Visual-RRT: Finding Paths toward Visual-Goals via Differentiable Rendering

Supplementary Material

In this supplementary material, we provide additional technical details and experimental results to complement the main paper. Sec. A describes the background of RRT planners. Sec. B presents comprehensive ablation studies analyzing frontier node sampling strategies and inertial gradient tree expansion schemes. Sec. C extends experimental validation with path quality analysis, real-world Fetch deployment, planning with generated goal images, tree structure visualization, and visual ambiguity analysis. Sec. D details implementation including dataset construction, and hyperparameter specifications.

Video Results: The video results are available at https://sgvr.kaist.ac.kr/Visual-RRT.

A. Background of RRT

Rapidly-exploring Random Trees (RRT) is a sampling-based motion planning algorithm that incrementally constructs a search tree to find motion paths in the robot's C-space. The algorithm operates through the following iterative process.

1. *Random Sampling*: Sample a random configuration q_{rand} from the C-space.
2. *Nearest Node Selection*: Find the nearest node q_{near} in the current tree to q_{rand}.
3. *Tree Extension*: Extend from q_{near} toward q_{rand} by a fixed step size ϵ to generate a new configuration q_{new}.
4. *Feasibility Check*: If q_{new} is feasible (*e.g.*, collision or joint-limit violations), add it to the tree as a child of q_{near}.
5. *Termination*: Repeat until a node reaches the goal region.

Goal-biased Exploitation. To balance exploration and exploitation, RRT typically employs goal biasing: with a certain probability, q_{rand} is set to the goal configuration q_{goal} instead of being sampled uniformly. This biasing steers tree growth toward the goal region, significantly accelerating convergence. However, this strategy fundamentally relies on the availability of an explicit goal configuration q_{goal}, typically specified as numerical joint angles. In visual-goal planning scenarios where goals are provided as images rather than configurations, this dependency poses a key challenge, as there is no explicit q_{goal} to bias toward.

B. Extended Ablation Studies

B.1. Frontier-based Exploration-Exploitation

Truncated Geometric Distribution. In Tab. S1, we analyze the effectiveness of different frontier node selection strategies on visual planning performance. In our method, the truncated geometric distribution parameter $\kappa \in [0, 1)$ controls the frontier selection bias: smaller κ focuses on sampling visually promising nodes. Rows (a-c) show that average success rates improve with higher κ. In particular, at 2.5 rad, $\kappa = 0.9$ maintains higher success rates compared with $\kappa = 0.5$. These results indicate that stochastic diversity in frontier selection helps navigate distant-goal scenarios where gradient-based exploitation is prone to local minima.

Table S1. **Ablation on frontier sampling strategies.** Success rates (%) across C-space distance bins for frontier node selection methods. Higher κ achieves better performance through stochastic diversity. Uniform sampling (d) and Top-K (e) show limitations compared to $\kappa = 0.9$ at larger distances (2.0 and 2.5 rad).

Frontier Sampling	Target distance bins (rad) 0.5	1.0	1.5	2.0	2.5	Avg.
(a) Trunc. geometric (κ=0.9)	**87.3**	87.5	**85.2**	**76.2**	**62.7**	**79.8**
(b) Trunc. geometric (κ=0.7)	86.7	**88.3**	83.4	74.0	55.0	77.5
(c) Trunc. geometric (κ=0.5)	86.3	85.7	82.6	72.5	54.2	76.2
(d) Uniform	17.3	3.5	1.3	0.3	0.8	4.7
(e) Top-K	86.8	86.7	82.7	71.3	53.2	76.1

Alternative Frontier Selection. As shown in Tab. S1, we evaluate uniform sampling (d) and Top-K selection (e) as alternative frontier node selection methods for comparison. Uniform sampling selects nodes without considering visual loss, while Top-K deterministically chooses the K best nodes. The uniform sampling strategy demonstrates limited effectiveness, as it allocates computational effort across all nodes without prioritizing visually promising regions. Top-K selection shows reasonable planning results comparable to $\kappa = 0.5$. This similarity reflects their shared behavior where both heavily concentrate selection on visually promising nodes. Accordingly, they exhibit similar challenges at larger distances (2.0 and 2.5 rad), compared to $\kappa = 0.9$.

B.2. Inertial Gradient Tree Expansion

In the main paper, we instantiate inertial gradient tree expansion with Adam optimizer states (*i.e.*, first- and second-moment estimates with iteration steps). The first-moment estimate in Adam plays the role of momentum along each branch of the tree, which motivates the term "inertial" in our formulation. In this supplementary material, we further compare several alternative optimization states within the same state-inheritance framework.

