Proceedings of 2017 IEEE
International Conference on Mechatronics and Automation
August 6 - 9, Takamatsu, Japan

A Novel RRT*-Based Algorithm for Motion Planning in Dynamic Environments

Olzhas Adiyatov and Huseyin Atakan Varol
Department of Robotics and Mechatronics
Nazarbayev University
Astana, Kazakhstan
{*oadiyatov, ahvarol*}*@nu.edu.kz*

***Abstract*— Sampling-based motion planning has become a powerful framework for solving complex robotic motion-planning tasks. Despite the introduction of a multitude of algorithms, most of these deal with the static case involving non-moving obstacles. In this work, we are extending our memory efficient RRT*FN algorithm to dynamic scenarios. Specifically, we retain the useful parts of the tree (the data structure storing the motion plan information) after a dynamic obstacle invalidates the solution path. We then employ two greedy heuristics to repair the solution instead of running the whole motion planning process from scratch. We call this new algorithm, RRT*FN-Dynamic (RRT*FND). To compare our method to the state-of-the-art motion planners, RRT* and RRT*FN, we conducted an extensive set of benchmark experiments in dynamic environments using two robot models: a non-holonomic mobile robot and an industrial manipulator. The results of these experiments show that RRT*FND finds the solution path in shorter time in most of the cases and verifies the efficacy of it in dynamic settings.**

***Index Terms*— Path planning, motion planning, sampling-based methods, dynamic environments, rapidly-exploring random trees.**

I. Introduction

With the robots getting ubiquitous, effective motion planning in complex environments with dynamic obstacles is becoming essential. Motion planning aims to find a sequence of discrete robot configurations from the initial to the goal state complying with constraints imposed by the environment and internal dynamics of the system. Preliminary research on the topic included cell decomposition based methods [1], artificial potential fields [2] and visibility graphs [3]. The motion-planning problem becomes a harder challenge in higher dimensional configuration spaces, prevalent in robotics. Due to the need for explicit representation of the obstacles in the configuration space, these planning algorithms are not suitable for high dimensional spaces with high number of obstacles.

Currently, sampling-based planners are the leading family of algorithms for higher dimensional motion planning. Sampling-based methods offer significant computational savings over complete motion-planning ones, since an explicit representation of the environmental constraints (i.e. obstacles) is not necessary. Instead, sampling-based planners employ collision checking in order to verify the feasibility of a candidate trajectory, and connect a set of collision-free points in order to create a graph of feasible trajectories. This graph is then used to find a collision-free path between the initial and goal states. Most popular sampling-based planners are Probabilistic Roadmaps (PRMs) [4] and Rapidly-Exploring Random Trees (RRTs) [5]. Both RRT and PRM are probabilistically complete. They converge to the feasible solution as the number of iterations approaches infinity. Their primary difference is the way they connect the sampled points to generate a graph.

Single query approaches such as RRT, which employ incremental sampling, are advantageous for scenarios with differential and/or integral constraints. Basic RRT algorithm grows a tree of feasible trajectories rooted at the initial state incrementally and returns a solution when one of the collision-free trajectories reaches the goal region. RRT does not use a metric to measure the motion optimality between the initial state and the other nodes. Due to its emphasis on exploration rather than exploitation, RRT tries to find a feasible solution as quickly as possible.

To address this issue, asymptotically optimal RRT* [6] extends RRT by performing local optimization to improve the results of the global motion-planning problem. To curb the memory need and speed up the search of neighborhood nodes in asymptotically optimal planning, RRT*FN [7] limits the maximum number of nodes in the tree and removes a node to add a new node in its incremental sampling procedure.

In real-world scenarios information about the environment is often incomplete, fragmented, and only updated iteratively over the course of time, occasionally with false information. The most straightforward approach of tackling robot motion-planning problem in such cases is to treat all obstacles as static and re-run the motion planner for static environments with the updated information. This approach, depending on the planning algorithm, will provide a feasible or even an optimal solution. However, the computation of the whole solution from scratch will take a considerable amount of time, which might not be available once the motion is started. Therefore, there is an interest in the development of computationally efficient motion planners tailored for problems with unknown and dynamic obstacles. Frequently related work in this topic is an extension of the static sampling-based motion-planning algorithms [8], [9], but there are algorithms designed for this very purpose as well. D* [10] and D* Lite [11] are examples of such deterministic algorithms for motion planning. The main shortcoming of these is the curse of dimensionality.

