IEEE Control Systems Letters paper presented at
2023 American Control Conference (ACC)
San Diego, CA, USA. May 31 - June 2, 2023

RRT Guided Model Predictive Path Integral Method

Chuyuan Tao†, Hunmin Kim‡, and Naira Hovakimyan†

Abstract— **This work presents an optimal sampling-based method to solve the real-time motion planning problem in static and dynamic environments, exploiting the Rapid-exploring Random Trees (RRT) algorithm and the Model Predictive Path Integral (MPPI) algorithm. The RRT algorithm provides a nominal mean value of the random control distribution in the MPPI algorithm, resulting in satisfactory control performance in static and dynamic environments without a need for fine parameter tuning. We also discuss the importance of choosing the right mean of the MPPI algorithm, which balances exploration and optimality gap, given a fixed sample size. In particular, a sufficiently large mean is required to explore the state space enough, and a sufficiently small mean is required to guarantee that the samples reconstruct the optimal control. The proposed methodology automates the procedure of choosing the right mean by incorporating the RRT algorithm. The simulations demonstrate that the proposed algorithm can solve the motion planning problem for static or dynamic environments.**

I. Introduction

Motion planning problems have been widely discussed in recent years in the field of robotics, [1], [2], [3], [4], [5]. The main goal of motion planning problems is to find an optimal path for the agents to move from an initial position to a target position in known stationary environments while preventing collisions. It is challenging to solve the same problem efficiently in dynamic environments and implement the algorithms on the robotic systems in real time.

For motion planning problems, sampling-based methods have been proven to be effective for complex systems since the methods avoid calculating the derivatives of the dynamic equation and the cost function. In particular, the Probabilistic Roadmap (PRM) algorithm [6] is the first sampling-based algorithm that solves the motion planning problem. The algorithm uses a local planner to connect the sampling configuration in free space. The Rapid-exploring Random Trees (RRT) algorithm [7], [8], one of the most famous sampling-based algorithms, combines the exploration of the configuration space and the biased sampling around the goal configuration space. Most of the RRT algorithm variants can efficiently solve motion planning problems but cannot find an optimal solution. The RRT* algorithm has been developed in [9] to find an optimal solution by using incremental rewirings of the graph to provide an asymptotically optimal solution to the motion planning problems. However, compared to the RRT algorithm, the RRT* algorithm and its variants have a relatively longer execution time because the algorithm calculates the neighboring nodes and rewires the graph.

Most RRT and RRT* algorithms cannot handle dynamic environments since it requires one to abandon the current path, and derive a new path from scratch. Dynamic Rapidly-exploring Random Trees (DRRTs) algorithm [10] was developed to address the problem by trimming the original results and exploring to get the target again. In [11], the authors provide a variant of replanning RRT algorithms combined with the Multipartite RRT (MP-RRT) algorithm. The MP-RRT algorithm biases the sampling distribution towards previous useful states and analytically computes which part of the previous RRT results can be re-utilized. Yet, both algorithms could not guarantee an optimal solution to the motion planning problem since the algorithms are based on non-optimal RRT algorithms. Thus, we provide a different approach to solving optimal real-time motion problems.

One alternative way to efficiently solve optimal motion planning problems with dynamic environments is to use the Model Predictive Integral Control (MPPI) algorithm [12], [13]. By sampling the forward trajectories of dynamic systems, the MPPI algorithm avoids calculating the derivatives of the dynamic functions or the cost functions [14]. Since the forward trajectories can be sampled efficiently by Graphic Processing Units (GPUs), the algorithm can be applied to diverse robotic systems by finishing the calculation in a fixed time [15]. The MPPI algorithm enables real-time implementation by adjusting the fixed running time, whereas a longer running time reduces the optimality gap. Since the algorithm solves the motion planning problem iteratively, the algorithm can handle dynamic environments directly. However, the performance of the algorithm is influenced by the hyper-parameters dramatically, especially the mean value of the control input sample distribution. Intuitively, a small mean value may result in conservative exploration, and a large mean value may result in risky behaviors. In particular, in dynamic environments, to get better performance, a time-varying mean value is needed. Thus, in this work, we utilize the RRT algorithm to design a better sample mean to guide the MPPI algorithm in exploring the workspace and sampling the trajectories.