Momentum in Tree Expansion. We examine the effectiveness of inertia using a simpler momentum formulation. In this variant, each node stores a velocity vector u_p that is inherited from parent to child and updated with Polyak momentum [11]. Given a parent configuration q_p and its velocity

Table S2. **Ablation on optimizer state inheritance.** Success rates (%) across C-space distances for different inherited optimizer states in vRRT. Naive gradient descent without any optimizer state (a) is less effective at exploiting visual gradients. Inheriting a simple momentum term (b)–(g) substantially improves performance over stateless tree expansion. Adaptive optimizers that accumulate gradient statistics (h)–(k) further outperform the best momentum-only baseline, with Adam achieving the highest average success rate.

Inherited Optimizer State	Target distance bins (rad) 0.5	1.0	1.5	2.0	2.5	Avg.
(a) Naive GD (Eq. 1)	21.8	3.8	2.2	4.7	3.5	7.2
(b) Momentum ($\mu = 0.5$)	37.8	16.3	10.1	12.8	10.8	17.6
(c) Momentum ($\mu = 0.7$)	54.0	28.6	21.3	22.2	16.7	28.5
(d) Momentum ($\mu = 0.8$)	68.8	38.6	32.9	26.7	15.3	36.5
(e) Momentum ($\mu = 0.85$)	73.2	49.4	38.7	31.5	22.0	43.0
(f) Momentum ($\mu = 0.9$)	73.8	60.4	46.3	37.7	23.5	48.3
(g) Momentum ($\mu = 0.95$)	72.3	67.5	57.1	43.7	32.2	54.6
(h) Adaptive gradient [3]	83.3	80.4	71.9	59.2	43.5	67.7
(i) RMSProp [12]	89.5	80.2	70.6	53.8	42.7	67.4
(j) Lion [1]	87.8	86.3	81.6	64.3	47.3	73.5
(k) Adam [5] (Eq. 5)	87.3	87.5	85.2	76.2	62.7	79.8

u_p, the child inherits velocity u_{new} and configuration q_{new} as:

$$u_{\text{new}} = \mu\, u_p + (1-\mu)\, \nabla_q \mathcal{L}_{\text{render}}(q_p), \tag{S1}$$

$$q_{\text{new}} = q_p - \alpha\, u_{\text{new}}, \tag{S2}$$

where $\mu \in [0, 1)$ is the momentum coefficient and α is the step size. In Tab. S2, row (a) shows that naive gradient descent (Naive GD) fails to leverage visual gradients effectively without optimization inertia. Meanwhile, rows (b)–(g) show that inheriting this simple momentum term along the tree improves visual motion planning performance over stateless gradient descent, demonstrating the importance of propagating optimizer states across tree expansions in our framework.

Alternative Optimization State Inheritance. Our framework only requires each node to maintain a compact summary of its gradient history during tree expansion, which can be realized through various optimizer designs. Thus, we further evaluate several history-aware optimization variants (Adaptive gradient [3], RMSProp [12], Lion [1], Adam [5]) by plugging them into the same state-inheritance mechanism. In particular, Adaptive gradient and RMSProp accumulate per-parameter squared gradients to adapt the effective step size, while Lion and Adam additionally maintain directional statistics over past gradients. As shown in Tab. S2 (h)–(k), these history-aware instantiations substantially outperform naive gradient descent (Naive GD), indicating that propagating optimizer states along the tree is more important than the exact choice of optimizer. Among these variants, Adam achieves the highest average success rate, supporting our default design choice in the main paper.

C. Extended Motion Planning Evaluation

We provide comprehensive analysis of vRRT's visual-goal motion planning performance. We first validate path quality and real-world deployment, then analyze our tree structure and computational efficiency. Finally, we discuss inherent limitations arising from visual ambiguity.

Table S3. **Comparing RRT* path lengths across recovered goals.** RRT* paths to vRRT-recovered goals are consistently longer, suggesting the methods solve different problem subsets: vRRT addresses more challenging instances requiring longer paths, while Dr.Robot [8] succeeds on shorter-distance cases.

Robot	RRT*	Target distance bins (rad) 0.5	1.0	1.5	2.0	2.5	Avg.
Franka	w/ Dr.Robot goal	0.50	0.99	1.40	1.90	2.49	1.46
	w/ vRRT goal	0.50	1.00	1.55	2.08	2.63	**1.55**
UR5e	w/ Dr.Robot goal	0.50	1.02	1.52	2.00	2.55	1.52
	w/ vRRT goal	0.50	1.01	1.55	2.10	2.59	**1.55**
Fetch	w/ Dr.Robot goal	0.47	0.98	1.49	1.93	2.46	1.47
	w/ vRRT goal	0.49	1.00	1.52	2.07	2.57	**1.53**

Table S4. **Real-world Fetch experiments.** Success counts across three scenes with 25 tasks each. vRRT achieves 80% success rate (60/75) in real-world deployment, demonstrating potential for sim-to-real transfer in visual-goal motion planning.