978-1-5090-6759-6/17/$31.00 ©2017 IEEE

To alleviate this, researchers came up with hybrid algorithms (an adaptation of PRM with D* and Anytime D*) for robot motion planning in dynamic environments [12]. There are also RRT-based motion planners for dynamic environments. Ferguson et al. [13] presented DRRT, which rapidly removes the invalidated nodes and regrows the tree until a new solution is found.

In this paper we present an extension of RRT*FN algorithm called RRT*FN-Dynamic (RRT*FND). Our algorithm is geared towards motion planning in the presence of dynamic obstacles. In our algorithm once a dynamic obstacle invalidates a part of the tree, those nodes are removed. However, the nodes on the solution path not colliding with the obstacle are kept intact. Then our algorithm tries to reconnect the tree to one of the solution path nodes disconnected from the main tree by the obstacle. Even though our algorithm has similarities with and indeed is inspired from the DRRT, it differs in three main areas: First our algorithm is based on the RRT*FN, which has a limit on the maximum number of nodes. This allows us to remove the invalidated nodes rapidly even in high dimensional and complex problems. Secondly, RRT*FN solves an optimal path planning problem instead of the feasible one solved by DRRT. Thirdly, we take into account the nodes on the solution path and solve the smaller problem of reconnecting the intact tree to the solution path.

The rest of the paper is organized as follows. Next section will define the problem of motion planning in dynamic environments and also introduce the notation used throughout the paper. Afterwards an overview of RRT* and RRT*FN will be provided. Then our variant RRT*FND is presented thoroughly. After showing the efficacy of RRT*FND via a battery of simulation experiments, the paper is concluded.

II. Problem Definition and Notation

The notation and definitions introduced in this section will be used throughout the paper. Let $P \subset \mathbb{R}^d$ be the state space of the problem, where $d \in \mathbb{N}$, $d \geq 2$. $O \subset P$ is the static obstacle region. Furthermore, $D(t) \subset P$ is the dynamic obstacle region, where time $t \geq 0$. Anticipation of movement of the obstacle in the dynamic environment is hard [14], therefore the movement of an obstacle will be treated as completely random. The unified obstacle region is characterized as $P_{obs} \subset (O \cup D(t))$. The obstacle-free region is defined as $P_{free} = P \setminus P_{obs}$. The initial state p_{init} and the final state p_{final} both belong to P_{free}. Let $\sigma \colon [0,1] \to \mathbb{R}^d$ be a continuous function and denote a path. A path is said to be collision-free, if $\sigma(\alpha) \in P_{free}$ for all $\alpha \in [0,1]$. A collision-free path is a feasible path, if $\sigma(0) = p_{init}$ and $\sigma(1) = p_{goal}$. The 3-tuple $(p_{init}, p_{final}, P_{free})$ contains the parameters, which define the path planning problem. A feasible path planning problem attempts to discover a path with bounded variation such that $\sigma(0) = p_{init}$ and $\sigma(1) = p_{goal}$, if a feasible path exists.

A cost function is needed to transform the feasible path planning problem to the optimal one. The cost function $c \colon \Psi \to \mathbb{R}_{\geq 0}$ is a monotonic, bounded, and strictly positive function for all collision-free paths. The cost function is zero, if and only if $\sigma(\alpha) = \sigma(0), \forall \alpha \in [0,1]$ (i.e. the system remains in the initial state without making any movement). Optimal path planning becomes an optimization problem with the triplet of parameters $(p_{init}, p_{final}, P_{free})$ and a corresponding cost function $c \colon \Psi \to \mathbb{R}_{\geq 0}$. It seeks to find a feasible path σ_{best} between the initial and goal states such that

$$c(\sigma_{best}) = \min\{c(\sigma) : \sigma \text{ is feasible}\}$$

In the context of this paper, all path planning algorithms return a tree $\tau = (V, E)$ which is a specialized graph consisting of vertices (nodes) $V \subset P_{free}$ and edges $E \in V \times V$. The solution path is computed from the tree by finding the shortest path on this tree.