The idea of using the RRT and RRT* algorithms to solve motion planning problems in dynamic environments is inspired by [16]. In this work, the authors propose the RRT^X algorithm, which combines the replanning ideas provided in the DRRT algorithm and RRT* algorithm to continuously update the path during the exploration when the environment

This research is supported by NSF CPS #1932529, AFOSR #FA9550-21-1-0411, NASA #80NSSC22M0070 and #80NSSC20M0229 awards.

†Chuyuan Tao and Naira Hovakimyan are with the Department of Mechanical Science and Engineering, University of Illinois at Urbana-Champaign, USA. {chuyuan2, nhovakim}@illinois.edu

‡Hunmin Kim is with the Department of Electrical and Computer Engineering, Mercer University, USA. kim_h@mercer.edu

changes. However, the algorithms require large computation power and are hard to implement on the robots in real time. The idea of using a nominal or baseline controller to improve the performance of the MPPI algorithm is inspired by [17], [18]. In [17], the authors present a method using the entire planning tree from the RRT* algorithm to approximate the value functions in the MPPI algorithm. However, due to the learning procedure in the algorithm, the cost of finding an optimal solution to the motion planning problem is computationally expensive. In [18], the authors control the variance of the MPPI algorithm to handle the dynamic environments and provide a faster running time and better collision avoidance in a ground unicycle simulation. However, the algorithm requires the linearized dynamic model, which results in expensive computation for complex or high-dimensional systems.

Contributions. This work presents a sampling-based algorithm to solve the motion planning problem. We use the RRT algorithm to provide a nominal mean value for the random control distribution of the MPPI algorithm. The proposed algorithm advances the RRT algorithm in terms of its applicability to navigation in dynamic environments and optimality. With respect to the MPPI algorithm, it reduces the need to fine-tune the mean value. We provide simulation results to discuss the importance of tuning the mean value in the original MPPI algorithm. In particular, the sampling-based algorithms need sufficiently large means to explore the state space and sufficiently small means to guarantee that the samples reconstruct the optimal control. Thus, our proposed method avoids fine-tuning the mean value by using the nominal path provided by the RRT algorithms. Our algorithm finds the optimal solutions by allowing the MPPI algorithm to explore freely, and it has a better performance in running time by using the offline RRT algorithm to provide a nominal path to guide the MPPI algorithm. If the MPPI algorithm reaches the area where the nominal path provided by the offline RRT algorithm has not been explored before, our algorithm uses an online real-time Replanning RRT algorithm to provide new nominal paths. Finally, we implement our algorithm on a unicycle robot and solve the motion planning problem in static and dynamic environments.

II. Problem Statement

Let $\mathcal{X}^d$ denote the d dimensional space and $\mathcal{X}_{obs} \subset \mathcal{X}^d$ be the obstacle space. We define the free space $\mathcal{X}_f = \mathcal{X}^d \setminus \mathcal{X}_{obs}$, where the robot can reach. The start and goal states are x_s and x_g, respectively. We assume that the robot has a nonlinear control affine dynamical system:

$$dx_t = (f(x_t) + g(x_t)u_t)dt + \sigma(x_t)dW_t, \tag{1}$$

where $x_t \in \mathbb{R}^n$ is the state, $f : \mathbb{R}^n \to \mathbb{R}^n$, $g : \mathbb{R}^n \to \mathbb{R}^{n\times m}$ and $\sigma : \mathbb{R}^n \to \mathbb{R}^n$ are locally Lipschitz continuous functions, and dW_t is a Wiener process with $\langle dW_k dW_l \rangle = \nu_{kl}(x_t, u_t, t)dt$.

Problem 1: Given $\mathcal{X}, \mathcal{X}_{obs}, x_g$, and $x_t(0) = x_s$, we aim to find the optimal control input u^* that would lead to the shortest path to state x_g in the static or dynamic environments (i.e., time-varying $\mathcal{X}_{obs}$). Specifically, we aim to minimize the cost function $S(x_t, u_t)$, defined as:

$$S(x_t, u_t) = \phi(x_{t+T}) + \sum_{j=t}^{t+T-1} q(x_j, u_j) \tag{2}$$

subject to $x_j \in \mathcal{X}_f$, where $q(x_j, u_j)$ is a running cost function, and $\phi(x_{t+T})$ is the terminal cost function.

