RG-Trees: Trajectory-Free Feedback Motion Planning Using Sparse Random Reference Governor Trees

Ferhat Golbol, M. Mert Ankarali and Afsar Saranli

***Abstract*— Sampling based methods resulted in feasible and effective motion planning algorithms for high dimensional configuration spaces and complex environments. A vast majority of such algorithms as well as their application rely on generating a set of open-loop trajectories first, which are then tracked by feedback control policies. However, controlling a dynamic robot to follow the planned path, while respecting the spatial constraints originating from the obstacles is still a challenging problem. There are some studies which combine statistical sampling techniques and feedback control methods which address this challenge using different approaches. From the feedback control theory perspective, Reference Governors proved to be a useful framework for constraint enforcement. Very recently, Arslan and Koditschek (2017) introduced a feedback motion planner that utilizes Reference Governors that provably solves the motion planning problem in simplified spherical worlds. In this context, here we propose a "trajectory-free" novel feedback motion planning algorithm which combines the two ideas: random trees and reference governors. Random tree part of the algorithm generates a collision-free region as a set of connected simple polygonal regions. Then, reference governor part navigates the dynamic robot from one region to the adjacent region in the tree structure, ensuring it stays inside the current region and asymptotically reaches to the connected region. Eventually, our algorithm robustly routes the robot from the start location to the goal location without collision. We demonstrate the validity and feasibility of the algorithm on simulation studies.**

I. Introduction and Related Work

Obstacle avoidance motion planning is one of the most fundamental tasks of mobile robotics. The problem can be stated as follows: given the robot dynamics and the description of the environment, the motion planner finds a sequence of commands such that the robot reaches the goal configuration(s), while respecting the constraints such as avoiding collisions with the obstacles, remaining within velocity, acceleration and motor voltage limits. Traditionally, an off-line planner determines a collision-free trajectory that reaches a goal configuration for the kinematic model of the robot, and an on-line feedback controller follows that trajectory as closely as possible [1].

Several attempts have been made to use the reference governor framework for motion planning under constraints. Arslan and Koditschek [2] propose a provably correct, computationally efficient motion planner for a world of spherical obstacles. Petersen et al. [3] use rotating virtual hyperplane concept, introduced by Park et al. [4], which puts an additional, time varying constraint to avoid collision. Finally, without emphasizing motion planning, Gilbert and Kolmanovsky [5] suggest hybrid reference governors: the idea of dividing the configuration space into convex, collision-free subsets, which are managed using reference governors. Technically the basic idea of our method, random reference governor trees for feedback motion planning, is similar to the principles presented in this study.

The authors are with the Department of Electrical and Electronics Engineering, Middle East Technical University, 06800 Ankara, Turkey {ferhatg, mertan, afsars}@metu.edu.tr

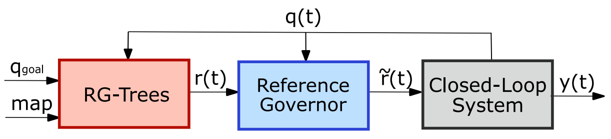

Fig. 1. Block diagram of the algorithm. RG-Trees algorithm determines medium term set points, $r(t)$, which are reachable from the current configuration $q(t)$ and steer to the goal location q_{goal}. Reference governor modifies $r(t)$ such that the closed-loop robot does not collide with the obstacles.

In this paper, we propose a new trajectory-free, sampling based, computationally efficient motion planning algorithm that can handle arbitrary obstacle configurations provided that the system dynamics are linear or feedback linearizable. The algorithm is composed of two stages: a random tree generation stage and a feedback motion control stage. The block diagram topology of our method is illustrated in. Fig. 1. The motion planning stage generates a tree that connects the start location to the goal location, where the nodes are collision-free, overlapping, square-shaped areas/volumes. The goal location is located inside the root of the tree, and start location is located at a leaf node. Then, the reference-governor-based motion control stage navigates the robot from the robot's current node to the parent node, eventually to the root node where the goal position resides.