III. Background: RRT* and RRT*FN

RRT* is an asymptotically optimal extension of the RRT. Since RRT acts as the backbone of the RRT*, it is well-suited to describe it first. In RRT, a tree is initialized at the start state. A new state is sampled randomly at each iteration and the nearest neighbor of this new sample is found. If a newly sampled state is not reachable within a single step (due to system dynamics, motion constraints and obstacles) from the nearest neighbor, a new state is generated towards the new sample by the `Steer` routine such that a single step reachability condition is fulfilled. This new single-step reachable state will be added to the tree if there is an obstacle-free path segment from its nearest neighbor. Ultimately, RRT will explore the state space and provide a feasible solution.

RRT* extends RRT with three new concepts. The first one is the local neighborhood of a newly added node. The second one is the cost function, which provides the total cost from the initial state to a node in the tree. The third one is the rewiring procedure, which rewires the local neighborhood of the newly added node such that the cost to the initial state is decreased.

Now we will describe the operation of the RRT* in detail. RRT* first tries to connect the new sample to the nearest node. However, instead of setting the nearest node as the parent of p_{new} right away, the algorithm determines the nodes P_{near} in the local neighborhood of p_{new}. P_{near} can be found by selecting the nodes inside a region around p_{new} with a radius r. Subsequently, `ChooseParent` routine selects the best parent for p_{new}. The selection of the parent is detailed in Algorithm 3. Specifically, p_{new} is connected to the parent, which minimizes the total cost from p_{init} to p_{new}. This is one of the two instances where RRT* optimizes the whole trajectory. After p_{new} is inserted to the tree as a new node, the local neighborhood is optimized again with the `ReWire` routine, as shown in Algorithm 4. In this case, the total cumulative cost of the path from p_{init} to all nodes in P_{near} is calculated assuming that p_{new} is the parent node of the nodes in P_{near}. If the assignment of p_{new} as the parent decreases the total cost from p_{init} to $p_{near} \in P_{near}$, then the `Reconnect` removes the edge between p_{near} and its parent node and creates an edge between p_{near} and p_{new}, the new parent node. This way, RRT* also utilizes the newly inserted node to shorten

Algorithm 1 $\tau = (V, E) \leftarrow$ RRT*FN(p_{init}, p_{goal})

$\tau \leftarrow$ InitializeTree()
$\tau \leftarrow$ InsertNode(p_{init}, τ)
for $i = 1$ to $i = N$ **do**
  $p_{rand} \leftarrow$ Sample(i)
  $p_{nearest} \leftarrow$ Nearest(τ, p_{rand})
  $p_{new} \leftarrow$ Steer$(p_{nearest}, p_{rand})$
  if ObstacleFree$(p_{nearest}, p_{new})$ **then**
    $P_{near} \leftarrow$ Near(τ, p_{new})
    $p_{min} \leftarrow$ ChooseParent$(P_{near}, p_{nearest}, p_{new})$
    $\tau \leftarrow$ InsertNode(p_{min}, p_{new}, τ)
    $\tau \leftarrow$ ReWire$(\tau, P_{near}, p_{min}, p_{new})$
    if NumberOfNodes$(\tau) > M$ **then**
      $\tau \leftarrow$ ForcedRemoval$(\tau, p_{new}, p_{goal})$
    end if
  end if
end for
return τ

Algorithm 2 $\tau = (V, E) \leftarrow$ ForcedRemoval$(\tau, p_{new}, p_{goal})$

$P_{childfree}$ = ChildlessNodes(τ)
p_{best} = BestPathLastNode(τ, p_{goal})
$p_{remove} =$ Random$(P_{childfree} \backslash \{p_{best}, p_{new}\})$
$\tau \leftarrow$ RemoveNode(p_{remove}, τ)
return τ

Algorithm 3 $p_{min} \leftarrow$ ChooseParent$(P_{near}, p_{nearest}, p_{new})$

$p_{min} \leftarrow p_{nearest}$
$c_{min} \leftarrow$ Cost$(p_{nearest}) + c(p_{new}, p_{nearest})$
for $p_{near} \in P_{near}$ **do**
  if ObstacleFree(p_{near}, p_{new}) **then**
    $c' =$ Cost$(p_{near}) + c(p_{new}, p_{near})$
    if $c' < c_{min}$ **then**
      $p_{min} \leftarrow p_{near}$
      $c_{min} \leftarrow c'$
    end if
  end if
end for
return p_{min}

Algorithm 4 $\tau \leftarrow$ ReWire$(\tau, P_{near}, p_{min}, p_{new})$

for $p_{near} \in P_{near} \backslash p_{min}$ **do**
  if ObstacleFree(p_{new}, p_{near}) and Cost$(p_{new}) + c(p_{new}, p_{near}) <$ Cost(p_{near}) **then**
    $\tau \leftarrow$ Reconnect$(p_{new}, p_{near}, \tau)$
  end if
end for
return τ

the total path to the nodes in the vicinity. The algorithm is run for N iterations.