III. Approach

To this end, we propose the RRT-guided MPPI algorithm in Section III-C, which addresses Problem 1. By generating mean values using the RRT (presented in Section III-A), the MPPI algorithm (presented in Section III-B) does not need fine parameter tuning in dynamic environments. Furthermore, the MPPI algorithm provides real-time implementable optimal control inputs.

Algorithm 1 RRT algorithm

Given: Initial vertices $\mathcal{S} \leftarrow \{s_s\}$
Given: Initial edges $\varepsilon \leftarrow \varnothing$
for $i = 1, ..., n$ **do**
  $s_{sample} \leftarrow$*SampleState*();
  $s_{near} \leftarrow$*NearestNeighbor*$(\mathcal{S}, s_{sample})$;
  $s \leftarrow$*Steer*$(s_{near}, s_{sample}, \gamma)$;
  if *ObstacleFree*(s_{near}, s) **then**
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{s\}$;
    $\varepsilon \leftarrow \varepsilon \cup \{(s_{near}, s)\}$;
  end if
  if $d(s, s_g) < \gamma$ **then**
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{s_g\}$;
    $\varepsilon \leftarrow \varepsilon \cup \{(s, s_g)\}$;
    return $p =$ *ExtractPath* $(\mathcal{S}, \varepsilon)$
  end if
end for

A. Rapidly-Exploring Random Trees Algorithm

We present the RRT algorithm in Algorithm 1. The algorithm uses function *SampleState*() to uniformly sample a new state s_{sample} in the configuration space $\mathcal{X}$. Then, the algorithm finds the nearest vertex s_{near} with the function *NearestNeighbor*() and projects the sample state s_{sample} to the ball with radius γ with a function *Steer*(). If the new edge between the s_{near} and s is free from collision, the projected state s will be added to the vertex set, and the new edge $(s, s_{nearest})$ will be added to the vertex set. If the new vertex s is within a radius γ of the goal state s_{goal}, then the RRT reaches the target and returns the path p. Otherwise, the algorithm adds the new vertex and advances the exploration.

The RRT algorithm focuses on fast iteration while not guaranteeing to find the optimal solution to the motion planning problem. RRT* [9] is the first variant of the RRT algorithm that could ensure asymptotic optimality. By allowing the new vertices to "rewire" graph edges within the local neighborhood, the algorithm guarantees asymptotic optimality with the cost of increasing running time. However,

the running time of the RRT* algorithm also increases dramatically, and it is hard to implement the algorithm in real time. In this work, instead of using the RRT* algorithm, we utilize the MPPI algorithm to find the optimal solution to the motion planning problem.

B. Model Predictive Path Integral Control Algorithm

In this section, we introduce the MPPI algorithm [14], [15] to solve the motion planning problem. First, we need to sample K trajectories with time horizon T with random control input $u_{i,j} \sim \mathcal{N}(\mu, \Sigma)$, where $i = 1, \cdots, K$ is the sample trajectory index. In each trajectory τ_i, $u_i = [u_{i,t}, \ldots, u_{i,t+T-1}]^T$ denotes the actual control input sequence, and $[x_{i,t+1}, \ldots, x_{i,t+T}]^T$ denotes the states of the current sample trajectory. The evaluated cost for i^{th} trajectory is given by

$$S(\tau_i) = \phi(x_{i,t+T}) + \sum_{j=t}^{t+T-1} q(x_{i,j}, u_{i,j}), \tag{3}$$

where $q(x_{i,j}, u_{i,j}) = (x_g - x_{i,j})^T (x_g - x_{i,j}) + \frac{1}{2} u_{i,j}^T R u_{i,j}$, where R is a positive definite control penalty matrix. We define the weight of i^{th} trajectory ω_i as:

$$\omega_i = \exp(-\frac{1}{\lambda}(S_i)),$$

where λ is the parameter that decides how much we trust the better-performed trajectories. Then the MPPI algorithm updates the control input using the following equation

$$u_j = \frac{\sum_{i=1}^{K} \omega_i u_{i,j}}{\sum_{i=1}^{K} \omega_i} \tag{4}$$

for $j = t, \cdots, t+T-1$, which approximates the optimal control inputs using sampled trajectories.