	Scene 1	Scene 2	Scene 3	Total
Tasks	25	25	25	75
Successes	18	24	18	60

C.1. Path Quality Analysis

In Tab. 1 of the main paper, vRRT achieves higher success rates than the baselines, but yields moderately longer paths than Dr.Robot + RRT*. To determine whether this difference reflects planning inefficiency or differences in problem coverage, we analyze the recovered goal configurations of both methods by running RRT* from the same start configuration to each recovered goal. Tab. S3 shows that goals recovered by vRRT consistently induce longer RRT* paths than those recovered by Dr.Robot. This suggests that the two methods succeed on different subsets of problems: vRRT solves more challenging instances that inherently require longer paths, whereas Dr.Robot tends to succeed on easier ones. Therefore, the longer paths reported for vRRT in Tab. 1 are better explained by broader problem coverage than by planning inefficiency. Fig. S1 presents representative trajectories from vRRT and RRT* across 18 scenes. Despite relying only on visual objectives without explicit goal configurations, vRRT consistently produces paths that closely match the geometric structure of the RRT* reference solutions. This further supports that our frontier-based exploration–exploitation strategy can recover configuration-space-efficient paths from visual goals.

C.2. Real-world Validation

We deploy vRRT on a physical Fetch robot across three scenes with varying obstacle configurations: Scene 1 with one obstacle and Scenes 2-3 with two obstacles each. For