This paper is organized as follows. In Section II, RRT-based algorithms are summarized and the reference governor concept is explained. Section III introduces RG-Trees algorithm – the main contribution of this paper. Section IV reports our experimental scenario and simulation results, as well as discussion of these results. Section V summarizes our work and discusses future research directions.

II. Background

A. *RRT-Based Motion Planning*

Rapidly exploring random trees (RRT) algorithm builds a tree of feasible paths by incrementally adding a random, collision-free edge to the tree [6]. Its algorithmic parameters such as time complexity, space complexity, completeness and optimality are discussed in Karaman and Frazzoli [7]. The algorithm works as follows: The tree is initialized to the start location as its root. Then, in a loop, a point is sampled from the free space. If the link connecting the sample point to the closest vertex in the tree is collision-free, the point and the link are added to the tree as a vertex and an edge. The loop iterates until the goal location is reached by a vertex.

There exist numerous modifications and extensions of RRT, each of which concentrating on a different aspect of the algoritm. For example, some researchers focused on the optimality aspects of the algorithm, [8]–[11], whereas some others extended the basic algorithm to solve dynamic motion planning problems [12]–[16]. On the other hand, similar to our method, some researchers [1], [17], [18] modified the RRT algorithm (or other sampling based techniques) to use regions (*funnels*) instead of points, which are then acted as the basins of attraction of some local feedback control policies. These feedback policies are connected in the essence of the sequential composition idea introduced by Burridge et al. [19]. Each region is associated with a feedback control policy, which is responsible for navigating the robot inside the area/volume to a final location which is also located inside a different but sequentially connected region.

B. Reference Governors

The reference governor is an add-on scheme for enforcing pointwise-in-time state and control constraints by modifying the reference command to a well-designed closed loop system [20], as illustrated in Fig. 2. It allows to design the controller without regard to the constraints, emphasizing controller simplicity, speed of response, robustness and disturbance rejection. Then, the reference governor is used in a cascade configuration around the closed loop system to ensure constraint enforcement [5]. Consider a closed loop system given with state space model

$$\begin{aligned} q_{t+1} &= Aq_t + Br_t, \\ y_t &= Cq_t + Dr_t, \end{aligned} \tag{1}$$

and a constraint set, $y_t \in Y \; \forall t$. The task of the reference governor is to find the optimal modified reference signal, $\tilde{r}_t$, given the initial state q_t, such that $\tilde{r}_t$ is as close to r_t as possible and no constraint violation occurs in the consequent motion.

Definition 1: The pair of an initial state and a reference signal, $(q_0, \tilde{r})$, is output admissible if, starting from the initial state q_0 and keeping the reference constant at $\tilde{r}$, no constraint violation occurs in the consequent motion.

Definition 2: Given a dynamic system and constraint set Y, the set of all output admissible pairs is called the Maximal Output Admissible Set (MOAS), O_∞ [21].

Algorithmic calculation of MOAS is explained in detail in [21]. We have from [20] that if

- the pair (C, A) is observable,
- A is asymptotically stable,
- constraint set Y is a polytope containing the equilibrium point in its interior, i.e., $Y = \{y \mid Sy \leq s\}$ [1]

[1]Note that the vector comparison operator $\leq$ here indicates that all elements of the LHS are smaller than or equal to the corresponding element in the RHS.

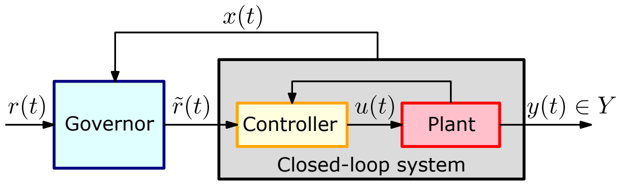

Fig. 2. Block diagram of a reference governor applied to a closed loop system [22]

MOAS turns out to be a polytope:

$$O_\infty = \{(q_0, \tilde{r}) \mid H_q q_0 + H_r \tilde{r} \leq h\}, \tag{2}$$

for which H_q, H_r and h can be calculated offline, if the system is time invariant and the constraint set is static.