In conclusion, the MPPI algorithm uses sample trajectories to find the optimal control input to solve the motion planning problem. Because the MPPI algorithm avoids calculating the derivatives of the nonlinear dynamic systems or the value functions, it can be implemented in real-time with the help of parallel computations on the GPUs, even for complex dynamic systems.

While the MPPI algorithm has clear merits mentioned in the previous paragraph, its performance is dramatically influenced by the mean of the control input distribution. In the unicycle simulations presented in Figure 1, the MPPI algorithm may fail to solve the motion planning problems due to the bad choice of the mean value. In Figure 1, the red path is the result when the mean value of the control input is $\mu = [1, 0]^T$; the yellow path is the result when the mean value is $\mu = [1, 1]^T$; and the green path is the result when the mean value is $\mu = [0, 0]^T$. We can find out easily that a smaller mean value hinders the exploration and cannot finish the path planning task in the provided horizon. A larger mean value may result in a safety violation. If the mean value is large, the MPPI algorithm is more aggressive and finishes the task faster. However, it also provides more risky control inputs and should require a larger sample size to get an optimal solution.

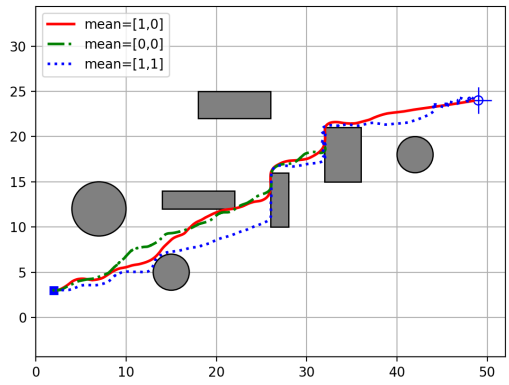


Fig. 1: MPPI algorithm with various mean values in a static environment.

An example of MPPI trajectory in a dynamic environment is shown in Figure 2, where we increase the radius of circle obstacles by 2 and 4. The mean value for the MPPI is $[1, 0]^T$, which has a perfect performance in a static environment but fails the task, being stuck between obstacles in dynamic environments, as the second figure shown in Figure 2. Thus, we can conclude that for a dynamic environment, the MPPI algorithm needs a time-varying mean value to obtain a fine performance. To automate the procedure of choosing dynamic mean values, we combine it with the RRT algorithm.

C. Replanning RRT Guided MPPI Algorithm

We propose a new sampling-based method that utilizes the RRT algorithm to guide the MPPI algorithm to solve the optimal motion planning problem defined in Problem 1. Our algorithm performs well without tuning the mean value of the control distribution. Our algorithm also has a fast running speed, and thus it can be implemented in real-time.

First, we use the RRT algorithm to provide an offline nominal path p_n, which provides a possible solution to solve the motion planning problem. Although the RRT algorithm has a relatively fast iteration speed, the algorithm is still hard to implement in real time, which will also be shown later in the simulations. So we first run the RRT algorithm offline, not in real-time. Since the RRT algorithm only provides the state information instead of the control information, we then use Lyapunov controllers or PD controllers to obtain a nominal control input u_n in real-time. However, since the RRT algorithm cannot guarantee the optimal solution, we use the MPPI algorithm with nominal control input u_n as the mean of the random control distribution to explore an optimal control input u^* at each time step.

As the difference between the nominal control input u_n and the optimal control input u^* becomes increasingly large, the optimal control input may lead the agents to reach the area where the path p_n of the RRT algorithm never reached before. In this case, the nominal path may have a negative influence on the MPPI algorithm. To address this issue, we use

