[10] La Valle SM. Motion planning. IEEE Robotics & Automation Magazine 2011; 18 (2): 108–118.

[11] Agha-Mohammadi AA, Chakravorty S, Amato NM. FIRM: Sampling-based feedback motion-planning under motion uncertainty and imperfect measurements. The International Journal of Robotics Research 2014; 33 (2): 268–304.

[12] Caldwell CV, Dunlap DD, Collins EG. Motion planning for an autonomous underwater vehicle via sampling based model predictive control. In: OCEANS 2010 MTS/IEEE SEATTLE; Seattle, WA, USA; 2010. pp. 1–6.

[13] Moore J, Cory R, Tedrake R. Robust post-stall perching with a simple fixed-wing glider using LQR-Trees. Bioinspiration & Biomimetics 2014; 9 (2): 025013.

[14] Reist P, Preiswerk P, Tedrake R. Feedback-motion-planning with simulation-based LQR-trees. The International Journal of Robotics Research 2016; 35 (11): 1393–1416.

[15] Yang L, LaValle SM. A framework for planning feedback motion strategies based on a random neighborhood graph. In: Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation; San Francisco, CA, USA; 2000. pp. 544–549.

[16] Yang L, Lavalle SM. The sampling-based neighborhood graph: An approach to computing and executing feedback motion strategies. IEEE Transactions on Robotics and Automation 2004; 20 (3): 419–432.

[17] Lindemann SR, LaValle SM. Simple and efficient algorithms for computing smooth, collision-free feedback laws over given cell decompositions. The International Journal of Robotics Research 2009; 28 (5): 600–621.

[18] Golbol F, Ankarali MM, Saranli A. RG-Trees: Trajectory-Free Feedback Motion Planning Using Sparse Random Reference Governor Trees. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS); Madrid, Spain; 2018. pp. 6506–6511.

[19] Ege E, Ankarali MM. Feedback motion planning of unmanned surface vehicles via random sequential composition. Transactions of the Institute of Measurement and Control 2019; 41 (12): 3321–3330.

[20] Ozcan M, Ankarali MM. Feedback Motion Planning For a Dynamic Car Model via Random Sequential Composition. In: 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC); Bari, Italy; 2019. pp. 4239–4244.

[21] LaValle SM, Kuffner JJ, Donald B, et al. Rapidly-exploring random trees: progress and prospects. Algorithmic and Computational Robotics: New Directions 2001; (5): 293–308.

[22] Kolmanovsky I, Garone E, Di Cairano S. Reference and command governors: A tutorial on their theory and automotive applications. In: 2014 American Control Conference; Portland, OR, USA; 2014. pp. 226–241.

[23] Garone E, Di Cairano S, Kolmanovsky I. Reference and command governors for systems with constraints: A survey on theory and applications. Automatica 2017; 75 : 306–328.

[24] Zeng W, Church RL. Finding shortest paths on real road networks: the case for A. International Journal of Geographical Information Science 2009; 23 (4): 531–543.

[25] LaValle SM. Planning Algorithms. Cambridge, UK: Cambridge University Press, 2006.

[26] Burridge RR, Rizzi AA, Koditschek DE. Sequential composition of dynamically dexterous robot behaviors. The International Journal of Robotics Research 1999; 18 (6): 534–555.