For MOAS calculated in (2), the Scalar Reference Governor is formulated as follows: the modified reference $\tilde{r}$ is initialized to an initially output admissible value given the initial state q_0. Then, it is updated in a loop according to the following equation:

$$\tilde{r}_t = \tilde{r}_{t-1} + \kappa_t (r_t - \tilde{r}_{t-1}), \tag{3}$$

where κ_t is maximized subject to

- $0 \leq \kappa_t \leq 1$
- $(q_t, \tilde{r}_t)$ is output admissible, i.e., $H_q q_t + H_r \tilde{r}_t \leq h$

III. RG-Trees

RG-Trees algorithm consists of two stages. In the first stage, a tree structure that is composed of connected square shaped regions is generated using the map of the environment, ignoring the robot motion dynamics. First region is created around the goal location. Then, tree extension is repeated until one of the newly created regions covers the start location. In this way we ensure that there is a “safe connection” between the start location and the goal location.

In the second stage, a reference governor based control strategy that navigates the dynamic robot to goal location is used. It is important to note that this algorithm does not generate an open-loop trajectory. Instead, it calculates a set of connected regions where the robot can be controlled safely. The trajectory of motion is then determined by the equations of motion of the robot.

A. Tree Generation

The aim of the tree generation stage is to cover the configuration space, fully or partially, by connected nodes (regions) for which the robot’s motion can be controlled independently, to construct a global motion planner [1], [18]. In [1], configuration space is covered by balls probabilistically, and a graph structure is constructed. Then, global navigation problem is reduced to a search problem on the graph. In [18], instead of a graph, a tree structure is used and tree generation is stopped as soon as initial and goal configurations are connected, emphasizing computational simplicity over path optimality. In these methods the nodes are chosen to be circular, because of the applied control methods (Navigation functions in [1] and Lyapunov function based controller in [18]).

In this work, we use a reference governor based controller. Since finite determination of reference governors is guaranteed under convex polygonal constraint set [20], this algorithm's nodes are not circular. For the sake of simplicity and feasibility, we adopted square-shaped regions. Note that one can use the algorithm using any convex polygon or a set of predetermined convex polygons, with only slight modifications.

We illustrate the tree generation algorithm in Fig. 3. The algorithm starts from the goal location. A node is generated around that point, and this node is expanded (See Sec III-A.2). Then, until a stop condition is satisfied, the following procedure is repeated:

- a random point is sampled from the space which is not covered by a node
- from the covered area, the closest point to this sample is calculated
- a new node is generated and expanded around that point

Termination condition can be that the configuration space is probabilistically covered [1]. However, in the example, tree generation is terminated as soon as the start location is covered by a node, as in [18]. The rationale behind this choice is that tree generation algorithm is incremental, if the robot leaves the covered space, new nodes can be added online, and the algorithm is sufficiently fast for a variety of systems to be performed in real time.

1) Node Generation: When a new node is to be generated around a point, the closest point on an obstacle to that point is calculated. Then, a square region is constructed with the former point being its center and the latter being its first corner.

The corner point is stored as the offset of the node's reference frame, (x_0, y_0), and the angle between the first edge and horizontal is stored as the node's rotation, θ, see Fig. 4. Moreover, edge length of that square is stored as the node's scale.

2) Node Expansion: Usage of larger nodes yields two major advantages. Firstly, since larger nodes cover larger areas of the free space, they result in a more sparse tree. Secondly, they allow the robot to move faster. The reference governor ensures that the robot's motion is confined to the current node, which implicitly incurs a velocity constraint on the robot. Larger nodes result in looser constraints.

For the sake of simplicity, the nodes are expanded in discrete steps. At each step, the node's scale is multiplied by a constant factor γ, keeping the offset, i.e. the corner touching the obstacle, and the rotation unaltered. If the expanded node collides with an obstacle, the last expansion is reverted. If no collision occurs, the procedure repeats.

Larger values for multiplication constant γ yield lower calculation time with lower spatial resolution, and smaller values yield better spatial resolution at the expense of slower calculations. A constant value of $\gamma = 1.2$ resulted in satisfactory results in our experiments.