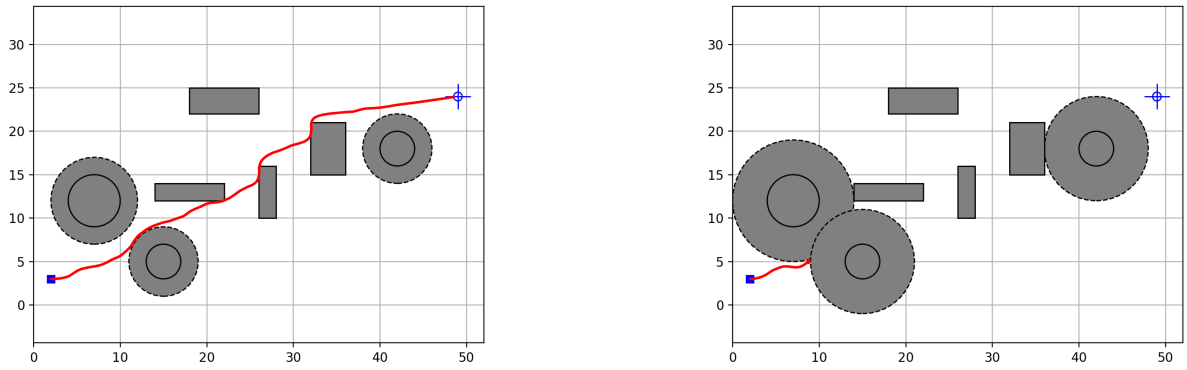


Fig. 2: Results with MPPI algorithm in dynamic environments. In the first environment, the radius of the circle obstacles increases by two at time $t = 0.5s$. The solid line represents the radius before the environment changes, and the dotted line represents the radius after the environment changes. The MPPI algorithm can still handle the change without changing the mean value. In the second environment, the radius increases by four at time $t = 0.5s$ as well. The MPPI algorithm with a fixed mean value fails to finish the task.

the replanning idea first presented in the DRRT algorithm, where the agents replan under the changing environment. However, unlike the DRRT algorithm, our algorithm replans when the nominal controllers u_n are no longer helpful. In our implementation, we use a distance R to justify if the replanning is needed. We use a *NearestNeighbor*() function to calculate the distance between the current state x and its closest point x_n in the nominal path, and if the distance is larger than R, our algorithm replans.

Algorithm 2 Replanning RRT algorithm

Given: Vertices $\mathcal{S} \leftarrow [p]$, $\mathcal{S}' \leftarrow [s_s]$
Given: Edges $\varepsilon \leftarrow \varnothing$
for $i = 1, ..., n$ **do**
  $s_{sample} \leftarrow$*SampleState*();
  $s_{near} \leftarrow$*NearestNeighbor*$(\mathcal{S}, s_{sample})$;
  $s'_{near} \leftarrow$*NearestNeighbor*$(\mathcal{S}', s_{sample})$;
  $s \leftarrow$**Steer**$(s_{sample}, s'_{near}, \gamma)$
  if *ObstacleFree*(s'_{near}, s) **then**
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{s\}$;
    $\varepsilon \leftarrow \varepsilon \cup \{(s_{near}, s)\}$;
  end if
  if $d(s, s_g) < \gamma$ or $d(s, s_{near}) < \gamma$ **then**
    $\mathcal{S}' \leftarrow \mathcal{S}' \cup \{s_g\}$ or $\mathcal{S}' \leftarrow \mathcal{S}' \cup \{s_{near}\}$;
    $\varepsilon \leftarrow \varepsilon \cup \{(s, s_g)\}$ or $\varepsilon \leftarrow \varepsilon \cup \{(s, s_{near})\}$;
    return $p =$ *ExtractPath* $(\mathcal{S}', \varepsilon)$
  end if
end for

The detail of the Replanning RRT algorithm is presented in Algorithm 2. Our replanning RRT algorithm has a different input compared to the previous RRT algorithm, where the vertices set $\mathcal{S}$ is given by the previous nominal path p_n, and vertices set $\mathcal{S}'$ contains the start state s_s. Next, we sample the state s_{sample} and find the nearest neighbor s'_{near} and project the states s the same way as the RRT algorithm in Algorithm 1.

Algorithm 3 RRT-MPPI algorithm

Given: Number of sample trajectories K and timesteps T;
Given: Initial variance Σ_0;
Given: Cost function parameters ϕ, q, R, λ;
Use offline RRT algorithm to get initial path p_n
while task is not completed **do**
  for $j \leftarrow t$ to $t + T - 1$ **do**
    Find the nearest state $s \in p_n$ to the current state x;
    if $d(s, x) \geq R$ **then**
      Use Replanning RRT algorithm to get new p_n
      Find new nearest state $s \in p_n$
    end if
    Get nominal control mean value $u_n = L(s, x)$
    for $i \leftarrow 0$ to $K - 1$ **do**
      Generate control variations $u_{i,j} \sim \mathcal{N}(u_n, \Sigma_0)$;
      Simulate discrete dynamic to obtain $x_{i,j}$;
      Calculate cost function $S(\tau_i)$ += $q(x_{i,j}, u_{i,j})$;
    end for
    Calculate the terminal cost $S(\tau_i)$ += $\phi(x_{i,t+T})$
  end for
  $\beta \leftarrow \min_i[S(\tau_i)]$;
  Get sample weights ω_i;
  Update control input using $\omega_{i,j}$ and $u_{i,j}$;
  Send u_t to actuator;
end while