One of the shortcomings of RRT* is the increasing number of nodes needed to achieve optimality. From a computational perspective, this puts a burden to the memory needs in high dimensional spaces requiring increased number of iterations to explore a larger search space. It also slows down motion planning due to the nearest neighbor search involving a tree with more nodes. RRT*FN addresses these issues by limiting the number of nodes in the tree. Specifically, RRT*FN leverages childless nodes as candidates for node removal. The underlying motivation for this scheme is the simplicity of removal of such nodes. If a childless node disappears, there is no need to deal with orphaned set of nodes in the tree. Except the node at the goal state, none of the nodes in the solution path is a childless node.

The pseudocode of RRT*FN is shown in Algorithm 1. As the name implies, RRT*FN introduces a new parameter which is the maximum number of nodes M allowed in the tree. Until the maximum number of nodes in the tree is reached, RRT*FN operates identical to RRT*. However, at the moment when the number of nodes exceeds M, RRT*FN starts to employ ForcedRemoval routine (see Algorithm 1). In short, after the insertion of a new node to the tree, ForcedRemoval removes one childless node randomly to keep the number of the nodes fixed. There is a possibility that the last node on the solution path can be childless. Removal of it would destroy or impair the solution. Therefore, if the last node of the solution path is childless, it is excluded from random sampling for ForcedRemoval.

IV. RRT*FN-Dynamic

A straightforward approach to the path planning problem with moving and/or unknown obstacles is to update the environment model and re-run a new motion planning problem assuming that all obstacles are static. By discarding the old tree this approach loses valuable information generated during the run-time of the algorithm. Lost information includes results of obstacle collision detection routines and exploration of the configuration space by the tree. Our aim is to save important information and reuse it as much as possible. Specifically, we augment the RRT*FN with two heuristics for motion planning with moving and/or random obstacles. Pseudocode of our approach is provided in Algorithm 5.

In the initial Grow phase, RRT*FN is executed. This step incorporates generation of a regular tree until a solution is discovered. After the initial solution is obtained, it is advantageous to grow the tree until the motion start time. This way, the solution path σ, which is implemented as a linked list that contains the path nodes starting from the p_{init}, is further optimized and the configuration space better explored. Afterwards, robot initiates movement. Once a new node $p_{current}$ on σ is reached, the obstacle information is updated. If an obstacle collision is detected in any path segment between $p_{current}$ and p_{goal}, the robot stops its movement. SelectBranch and ValidPath routines are executed to represent the valid information remaining in the corrupted tree in a computationally efficient manner. SelectBranch creates a subtree rooted at $p_{current}$ by discarding the nodes

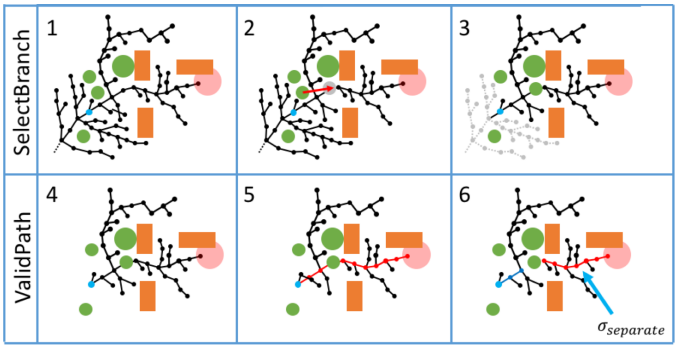

Fig. 1: Visual illustrations of SelectBranch (top row) and ValidPath (bottom row) routines.