Algorithm 1 RG Tree Generation

```
1:  $GoalNode \leftarrow$ GenerateSquareRegion($q_{goal}$)
2:  $GoalNode \leftarrow$ Expand($GoalNode$)
3:  $G \leftarrow$ InitializeTree($GoalNode$)
4:  for $k = 1$ to $K$ do
5:      $q_{rand} \leftarrow$ SampleFree()
6:      $q_{nearest} \leftarrow$ Nearest($G, q_{rand}$)
7:      $Node_q \leftarrow$ GenerateSquareRegion($q_{nearest}$)
8:      $Node_q \leftarrow$ Expand($Node_q$)
9:      $G$.InsertNode($Node_q$)
10:     if $q_{init} \in G$ then
11:         return G
12:     end if
13: end for
14: Return G
```

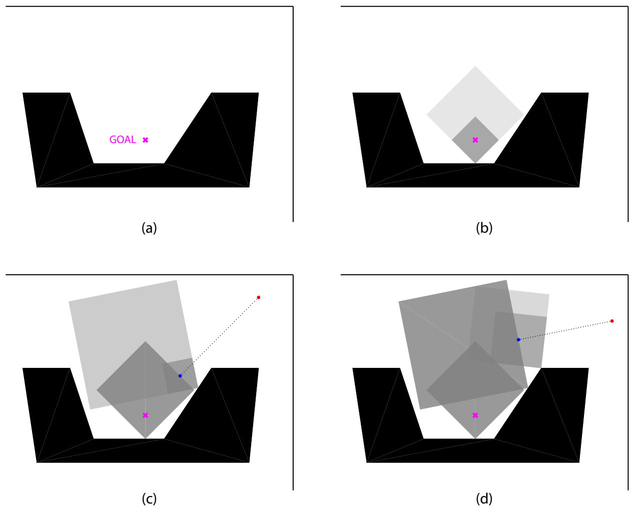

Fig. 3. RG-Tree generation algorithm: (a) Initial map of the arena, (b) A node generated(dark grey) and expanded(light grey) around goal position, (c) A random point(red) sampled from the free space, closest point to the previous tree(blue) calculated, a node generated(dark grey) and expanded(light grey) around that point, (d) Another node is generated and expanded as in (c).

B. Motion Control

Gilbert and Kolmanovsky [5] proposed a hybrid reference governor method, that divides the configuration space into overlapping, convex sets, and designs a *different* reference governor for each set. However, calculation of MOAS for a given dynamical system and constraint set is computationally expensive. In our study, for computational efficiency and considering limitations in real-time motion planning applications, we propose a method to minimize the number of MOAS calculations. Observe the following:

- MOAS calculation is linear in the constraint set,
- If the equations of motion of the system are symmetric under rotation, MOAS can be rotated (and translated),
- A reference governor with a large number of constraints can be divided into two or more reference governors which address different constraints of the older reference governor.

The first statement can be rewritten as follows: If

$$MOAS\left(\{y \mid Sy \leq s\}\right) = \{(q_0, \tilde{r}) \mid H_q q_0 + H_r \tilde{r} \leq h\}, \quad (4)$$

for a particular system, then

$$MOAS\left(\{y \mid Sy \leq \alpha s\}\right) = \{(q_0, \tilde{r}) \mid H_q q_0 + H_r \tilde{r} \leq \alpha h\} \ \forall \alpha. \quad (5)$$

That is, for all square-shaped positional constraints of the form

$$\{(x, y) \mid 0 \leq x \leq \alpha,\ 0 \leq y \leq \alpha\}, \quad (6)$$

it would suffice to calculate MOAS for the constraint set

$$\{(x, y) \mid 0 \leq x \leq 1,\ 0 \leq y \leq 1\}. \quad (7)$$

The second statement indicates that, since the double integrator system is symmetric under rotation, any square-shaped positional constraint set can be converted into the form given in (6). For example, for a 2D dynamic robot with the canonical state variable $q = [x\ y\ \dot{x}\ \dot{y}]^T$, for the square node given in Fig. 4, conversion is given by

$$\begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & \cos\theta & -\sin\theta \\ 0 & 0 & \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} \bar{x} \\ \bar{y} \\ \dot{\bar{x}} \\ \dot{\bar{y}} \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \\ 0 \\ 0 \end{bmatrix} \quad (8)$$

The last statement allows that the square-shaped positional constraints, required by the algorithm, constitute one reference governor, RG1, while all other constraints such as velocity and control input limitations constitute another reference governor, RG2, if necessary. Then, for any node of the generated tree, RG1 is rotated, translated and scaled accordingly while RG2 is used directly.