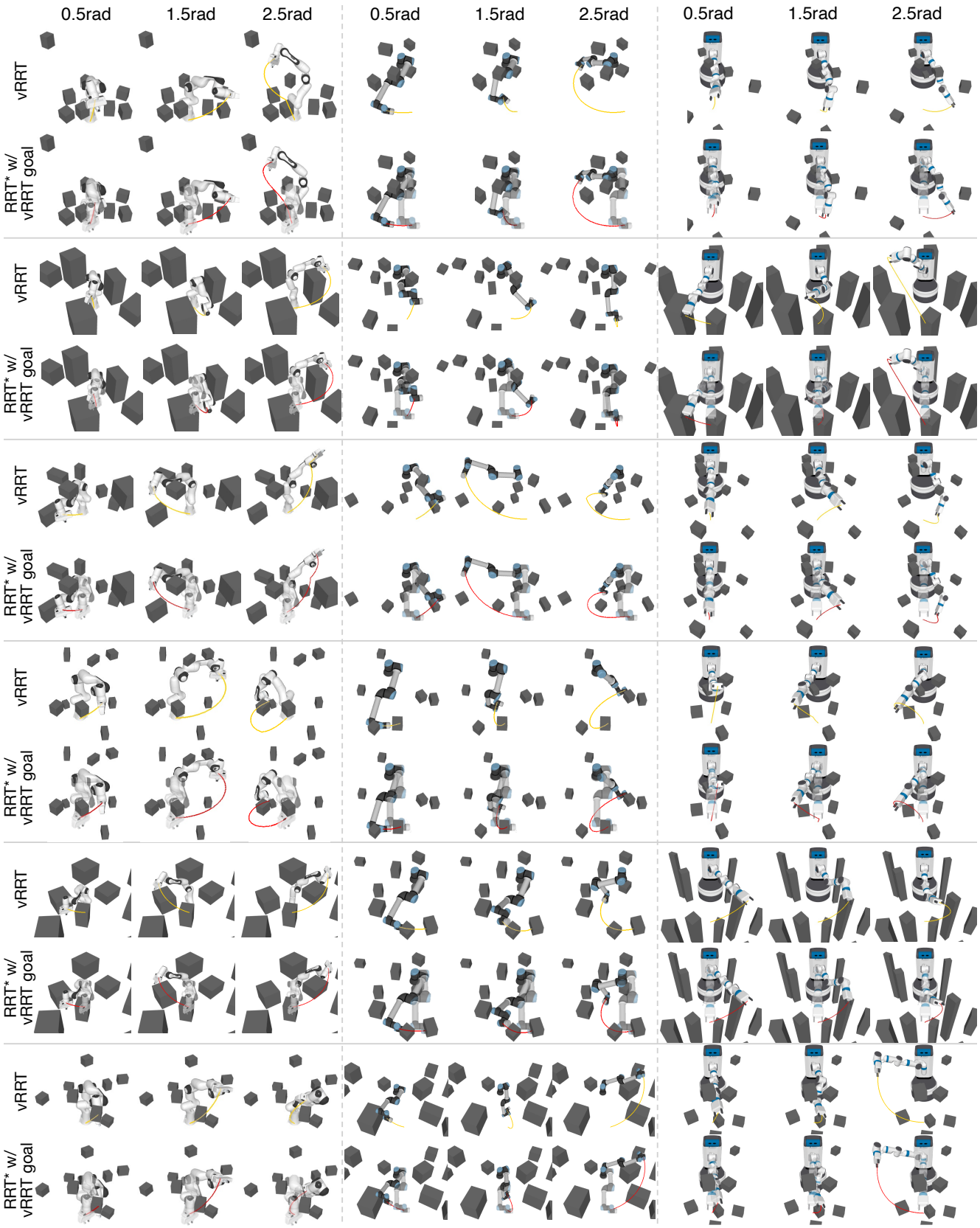

Figure S1. **Path comparison between vRRT and RRT*.** Paths from vRRT (yellow) and RRT* (red) across three robots in six scenes each. vRRT produces paths similar to RRT* despite using only visual objectives. Videos are provided in the project page.

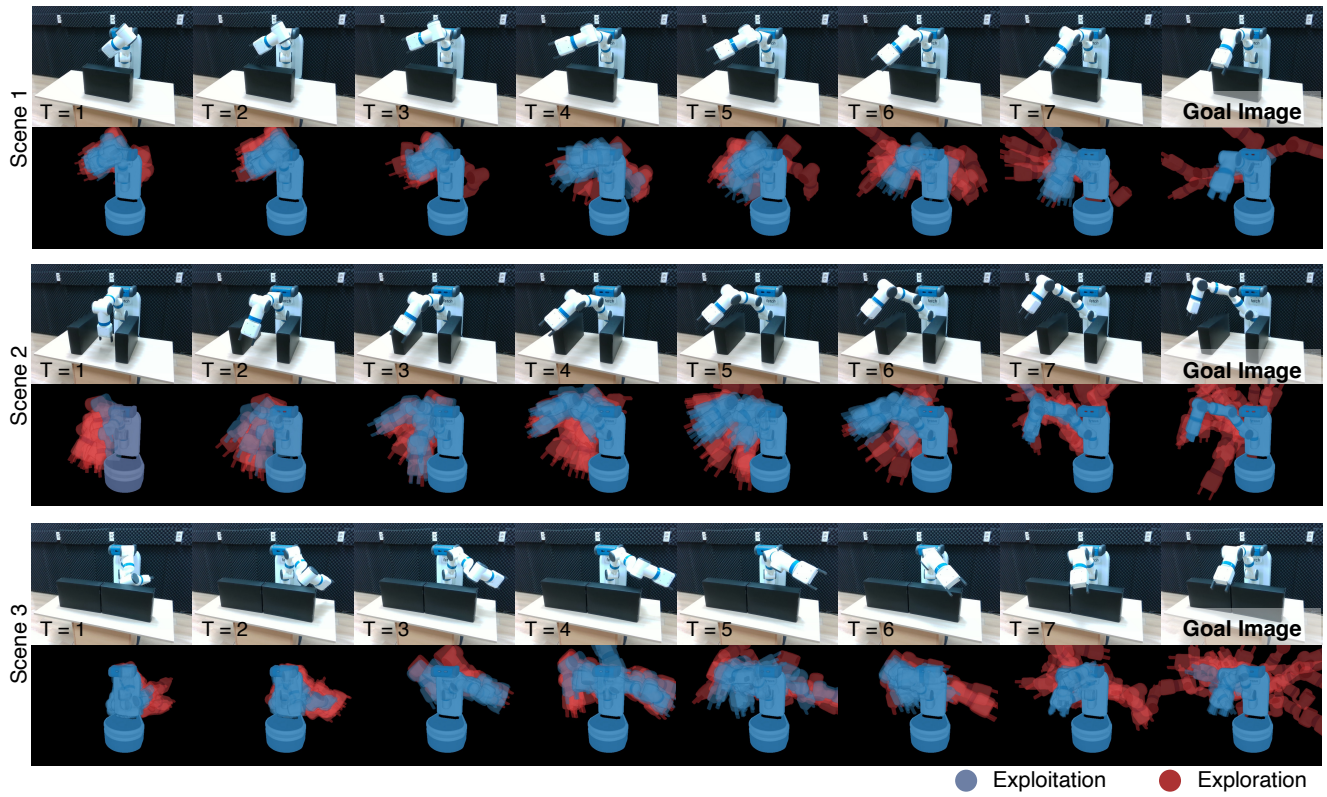

Figure S2. **Fetch deployment across three scenes.** Representative executions from three different obstacle configurations. Top row shows executed paths reaching visual goals. Bottom row shows planning dynamics where exploitation samples (blue) follow visual gradients while exploration samples (red) maintain coverage. The videos are provided in the project page.