Algorithm 5 $\tau = (V, E) \leftarrow$ RRT*FND(p_{init})

```
τ ← RRT*FN(p_init) {Grow phase}
p_current ← p_init
σ ← SolutionPath(τ, p_current)
InitMovement()
while p_current ≠ p_goal do
    D ← UpdateObtacles()
    if DetectCollision(σ, p_current) then
        StopMovement()
        τ ← SelectBranch(p_current, τ)
        p_separate ← ValidPath(σ)
        ReconnectFailed ← true
        P_near ← Near(τ, p_separate)
        for p_near ∈ P_near do
            if ObstacleFree(p_near, p_separate) then
                τ ← Reconnect(p_near, p_separate, τ)
                ReconnectFailed ← false
                break
            end if
        end for
        if ReconnectFailed = true then
            τ ← Regrow(τ, p_separate, SetBias(σ_separate))
        end if
        σ ← SolutionPath(τ, p_current)
        ResumeMovement()
    end if
    p_current ← NextNode(σ)
end while
```

from p_{init} till $p_{current}$ and their offspring. ValidPath examines the previous solution path removes all the nodes colliding with the new obstacle and their offspring, retains the usable part of the solution path connected to the p_{goal}. This part is named $\sigma_{separate}$ which starts at the node $p_{separate}$ and ends at the terminal node p_{goal}. Operation of SelectBranch and ValidPath are illustrated in Fig. 1.

The second phase consists of two opportunistic attempts, Reconnect and Regrow, to repair the solution. Firstly, Reconnect looks to the vicinity of each node of $\sigma_{separate}$ and determines whether any node from the parent tree can be connected to one of these nodes in a single step. If such node is found, the parent tree is reconnected to the orphaned path segment and all the preceding path nodes in $\sigma_{separate}$ are deleted. Secondly, Regrow is performed if Reconnect routine fails to find a solution. In Regrow, the main tree grows until there is a case where any node from the $p_{separate}$ can be reconnected to the main tree. It is not compulsory to connect the main tree to the root of the orphaned solution path $p_{separate}$. Optionally, the growth of the main tree can be biased towards the $\sigma_{separate}$. This might be a disadvantage, due to the need in careful tuning of the parameter. The nodes which did not connect to the main tree will be removed after the solution path is found.

Reconnect procedure is visualized in the left column of Fig. 2. The first subplot for the Reconnect routine depicts the state of the tree with a solution path for motion planning problem with static obstacles. Afterwards, a dynamic obstacle moves invalidating nodes including some nodes on the solution path. Reconnect connects the tree to the $\sigma_{separate}$ in a single step, thus repairing the solution path. Right column of Fig. 2 illustrates the Regrow routine. The first step shows the initial condition of the tree before the movement of the

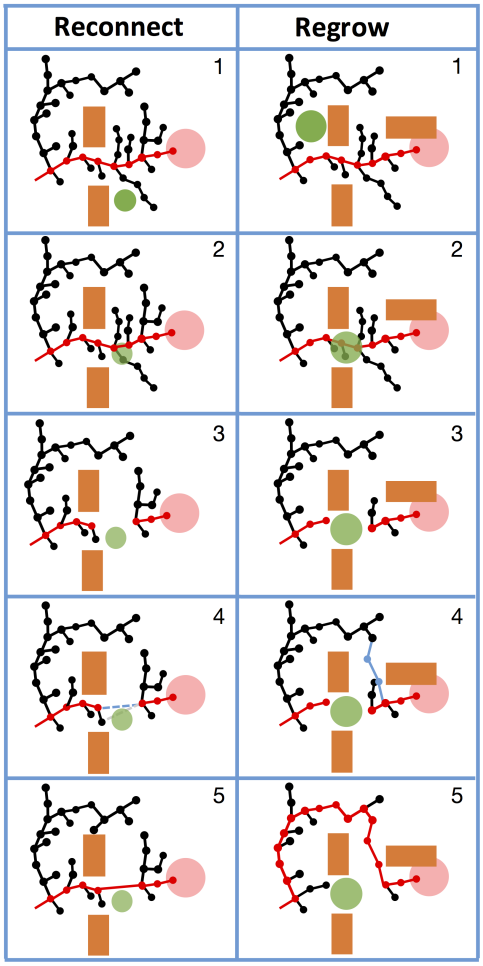

Fig. 2: Visual illustrations of Reconnect (left column) and Regrow (right column) strategies for tree repair.

dynamic obstacle. Afterwards, the obstacle blocks the narrow passage between two walls destroying the solution path. Since a single step reconnection of the tree is not possible, new nodes are added to the tree with a bias towards the nodes of the $\sigma_{separated}$. Eventually, the tree is connected to one of the nodes of $\sigma_{separated}$ resulting in a solution path significantly different than the original one. Since nodes cannot have two parents, all former ancestor nodes of the node that was connected to the main tree are deleted.