The task of motion control part is to asymptotically steer the robot from CurrentNode to NextNode. Therefore, the medium-term reference r_t should lie inside the intersection of these nodes. We choose it to be the centroid of the intersection region. When the robot enters NextNode; CurrentNode, NextNode and r_t are updated accordingly. Motion control is summarized in Algorithm 2.

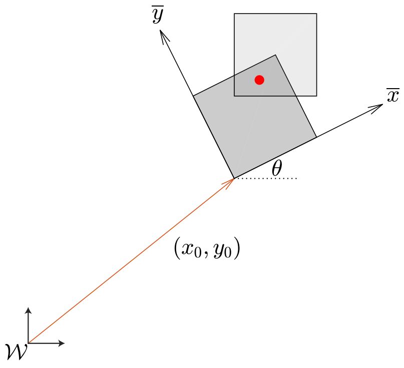

Fig. 4. Two consecutive nodes, CurrentNode(dark grey) and NextNode(light grey), generated by the algorithm. The centroid of the intersection(red point) is the set-point in CurrentNode. $(\bar{x}, \bar{y})$ is CurrentNode's coordinate frame, offset by (x_0, y_0) and rotated by θ from the global frame.

Algorithm 2 RG Tree Execution

```
1: $CurrentNode \leftarrow G$.LastNode()
2: $NextNode \leftarrow CurrentNode$.Parent()
3: $r \leftarrow$ Centroid($CurrentNode, NextNode$)
4: $\tilde{r} \leftarrow q_{init}$
5: $RG1 \leftarrow$ ReferenceGovernor($UnitSquareConstraints$)
6: $CurrentRG \leftarrow$ RTS($RG1, CurrentNode$)    ▷ Rotate, Translate and Scale RG1 according to CurrentNode
7: $NextRG \leftarrow$ RTS($RG1, NextNode$)
8: while $q_{goal}$ not reached do
9:     if in $GoalNode$ then
10:        $r \leftarrow q_{goal}$
11:    else if OutputAdmissible($q, NextRG$) then
12:        $CurrentNode \leftarrow NextNode$
13:        $NextNode \leftarrow CurrentNode$.Parent()
14:        $r \leftarrow$ Centroid($CurrentNode \cap NextNode$)
15:        $CurrentRG \leftarrow NextRG$
16:        $NextRG \leftarrow$ RTS($RG1, NextNode$)
17:    end if
18:    $\tilde{r} \leftarrow$ ModifiedReference($q, r, \tilde{r}, CurrentRG$)
19:    $\tilde{r} \leftarrow$ ModifiedReference($q, r, \tilde{r}, RG2$)
20: end while
```

IV. Implementation, Results and Conclusions

This section presents our simulation results, illustrating the flow of the RG-Trees algorithm and its feasibility.

A. Robot Motion Model

In the simulation, a fully-actuated, double-integrator, planar, point robot model is used. State vector of the robot is $q = [x\ y\ v_x\ v_y]^T$, and the input vector is $u = [a_x\ a_y]$. Positional components of the robot are measured. The system is discretized for $T_s = 0.05s$ using forward Euler method.