However, we will also find the closest state s_{near} to the vertices set $\mathcal{S}'$. Then we check if the new edges (s'_{near}, s) are in the collision-free space $\mathcal{X}_f$. Finally, we repeat the previous procedure until the distance between new vertex s and target state s_g or closest state on the nominal path s_{near} is smaller than the radius γ and return the new path p. Since our replanning algorithm uses the MPPI algorithm to give a penalty to the obstacles at each time step, we do not need to

trim the previous result. As a result, the proposed algorithm is significantly faster than the original RRT algorithm. The first part of our algorithm requires an offline RRT algorithm to provide a nominal path p_n. The second part of our algorithm, which is based on the Replanning RRT algorithm and MPPI algorithm, can be implemented on the robots in real-time.

With the new nominal path $\mathcal{S}'$, we then use the Lyapunov controllers or the PD controllers to get the new nominal control input u'_n. We can obtain a sampled trajectory $\tau_i = [x_{i,t}, ..., x_{i,t+T-1}]^T$ with the new distribution $\mathcal{N}(u'_n, \Sigma)$, where T is the time horizon of the MPPI algorithm and Σ is the fixed variance. Then we calculate the cost of the i^{th} sampled trajectory by using the quadratic cost function $S(\cdot)$ and using the following equation, and we calculate the weight of each trajectory:

$$\omega_i = \exp\left(-\frac{1}{\lambda}S(\tau_i) - \min(S(\tau_i))\right). \tag{5}$$

Note that we need to find the minimum value of all trajectories to prevent the numerical instability of the algorithms [18]. Finally, we use the normalized trajectory weights to calculate the control update law: for $j = t, \cdots, t+T-1$,

$$u_j = \frac{\sum_{i=1}^{K} \omega_i u_{i,j}}{\sum_{i=1}^{K} \omega_i}. \tag{6}$$

The proposed RRT-guided MPPI algorithm is summarized in Algorithm 3.

IV. Simulations

A. Unicycle Dynamics

We implement our algorithm on a two-dimensional unicycle dynamic system with:

$$\begin{bmatrix} \dot{x}^d \\ \dot{y}^d \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ \frac{\tan\phi}{L} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v + \delta^v \\ \omega + \delta^\omega \end{bmatrix},$$

where x, y are the coordinates, θ is the heading angle, and ϕ is the steering angle. v is the linear velocity control input, and ω is the angular velocity control input. $L = 0.5$ is the length of the wheelbase. $\delta = [\delta^v, \delta^\omega] \sim \mathcal{N}(\bar{0}, I)$ is the random control input perturbation. The time step for the discrete-time simulation is $\Delta t = 0.05s$. We use the following discrete dynamics in the MPPI algorithm:

$$\begin{bmatrix} x^d_{t+1} \\ y^d_{t+1} \\ \theta_{t+1} \\ \phi_{t+1} \end{bmatrix} = \begin{bmatrix} x^d_t \\ y^d_t \\ \theta_t \\ \phi_t \end{bmatrix} + \Delta t \begin{bmatrix} \cos\theta_t & 0 \\ \sin\theta_t & 0 \\ \frac{\tan\phi_t}{L} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_t + \delta^v_t \\ \omega_t + \delta^\omega_t \end{bmatrix}.$$

B. Simulation Setups

The maximum sample size in the RRT algorithm is set to 20000, and the projection radius is set to $\gamma = 0.5$. We set the sample size for the MPPI algorithm to be $K = 10000$, the time horizon to be 20, and $\lambda = 1.0$. The cost function is defined as:

$$q(x) = \|x - x_g\|_2^2 + 1000 * \mathbb{1}_{x \in \mathcal{X}_{obs}},$$

where x represents the current states, and x_g represents the goal state. $\mathcal{X}_{obs}$ is the obstacle set over $\mathbb{R}^2$, and $\mathbb{1}$ is the indicator function. We test our algorithm in two different environments, a static environment and a dynamic environment. In both simulations, the start states are $x_s = [2, 3, 0, 0]^T$, and the goal states are $x_g = [49, 24, 0, 0]^T$. We use a Lyapunov controller to design the velocity control input and a Proportional controller to design the angular velocity control input:

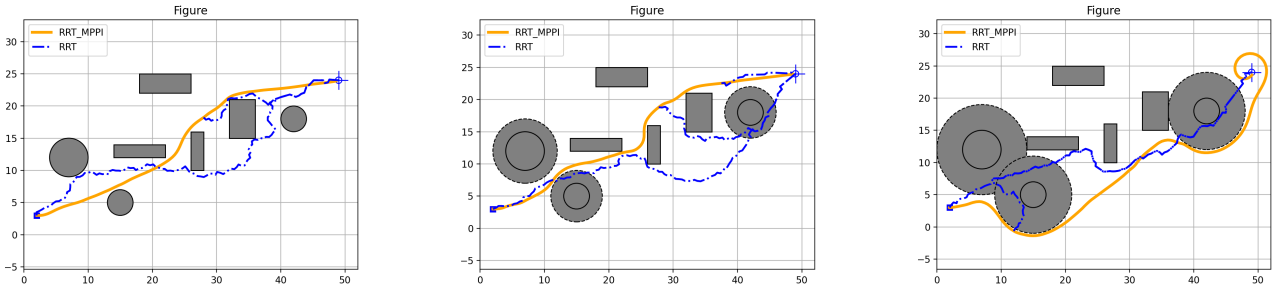
$$\begin{aligned} u_v &= e_d v_{max} \frac{(1 - \exp(-\alpha \|e_d\|^2))}{\|e_d\|}, \\ u_\omega &= k_p e_\theta, \end{aligned} \tag{7}$$

where e_d, e_θ are the error between the desired target state and current states.

C. Results

We first test our algorithm in a fully known static environment with the replanning conditions $R = 6$. Figure 3a shows the result of a unicycle robot navigating through the obstacles. The black rectangles represent the boundary of the environments, the grey circles and rectangles denote the obstacles, the blue square denotes the start state x_s, and the blue cross denotes the goal state x_g. The blue line in the figures is the result of the replanning RRT path, and the orange line in the figures is the resulting control output from the RRT-MPPI algorithm. Next, we implement our algorithm in dynamic environments where the radius of the circle obstacles increases by 2 and 4. We plot the environment changes by plotting the circles with dot lines as their boundaries. Figure 3b and Figure 3c show that our algorithm can handle dynamic environments. Note that in dynamic environments, the nominal path provided by the RRT path may violate safety. However, since the MPPI algorithm can explore freely, our algorithm is still able to find the solution to the optimal motion planning problem. Besides, we want to implement the algorithm in real time, so the RRT algorithm we adopt here is relatively inaccurate and can only guide the MPPI algorithm.

We repeat the previous experiments for 10 times and change the value of the replanning condition from $R = 2$ to $R = 8$. In Figure 4, we plot the average time, the maximum and minimum running time of our algorithm, and the original MPPI algorithm with mean $[1, 0]^T$ in static and dynamic environments. The time of the offline RRT algorithm is in purple color. Note that even the offline RRT algorithm takes around 0.2 seconds, it is still not fast enough to be implemented in real-time. The online RRT-MPPI algorithm for the static environment is in blue color, and the dynamic environment is in yellow color. We also compare the running time with the MPPI algorithm with a fixed mean value $[\mu_v, \mu_\omega]^T = [1, 0]^T$, which is the grey color in the figures. As the radius decreases, the RRT-MPPI algorithm can provide a more accurate nominal controller. However, the time of the replanning procedure increases as well, and as a result, the total time to complete the task becomes longer. We can see



(a) Static Environment (b) 1st Dynamic Environment (c) 2nd Dynamic Environment

Fig. 3: Results with RRT-MPPI algorithm in static or dynamic environments. The blue dash-dot lines are the paths provided by the Replanning RRT algorithm, and the orange line is the result of our proposed method.