V. Simulation Experiments

An extensive set of simulations were conducted to benchmark the efficacy of RRT*FND compared to the baseline algorithms RRT* and RRT*FN. Specifically, dynamic motion planning scenarios were executed on two robot models: a mobile robot and a 6 degree of freedom industrial robotic manipulator. The simulations were performed on a desktop computer with Intel Core i7 (2.80GHz) and 4GB of memory. RRT*FN and RRT*FND were implemented using Open Motion Planning Library (OMPL) [15] as the base in C++ programming language. DART Library [16] was utilized for visualization and obstacle collision checking purposes in the industrial manipulator case study.

A. Mobile Robot

The mobile robot was modeled as a Dubins Car [17], which is a simplistic non-holonomic model with a minimum radius of turn greater than zero (see Fig. 3). The complicated kinematics with non-holonomic constraints makes the path planning problems formidable compared to the holonomic point-like mobile robot models which are frequently used for evaluating motion planners. Operating on a two dimensional map facilitates lucid illustrations of the RRT*FND operation. The kinematics of the robot is given by $\dot{x} = V\cos(\theta)$, $\dot{y} = V\sin(\theta)$ and $\dot{\theta} = \tan\phi$, where x and y are the coordinates of the center rear axle of the robot, θ is the orientation of the robot, and ϕ is the turning angle. Maximum allowable turning angle ϕ_{max} has an effect on the minimum turning radius ρ_{min} according to the relationship $\rho = L/\tan\phi$. The robot moves forward with a constant speed V.

The shape of the car is modeled as a rectangle with constant width and length. The path length between two states L was chosen as the minimization criterion:

$$L(p,\phi) = \int_0^{t_F} \sqrt{\dot{x}(t)^2 + \dot{y}(t)^2}dt$$

where $p = (x, y, \theta)$ and t_F is the duration of the motion till the robot reaches the goal. The length and width of robot car is set to 80 and 50 units, respectively. The maximum turning angle for the car was set to 32 degrees resulting in the minimum turning radius $\rho_{min} = 125$ units.

Each map as shown in Fig. 4 contains rectangular walls (static obstacles) and circles (dynamic obstacles). The yellow rectangle and the purple circle represents the mobile robot and goal region, respectively. Due to the stochastic nature of the algorithm, the benchmark procedure was organized as follows. Firstly, a solution was obtained from the initial position to the goal position on one of the maps. Then, the robot initiated movement and at each of path nodes the dynamic obstacle moved such that it corrupted a segment of the remaining path. Afterwards, for RRT* and RRT*FN algorithms, the tree was reset and the static motion planning problem was ran for 600 seconds, or until a solution was found. For RRT*FND, the algorithm detects a collision on the path and starts the repair procedures. For collision at every path segment, three algorithms were executed ten times. The maximum execution

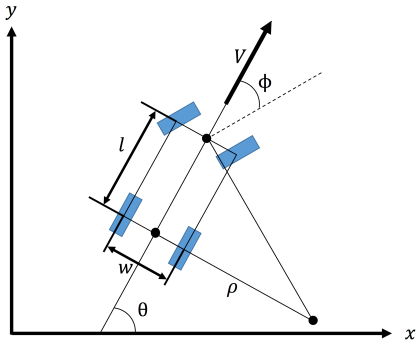

Fig. 3: Schematics of the Dubins Car model used for the simulation experiments.

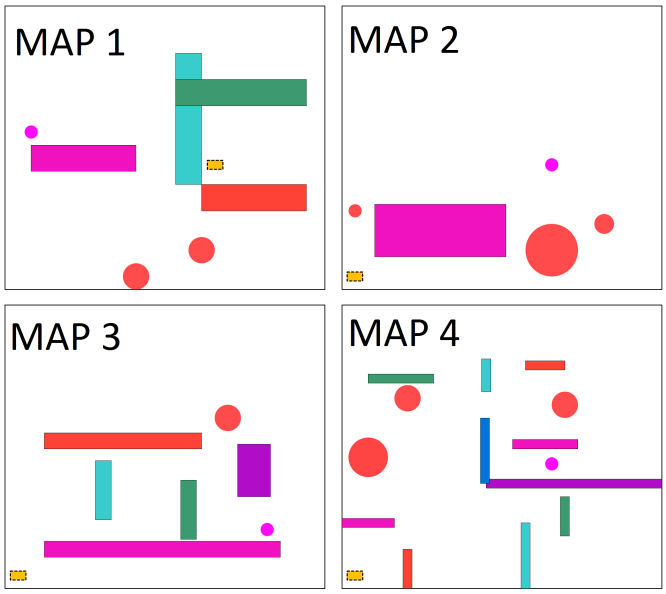

Fig. 4: Maps for the mobile robot benchmarks.