The reference governor works on the closed-loop system. In order to emphasize the effectiveness of the algorithm, as the inner loop controller, an underdamped PD controller is chosen. The controller's parameters are $K_p = 4.1$ and $K_d = 2.2$. The overall discrete time state space model is as follows:

$$A = \begin{bmatrix} 1 & 0 & 0.05 & 0 \\ 0 & 1 & 0 & 0.05 \\ -0.205 & 0 & 0.89 & 0 \\ 0 & -0.205 & 0 & 0.89 \end{bmatrix}, \ B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0.205 & 0 \\ 0 & 0.205 \end{bmatrix},$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \ D = 0, \quad (9)$$

B. Simulation Results

RG-Trees algorithm requires a collision check module, that checks whether a square-shaped node collides with an obstacle or the boundary, to run properly. For the sake of simplicity, polygonal obstacles and arena boundaries were chosen, although arbitrarily shaped environment with a proper collision checking module would work.

Arena boundary is square shaped, with an edge length of 12 m. In the sample environment, there exist six different

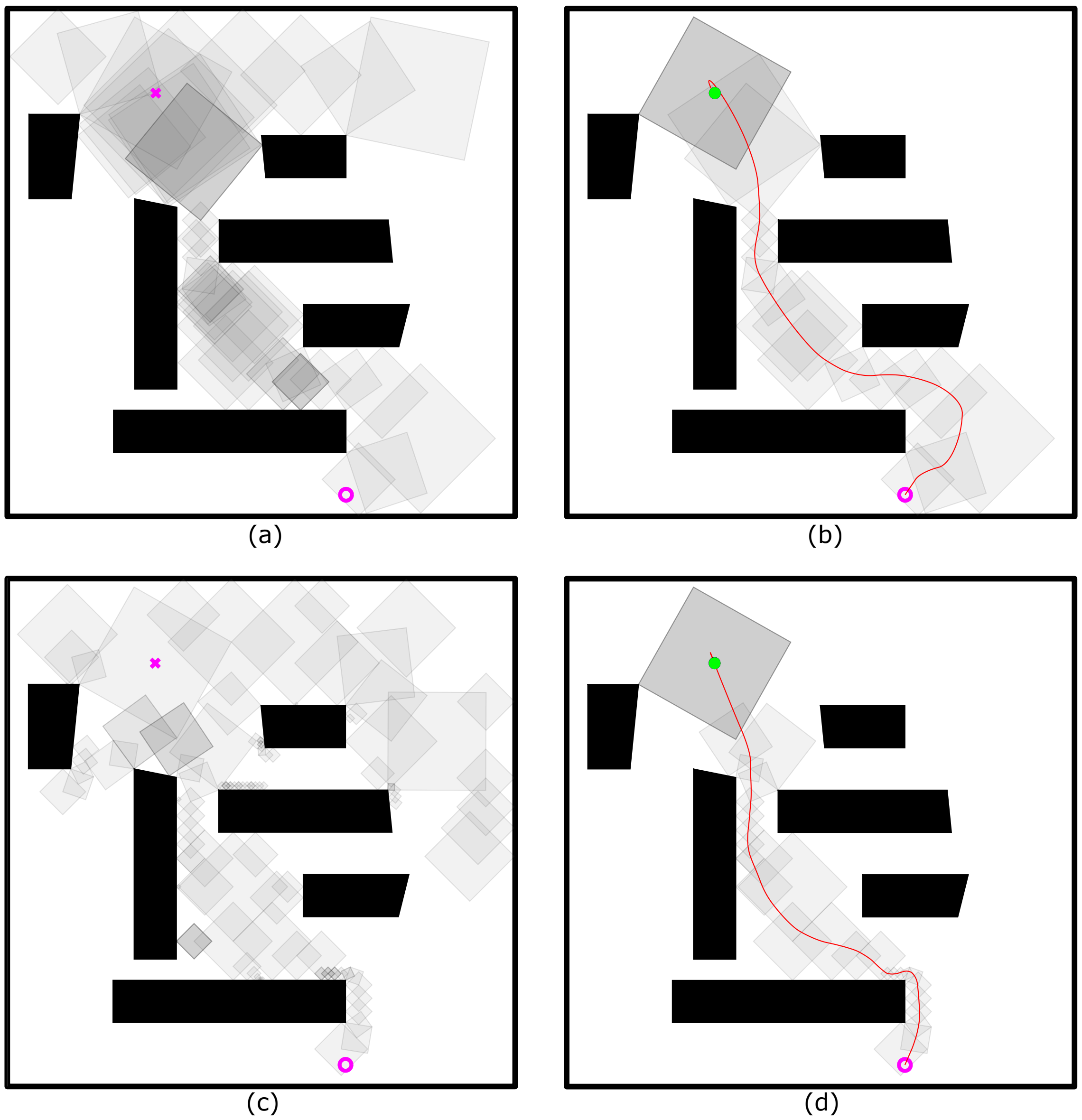

Fig. 5. (a) The tree generated by the algorithm. The algorithm is started from goal location (pink x) and terminated as soon as the start location (pink o) is covered by a node. 47 overlapping, square-shaped nodes were generated. (b) The trajectory followed by the dynamic robot. 18 of the 47 nodes generated were used in the path. (c) The tree generated when node expansion is disabled using the same random seed as in (a). 120 nodes were generated. (d) The trajectory of the robot for the tree given in (c). 28 nodes are used.

obstacles. In these simulations, the robot's initial location is $(8, 0.5)$ and goal location is $(3.5, 10)$. The arena is illustrated in Fig. 5.

In the results presented in this paper, we implemented the algorithm in MATLAB on an Intel i7 3.4 GHz computer running Windows OS. The algorithm begins the tree generation from the goal location, and terminates as soon as the start location is covered by a node. In this specific simulation, RG-Tree generation algorithm explored the start location after generating 47 nodes, in just under 0.3 s. The resulting path consists of 18 of these 47 nodes. Repeated simulations took 0.24 seconds on average and 0.6 seconds at maximum for the given configuration.