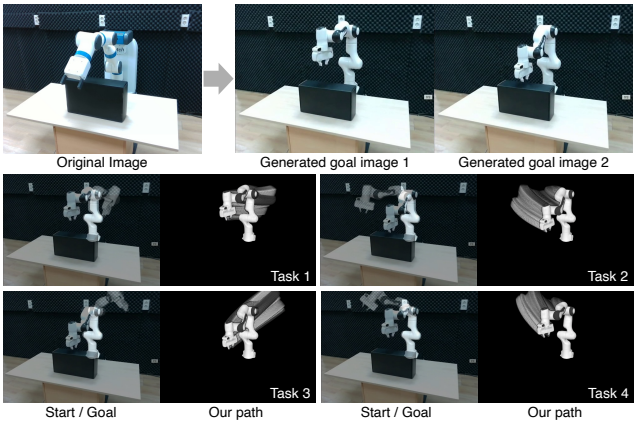

Figure S3. **Visual-goal motion planning with generated goal images.** To demonstrate broader applicability, we generate goals by prompting an generation model [2] to inpaint a Franka robot into Fetch scenes while maintaining background. Four tasks show paths matching synthesized poses, demonstrating potential to bridge generative models with executable planning.

each scene, we design 25 tasks requiring the robot to reach goals on either side or front/back of obstacles, verified for reachability via RRT. Tab. S4 shows vRRT achieves 80% success rate (60/75 tasks), demonstrating promising sim-to-real transfer. Fig. S2 shows representative executions and planning, successfully guiding the robot to visual goals despite domain gap. Failures occur when RGB appearance similarities between robot and scene elements create misleading visual gradients.

C.3. vRRT with Generated Goal Images

To further demonstrate practical generality of our visual-goal planning framework, we introduce an additional visual-goal motion planning experiment where the goal image is generated by an image generation model from a natural-language description. While recent advances enable synthesis of visually realistic robot scenes, generating motion paths that align with these synthesized frames remains challenging [7]. We use an image generation model [2] to replace the Fetch robot in Fig. S3 with a Franka Emika Panda while preserving background and viewpoint. To guide robot appearance, we provide our differentially rendered Franka as a conditioning image. For each synthesized goal, we randomly sample start configurations and plan paths using vRRT. Fig. S3 shows successful recovery of target configurations, demonstrating vRRT's potential to bridge generative models with executable planning.

Table S5. **Computational cost per iteration.** Average time allocation (%) when expanding 32 nodes per iteration across three robot platforms. Rendering dominates the cost indicating that reducing Gaussians directly improves vRRT efficiency.

Robot (# of Gaussians)	Rendering	Frontier sampling	RRT operations	Collision checking
Franka (52,393)	80.7	2.6	11.9	4.7
UR5e (16,327)	60.6	4.0	21.6	13.8
Fetch (70,335)	84.7	1.9	9.4	4.1

C.4. Tree Structure Analysis

To visualize how vRRT explores the configuration space during planning, we project robot joint configurations of our tree structure to 2D subspace via Principal Component Analysis (PCA) [10]. Each node in the visualization represents a single configuration from the robot C-space.

Frontier selection. In Fig. S4, we visualize the effectiveness of frontier-based exploration-exploitation scheme during tree expansion. Frontier nodes (orange), sampled via truncated geometric distribution based on visual loss ranking, concentrate toward the promising goal region, while non-frontier nodes (gray) maintain exploration coverage. This spatial distribution validates that our rank-based probabilistic sampling strategy effectively biases tree growth toward visually aligned configurations.

Inertial gradient expansion. Fig. S5 shows trees colored by inertial gradient expansion steps i. Low values (purple) indicate configurations primarily discovered through random steering exploration, while high values (yellow) represent configurations generated through our inertial gradient tree expansion. The tree structure reveals two key characteristics. First, exploration establishes broad coverage from the start, while gradient-steered nodes concentrate near the goal. Second, the convergence patterns differ across instances: (a) exhibits a cohesive stream of nodes flowing toward the goal, whereas (b, c) show multiple distinct branches extending to the target. These different convergence patterns suggest that inertial expansion can flexibly support both steady refinement along a dominant branch and parallel pursuit of multiple promising routes. These observations validate inertial gradient tree expansion: smooth color transitions along branches confirm continuous optimization trajectories through state inheritance, while the coexistence of low and high i nodes demonstrates effective balance between exploration and exploitation throughout planning.