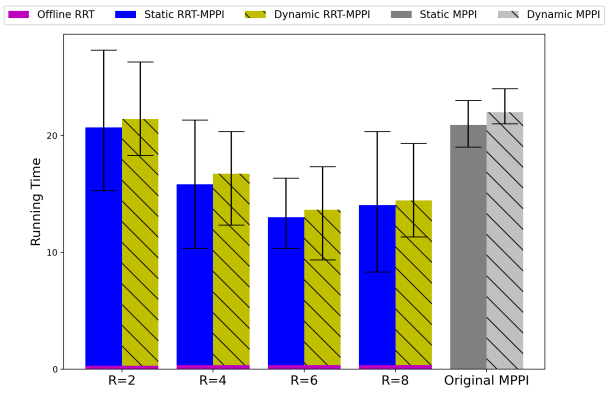


Fig. 4: Running time of RRT-MPPI algorithm with different replanning conditions and MPPI algorithm with the fixed mean value.

that when the radius $R = 6$, and the algorithm takes the least time to finish the motion planning task.

V. Conclusion

This paper presents a real-time RRT-MPPI algorithm to solve the motion planning problem in different environments. The proposed algorithm advances the RRT algorithm in terms of dynamic environment navigation and optimality and reduces the need to fine-tune the mean value of the MPPI algorithm. In particular, we use the RRT algorithm to provide the suitable nominal control mean value for the random distribution in the MPPI algorithm. This helps us avoid fine-tuning the mean value and balance the optimality and exploration. Finally, in the simulations, we use a unicycle robot to implement the algorithm in static and dynamic environments. We compare the running time of our RRT-MPPI algorithm with the fixed value MPPI algorithm in the experiments, showing that our algorithm is faster.

References

[1] F. Gao, L. Wang, B. Zhou, X. Zhou, J. Pan, and S. Shen, "Teach-repeat-replan: A complete and robust system for aggressive flight in complex environments," *IEEE Transactions on Robotics*, vol. 36, no. 5, pp. 1526–1545, 2020.

[2] B. Zhou, J. Pan, F. Gao, and S. Shen, "Raptor: Robust and perception-aware trajectory replanning for quadrotor fast flight," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1992–2009, 2021.

[3] Q. Nguyen, X. Da, J. Grizzle, and K. Sreenath, "Dynamic walking on stepping stones with gait library and control barrier functions," in *Algorithmic Foundations of Robotics XII*, pp. 384–399, Springer, 2020.

[4] Y. Cheng, P. Zhao, and N. Hovakimyan, "Safe model-free reinforcement learning using disturbance-observer-based control barrier functions," *arXiv preprint arXiv:2211.17250*, 2022.

[5] C. Tao, H.-J. Yoon, H. Kim, N. Hovakimyan, and P. Voulgaris, "Path integral methods with stochastic control barrier functions," in *2022 IEEE 61st Conference on Decision and Control (CDC)*, pp. 1654–1659, 2022.

[6] 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.

[7] 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.

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

[9] 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.

[10] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with rrts," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pp. 1243–1248, IEEE, 2006.

[11] M. Zucker, J. Kuffner, and M. Branicky, "Multipartite rrts for rapid replanning in dynamic environments," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 1603–1609, IEEE, 2007.

[12] G. Williams, A. Aldrich, and E. Theodorou, "Model predictive path integral control using covariance variable importance sampling," *arXiv preprint arXiv:1509.01149*, 2015.

[13] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1433–1440, IEEE, 2016.

[14] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, "Information theoretic mpc for model-based reinforcement learning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1714–1721, IEEE, 2017.

[15] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Information-theoretic model predictive control: Theory and applications to autonomous driving," *IEEE Transactions on Robotics*, vol. 34, no. 6, pp. 1603–1622, 2018.

[16] M. Otte and E. Frazzoli, "Rrtx: Asymptotically optimal single-query sampling-based motion planning with quick replanning," *The International Journal of Robotics Research*, vol. 35, no. 7, pp. 797–822, 2016.

[17] N. Hatch and B. Boots, "The value of planning for infinite-horizon model predictive control," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7372–7378, IEEE, 2021.

[18] J. Yin, Z. Zhang, E. Theodorou, and P. Tsiotras, "Trajectory distribution control for model predictive path integral control using covariance steering," in *2022 International Conference on Robotics and Automation (ICRA)*, pp. 1478–1484, IEEE, 2022.