To investigate the effects of node expansion given in section III-A.2, the simulation is repeated with node expansion disabled, using the same random seed. The algorithm generated a tree with 120 nodes in 0.20 s. The path uses 28 of these nodes. As expected, smaller nodes for the no-expansion case resulted in a slower robot motion. With node expansion disabled, the robot moved with an average speed of 0.621 m/s and reached the goal location in 20.5 s, whereas when node expansion enabled, the average speed is 0.920 m/s and the robot reaches the goal in 15.4 s. Simulation results are given in Fig. 5 and Table I. It is important to remark that MOAS calculation algorithm for the system given in (9) and constraint set given in (7) took 1.2 s – 4x the time it took to generate the whole tree. However, since both the motion model and constraint set are known and time

TABLE I
SIMULATION RESULTS WITH AND WITHOUT EXPANDING NODES

Expansion Mode	# Nodes	CPU Time	Path Depth	Average Speed	Arrival Time
ON	47	0.3 s	18	0.92 m/s	15.4 s
OFF	120	0.2 s	28	0.62 m/s	20.5 s

invariant, we were able to calculate and save MOAS off-line before the simulations. Then, instead of calculating on-line, the simulations loaded MOAS from the memory. It is also important to note that, instead of calculating a different MOAS for each of the 15 nodes on the path, which would require on-line calculation and take approximately 18 second, the algorithm calculated a single MOAS, off-line, and used that MOAS for all nodes by rotating, translating and scaling appropriately.

V. DISCUSSION AND FUTURE WORK

In this paper, we have introduced the RG-trees feedback motion planning method which synthesizes sampling based planning techniques and reference governors. The method mainly produces a random tree of connected square regions–rotated, translated and scaled under certain conditions. Inside each region, we navigate the robot using a reference-governor-based feedback control policy which guarantees a collision free asymptotic convergence. We demonstrated the validity of the algorithm on MATLAB simulations on 2D environment. However, the algorithm can be directly generalized to higher dimensions by incorporating higher dimensional polygons (e.g. cubes for 3D environment). In the near future we would like to implement our method on higher dimensional spaces and test the feasibility of the approach.

One of the limitations of the current method is that it is limited to systems where the dynamics are linear or feedback linearizable. Our ultimate goal in the near future is to generalize this concept to systems with non-linear and non-holonomic robotic systems. Another aspect that we would like to address is to improve the method by directly handling uncertainties and noises (measurement and process) which are inevitable in real world applications.