C.5. Computational Cost

We analyze vRRT's computational cost during tree expansion, where 32 nodes are inserted per iteration through exploration and exploitation. Tab. S5 shows the average time breakdown across components. Differentiable rendering dominates (60.6-84.7%), as each gradient-based expansion requires evaluating visual loss and backpropagating through the Gaussian Splatting model. This cost scales with the number of Gaussians representing the robot. Consequently, robots with more Gaussians exhibit higher planning times in Tab.1 of the main paper, indicating that reducing Gaussian count directly improves efficiency. Our algorithmic components, frontier-based sampling and inertial gradient tree expansion, account for moderate computational cost. Frontier sampling requires 1.9-4.0% through sorting cached rendering losses. RRT operations including random sampling, nearest-neighbor queries, steering, and tree rewiring, scale with tree size but remain independent of Gaussian resolution. Further optimization may be possible through improved data structures for nearest-neighbor queries. Collision checking is performed via MuJoCo and could be replaced with alternative methods depending on application requirements.

C.6. Visual Ambiguity in Goal Specification

By integrating sampling-based exploration with visual-gradient exploitation, vRRT inherits RRT's robust exploration that systematically covers the search space given sufficient iterations, while extending to visual-goal specifications. Our method demonstrates effective performance in both simulation and real-world settings by leveraging exploration to escape local minima that trap gradient-based methods. However, visual ambiguity presents a limitation where exploration cannot disambiguate visually indistinguishable configurations. Fig. S6 shows representative failure cases where vRRT converges to visually matching but configuration-space incorrect poses. Fetch and Franka demonstrate symmetric poses producing identical silhouettes; UR5e shows severe self-occlusion making distinct configurations appear identical. When multiple configurations are visually indistinguishable, determining the true goal from observation alone becomes infeasible. This represents a fundamental limitation of single-view specification, addressable through multi-view fusion or perceptual features rendering.

D. Additional Implementation Details

D.1. Dataset Construction

We construct two synthetic test datasets for motion planning and pose reconstruction using the same sampling process, differing only in scene construction. For each robot platform, we define a fixed canonical pose as the start configuration q_{start} for all test queries, following the same convention as Dr.Robot [8], shown as faded poses in Fig. S1.

Goal configuration sampling. We generate goal configurations by randomly perturbing all revolute joints from the canonical pose within their physical limits. We verify feasibility of each sampled goal by running RRT [6] from the canonical pose with a 30-second time budget. Queries are discarded if RRT fails to find a collision-free path or if executing the trajectory in MuJoCo reveals collisions. Vali-