time for finding a solution was set to 600 seconds. If the goal was reached within the maximum allocated time, the trial was marked as successful and the execution time was recorded.

B. Industrial Manipulator

In the second case study, the efficacy of the method was tested using an industrial robot (Staübli TX90XL), which is a 6-axis manipulator with a maximum reach of 1450 mm. The considered scenario was a point-to-point end-effector motion of the robot in a dynamic factory setting. In the simulation, the robot was mounted on top of a table in front of a wall with the blue transparent rectangular bounding box representing another static obstacle in the reachable workspace of the robot (see Fig. 5). Similar to the mobile robot case study, the dynamic obstacle, represented with a green rectangular box, was introduced randomly on the path of the robot after the motion starts. The state vector for the robot was defined as the set of the joint angles θ_i, where $i = 1, ..., 6$. Weighted Euclidean distance metric was used to compute the cost d between two configurations θ_i and θ_j as

$$d(\theta_i, \theta_j) = \sum_{k=1}^{6} \omega_i \sqrt{(\theta_{i_k} - \theta_{j_k})^2}$$

where $\omega = \big(1, 1, 1, 0.01, 0.01, 0.01\big)$. The start and goal states were defined to be (x, y, z) position of the end-effector and computed using forward kinematics from the state vector of the robot.

VI. Results and Discussion

The results of the benchmark for the mobile robot case study are summarized in the Table I. This table contains the average time of replanning for every approach (RRT*FND, RRT* and RRT*FN). In addition, for each of the algorithms the success rate is indicated. Our method RRT*FND found the solution with 100% success rate while RRT* and RRT*FN,

	RRT*FND		RRT*		RRT*FN	
	Avg Time (s)	Success rate	Avg Time (s)	Success rate	Avg Time (s)	Success rate
Map 1	14.95	1.00	19.528	0.90	16.916	0.80
	8.142	1.00	16.592	1.00	15.38	1.00
	6.020	1.00	81.847	0.90	8.548	1.00
	5.955	1.00	12.517	1.00	70.618	1.00
	4.471	1.00	42.774	1.00	8.507	1.00
	4.443	1.00	2.795	1.00	6.216	1.00
	4.255	1.00	11.553	0.90	1.594	0.80
	2.430	1.00	11.416	0.90	75.233	0.80
	2.038	1.00	3.331	1.00	1.439	1.00
Map 2	11.64	1.00	3.677	0.90	17.06	0.90
	9.444	1.00	15.10	0.90	2.336	0.90
	9.252	1.00	8.045	0.90	19.56	0.90
	9.252	1.00	2.145	0.80	1.786	0.90
	9.112	1.00	1.733	0.80	1.762	1.00
	9.058	1.00	5.242	1.00	2.017	1.00
	9.022	1.00	62.57	0.80	0.635	0.90
	9.028	1.00	1.290	1.00	1.022	0.80
Map 3	15.18	1.00	128.1	0.90	169.8	0.90
	12.26	1.00	192.6	1.00	154.2	1.00
	10.76	1.00	199.7	1.00	165.8	0.90
	9.234	1.00	161.5	0.90	148.8	0.90
	9.112	1.00	159.7	1.00	129.9	0.90
	8.362	1.00	180.6	0.80	113.8	1.00
	6.160	1.00	97.95	0.90	106.1	1.00
	5.978	1.00	95.69	0.90	95.76	0.90
	5.736	1.00	126.2	0.70	127.8	1.00
Map 4	16.32	1.00	169.1	1.00	143.8	1.00
	6.235	1.00	223.1	0.90	120.2	1.00
	5.929	1.00	88.09	0.70	188.2	1.00
	5.919	1.00	118.7	0.90	174.0	1.00
	4.891	1.00	150.2	0.80	174.9	1.00
	1.000	1.00	72.15	0.50	100.8	1.00

TABLE I: Benchmark results for Dubins Car dynamic motion planning problem.

which restart the complete motion planning problem due to workspace change, performed worse than our method, not returning a motion plan in every case. Specifically, the success rate dropped till 50% and 80% for RRT* and RRT*FN for some instances, respectively.