ACKNOWLEDGMENT

Ferhat Golbol was supported by the National Scientific and Technological Council of Turkey (TUBITAK).

REFERENCES

[1] L. Yang and S. M. LaValle, "A framework for planning feedback motion strategies based on a random neighborhood graph," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 1. IEEE, 2000, pp. 544–549.

[2] O. Arslan and D. E. Koditschek, "Smooth extensions of feedback motion planners via reference governors," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 4414–4421.

[3] C. Petersen, A. Jaunzemis, M. Baldwin, M. Holzinger, and I. Kolmanovsky, "Model predictive control and extended command governor for improving robustness of relative motion guidance and control," in *Proc. AAS/AIAA space flight mechanics meeting*, 2014.

[4] H. Park, S. Di Cairano, and I. Kolmanovsky, "Model predictive control for spacecraft rendezvous and docking with a rotating/tumbling platform and for debris avoidance," in *American Control Conference (ACC), 2011*. IEEE, 2011, pp. 1922–1927.

[5] E. G. Gilbert and I. V. Kolmanovsky, "Set-point control of nonlinear systems with state and control constraints: A lyapunov-function, reference-governor approach," in *Decision and Control, 1999. Proceedings of the 38th IEEE Conference on*, vol. 3. IEEE, 1999, pp. 2507–2512.

[6] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[7] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[8] A. Perez, R. Platt, G. Konidaris, L. Kaelbling, and T. Lozano-Perez, "Lqr-rrt*: Optimal sampling-based motion planning with automatically derived extension heuristics," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 2537–2542.

[9] B. Akgun and M. Stilman, "Sampling heuristics for optimal motion planning in high dimensions." Georgia Institute of Technology, 2011.

[10] F. Islam, J. Nasir, U. Malik, Y. Ayaz, and O. Hasan, "Rrt*-smart: Rapid convergence implementation of rrt* towards optimal solution," in *Mechatronics and Automation (ICMA), 2012 International Conference on*. IEEE, 2012, pp. 1651–1656.

[11] O. Arslan, V. Pacelli, and D. E. Koditschek, "Sensory steering for sampling-based motion planning," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2017, pp. 3708–3715.

[12] J. Kim and J. P. Ostrowski, "Motion planning a aerial robot using rapidly-exploring random trees with dynamic constraints," in *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, vol. 2. IEEE, 2003, pp. 2200–2205.

[13] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The international journal of robotics research*, vol. 20, no. 5, pp. 378–400, 2001.

[14] S. Karaman and E. Frazzoli, "Sampling-based optimal motion planning for non-holonomic dynamical systems," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 5041–5047.

[15] J. J. Park and B. Kuipers, "Feedback motion planning via non-holonomic rrt* for mobile robots," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 4035–4040.

[16] S. Dalibard, A. El Khoury, F. Lamiraux, A. Nakhaei, M. Taïx, and J.-P. Laumond, "Dynamic walking and whole-body motion planning for humanoid robots: an integrated approach," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1089–1103, 2013.

[17] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, "Lqr-trees: Feedback motion planning via sums-of-squares verification," *The International Journal of Robotics Research*, vol. 29, no. 8, pp. 1038–1052, 2010.

[18] E. Ege and M. M. Ankarali, "Feedback motion planning of unmanned surface vehicles via random sequential composition," in review.

[19] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek, "Sequential composition of dynamically dexterous robot behaviors," *The International Journal of Robotics Research*, vol. 18, no. 6, pp. 534–555, 1999.

[20] I. Kolmanovsky, E. Garone, and S. Di Cairano, "Reference and command governors: A tutorial on their theory and automotive applications," in *American Control Conference (ACC), 2014*. IEEE, 2014, pp. 226–241.

[21] E. G. Gilbert and K. T. Tan, "Linear systems with state and control constraints: The theory and application of maximal output admissible sets," *IEEE Transactions on Automatic control*, vol. 36, no. 9, pp. 1008–1020, 1991.

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