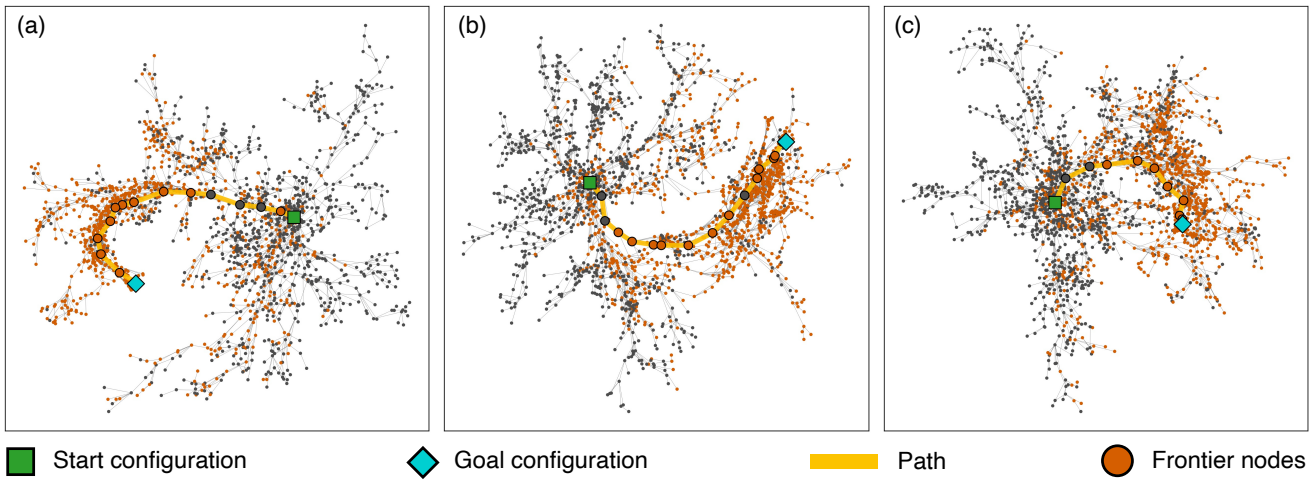

Figure S4. **Tree structure with frontier node visualization.** Configuration space projected onto 2D via PCA for three planning instances (a-c). Frontier nodes (orange) represent configurations with low visual loss selected for effective expansion, while gray nodes indicate non-frontier configurations. The spatial distribution shows frontier nodes progressively concentrate toward the goal region, validating that our frontier-based sampling effectively prioritizes visually promising areas for exploration and exploitation.

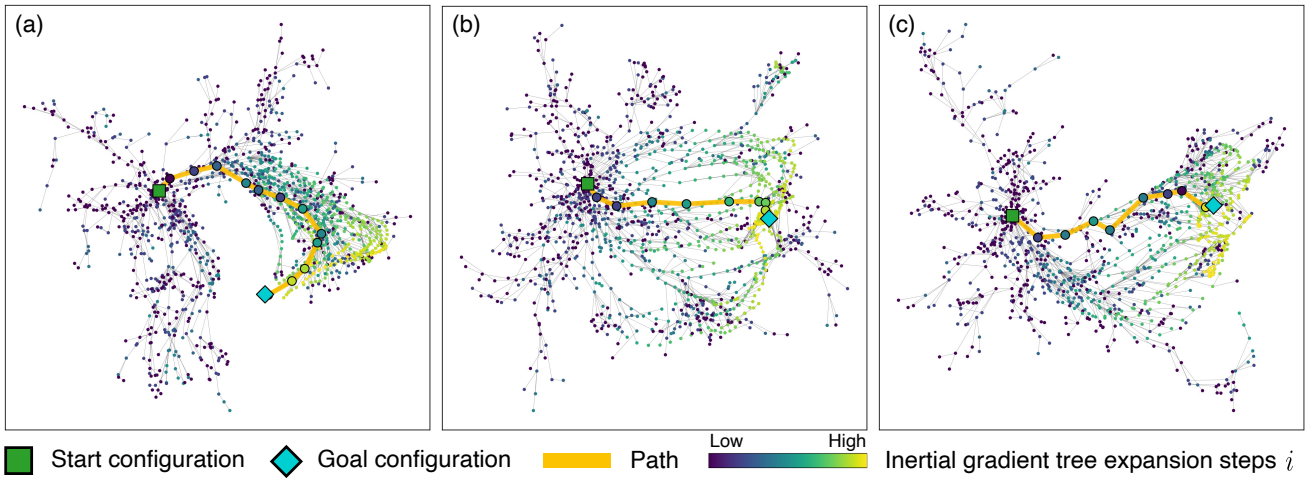

Figure S5. **Tree structure colored by inertial gradient expansion steps.** Robot joint configurations visualized in 2D via PCA across three planning instances (a-c). Node colors indicate inertial gradient tree expansion steps i: low values (purple) represent nodes primarily reached through exploration, while high values (yellow) indicate nodes refined through extensive gradient optimization. Progressive color transitions show that state inheritance enables continuous optimization across tree branches. Notably, convergence patterns vary across instances: (a) exhibits a cohesive gradient flow where optimization steps increase smoothly toward the goal, whereas (b) and (c) show multiple distinct optimization branches reaching the target.

dated queries are stratified by configuration-space distance $\|q_{\text{start}} - q_{\text{goal}}\|_2$ into bins at $[0.5, 1.0, 1.5, 2.0, 2.5]$ radians.

Scene construction. We construct six distinct scenes per robot: obstacle environments for motion planning and obstacle-free environments for pose reconstruction. For motion planning, each environment contains ten box-shaped obstacles with random dimensions drawn from predefined ranges (3-20 cm for Franka and UR5e; 5-70 cm for Fetch). We employ a two-stage placement strategy: (1) obstacles are positioned near robot links in the canonical pose without intersecting geometry; (2) remaining obstacles are placed randomly within the workspace. For pose reconstruction, scenes remain obstacle-free to isolate visual goal matching from collision avoidance.