RRT*FND has an initial overhead due to the `SelectBranch` and `ValidPath` routines. Therefore, in relatively simple Maps 1 and 2, computational time to repair a solution for RRT*FND was comparable to RRT* and RRT*FN. However, the computation times of RRT*FND for Maps 3 and 4 were significantly smaller than the baseline algorithms.

RRT*FND also proved to be an effective method for the industrial manipulator case study with higher degrees of freedom. The repair of the path using `Reconnect` and `Regrow` procedures of RRT*FND took on average 300 ms to repair the solution. A snapshot from one of the industrial robot simulations are shown in Fig. 5. In this figure, the blue and magenta industrial manipulators denote the initial state and the goal state, respectively. The yellow manipulator depicts the robot in motion. The blue curve represents the initial solution produced by RRT*FN. The magenta line shows the repaired path after the invalidation of the path by a dynamic obstacle.

In conclusion, RRT*FND was tested on two robot models, each of them demonstrated the viability of our method. The future work includes implementation of this dynamic motion planning framework to a physical industrial manipulator setup equipped with depth cameras for obstacle detection.

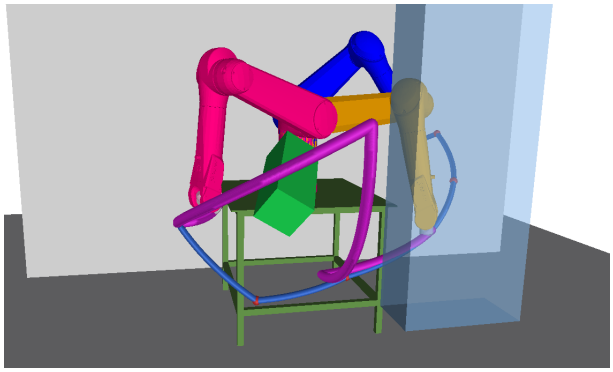

Fig. 5: Snapshot from the dynamic motion planning simulation of the industrial manipulator.

REFERENCES

[1] T. Lozano-Perez *et al.*, "Spatial planning: A configuration space approach," *IEEE Transactions on Computers*, vol. 32, no. 2, pp. 108–120, 1983.

[2] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *The International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.

[3] T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Communications of the ACM*, vol. 22, no. 10, pp. 560–570, 1979.

[4] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[5] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," 2000.

[6] 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.

[7] O. Adiyatov and H. A. Varol, "Rapidly-exploring random tree based memory efficient motion planning," in *Proc. of IEEE International Conference on Mechatronics and Automation (ICMA)*, 2013, pp. 354–359.

[8] M. Svenstrup, T. Bak, and H. J. Andersen, "Trajectory planning for robots in dynamic human environments," in *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010, pp. 4293–4298.

[9] M. Kallman and M. Mataric, "Motion planning using dynamic roadmaps," in *Proc. of IEEE International Conference on Robotics and Automation*, vol. 5, 2004, pp. 4399–4404.

[10] A. Stentz, "Optimal and efficient path planning for unknown and dynamic environments," DTIC Document, Tech. Rep., 1993.

[11] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 354–363, 2005.

[12] J. Van Den Berg, D. Ferguson, and J. Kuffner, "Anytime path planning and replanning in dynamic environments," in *Proc. of IEEE International Conference on Robotics and Automation*, 2006, pp. 2366–2371.

[13] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with RRTs," in *Proc. of IEEE International Conference on Robotics and Automation*, 2006, pp. 1243–1248.

[14] N. E. Du Toit and J. W. Burdick, "Robot motion planning in dynamic, uncertain environments," *IEEE Transactions on Robotics*, vol. 28, no. 1, pp. 101–115, 2012.

[15] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, Dec 2012, http://ompl.kavrakilab.org.

[16] G. T. G. Lab and H. R. Lab, "DART (Dynamic Animation and Robotics Toolkit)," 2016. [Online]. Available: http://dartsim.github.io

[17] S. M. LaValle, *Planning algorithms*. Cambridge University Press, 2006.