Dataset statistics. Our dataset comprises 100 validated queries per distance bin per robot. For motion planning, this yields 500 queries per robot across five bins with six scene variations each, totaling 9,000 problems. For pose reconstruction, we generate 500 queries per robot in obstacle-free scenes, totaling 1,500 problems.

D.2. Pipeline Details

Robot Gaussian Construction. We construct differentiable robot representations for all three platforms following

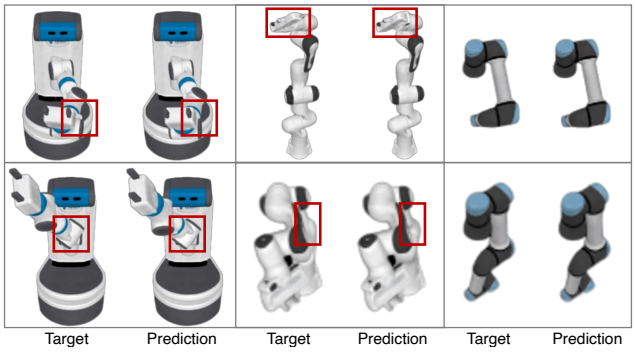

Figure S6. **Representative failure cases.** Target pose images (left) visually resemble predictions (right) but differ in joint space. Fetch and Franka: symmetric poses produce identical silhouettes. UR5e: self-occlusion allows hidden joints to rotate without visual change. These illustrate the inherent difficulty of visual-goal planning.

Dr.Robot [8]. Starting from MJCF XML descriptions in MuJoCo [13], which define kinematic structure, joint limits, and collision geometries, we sample diverse joint configurations within physical limits and render multi-view RGB images with varying camera poses. Franka and UR5e use standard MuJoCo models, while we custom a MJCF for Fetch following MuJoCo conventions.

Differentiable robot rendering. We train 3D Gaussian Splatting models [4] to represent each robot. The optimization follows standard Gaussian Splatting training, learning positional and rotational parameters per primitive. The resulting models combine forward kinematics with implicit linear blend skinning [9], enabling differentiable rendering at arbitrary joint configurations.

Rendering during Planning. We render the robot at the same resolution as the goal image (480×480) and compute a per-pixel L_2 loss between the rendered and goal images. The pre-trained Gaussian model provides visual gradients to the configuration space without additional training during planning. Since the visual gradient $\nabla_q \mathcal{L}_{\text{render}}(q_p)$ at a parent node q_p remains constant throughout a planning iteration, we implement a caching mechanism to avoid redundant rendering operations during tree expansion.

Optimization State and Settings. The root node (starting configuration) and nodes generated via random steering (exploration) are initialized with zero optimization states: $m = \mathbf{0}$, $v = \mathbf{0}$, and $i = 0$. In contrast, nodes created through visual-gradient steering (exploitation) inherit and update the optimization states from their parent nodes as described in Eq. (3)-(5) of the main paper, enabling momentum-consistent gradient descent trajectories across tree branches. We use $M = 200$, $r = 0.3$, $\eta = 0.7$, $\beta_1 = 0.9$, and $\beta_2 = 0.9$ for experiments. For real-world settings, we use RealSense factory intrinsics and estimate camera-to-robot extrinsics via hand-eye calibration, assuming fixed placement.

References

[1] Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, et al. Symbolic discovery of optimization algorithms. *Advances in neural information processing systems*, 36:49205–49233, 2023. 2

[2] Google DeepMind. Gemini, 2025. Generative model used for image synthesis. 4

[3] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011. 2

[4] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkuehler, and G. Drettakis. 3d gaussian splatting for real-time radiance field rendering. In *ACM Transactions on Graphics*, 2023. 7

[5] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. 2

[6] Steven LaValle. Rapidly-exploring random trees: A new tool for path planning. *Research Report 9811*, 1998. 5

[7] Sangmin Lee, Sungyong Park, and Heewon Kim. Dynscene: Scalable generation of dynamic robotic manipulation scenes for embodied ai. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 12166–12175, 2025. 4

[8] Ruoshi Liu, Alper Canberk, Shuran Song, and Carl Vondrick. Differentiable robot rendering. In *8th Annual Conference on Robot Learning (CoRL)*, 2024. 2, 5, 7

[9] M. Loper, Naureen Mahmood, J. Romero, Gerard Pons-Moll, and Michael J. Black. Smpl: A skinned multi-person linear model. 2023. 7

[10] Andrzej Maćkiewicz and Waldemar Ratajczak. Principal components analysis (pca). *Computers & Geosciences*, 19(3): 303–342, 1993. 5

[11] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17, 1964. 1

[12] Tijmen Tieleman. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26, 2012. 2

[13] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012. 7