

Towards End-to-End Safe Motion Planning for High-Dimensional Robot Systems

by

Challen Enniful Adu

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Robotics)
in the University of Michigan
2025

Doctoral Committee:

Professor Ram Vasudevan, Co-Chair
Professor Talia Moore, Co-Chair
Professor Maani Ghaffari
Professor Jessy Grizzle

Challen Enniful Adu
enniful@umich.edu
ORCID iD: 0009-0002-5957-3500
© Challen Enniful Adu 2025

DEDICATION

To my parents and sister, whose unwavering support and sacrifices made this possible.
To my girlfriend, who has been my rock during challenging times and always believes in me.
To my friends and extended family, who have always been cheering me on in the wings.

TABLE OF CONTENTS

DEDICATION	ii
LIST OF FIGURES	vii
LIST OF TABLES	xii
LIST OF APPENDICES	xv
ABSTRACT	xvi
CHAPTER	
1 Introduction	1
1.1 Motivation	1
1.2 Contributions	3
2 Rapid Trajectory Optimization with Pseudospectral Techniques and the Affine Geometric Heat Flow Equation	5
2.1 Introduction	5
2.1.1 Related Work	6
2.1.2 Chapter Contributions	8
2.2 Preliminaries	9
2.3 The Affine Geometric Heat Flow (AGHF) Partial Differential Equation . . .	10
2.3.1 Homotopies and Extracting Control Inputs	11
2.3.2 Ensuring $\mathcal{A}$ Coincides with the Control Input by Designing G . . .	12
2.4 Solving the AGHF Rapidly for High Dimensional Systems	13
2.4.1 Computing AGHF PDE Partial Derivatives Analytically	13
2.4.2 Pseudospectral Method for Solving the AGHF	15
2.4.3 Computing the PHLAME Jacobian Partial Derivatives Analytically	17
2.4.4 Rapidly Evaluating the AGHF (2.15) and its Jacobian (2.24)	18
2.5 Incorporating constraints into the AGHF	21
2.5.1 Constraint Lagrangian	21
2.6 Experiments and Results	24
2.6.1 Experimental Setup	24
2.6.2 PHLAME AGHF Evaluation vs Original AGHF [72]	27
2.6.3 Scalability with Increasing State Dimension in Unconstrained Systems	27
2.6.4 Trajectory Optimization for the Kinova Arm without Obstacles . .	29

2.6.5	Trajectory Optimization for Pinned Digit without Obstacles	30
2.6.6	Trajectory Optimization for the Kinova Arm with Obstacles	31
2.6.7	Trajectory Optimization for Pinned Digit with Obstacles	32
2.7	Conclusion and Limitations	34
3	Rapid Motion Planning with the AGHF PDE for Arbitrary Objective Functions and Constraints	35
3.1	Introduction	35
3.1.1	Related Work	35
3.1.2	Chapter Contributions	37
3.2	Preliminaries	38
3.3	The Affine Geometric Heat Flow Partial Differential Equation	39
3.3.1	Homotopies and the Action Functional	40
3.3.2	Solving the AGHF Rapidly	41
3.3.3	Ensuring $\mathcal{A}$ Coincides with the (OCP) Cost Function	42
3.4	Incorporating Arbitrary Constraints into the AGHF	44
3.4.1	Constrained Lagrangian	44
3.4.2	Incorporating Input Constraints in the Lagrangian	45
3.5	Designing a Phase 1 - Phase 2 Algorithm using the AGHF	46
3.6	Experiments and Results	47
3.6.1	Implementation of BLAZE	48
3.6.2	Experimental Setup	48
3.6.3	BLAZE Scalability with Increasing System Dimension	50
3.6.4	Kinova Gen3 Sphere Obstacle Avoidance	50
3.6.5	Kinova Gen3 Hard Task-Based Scenarios	51
3.6.6	Hardware Experiments	55
3.7	Conclusion	56
3.8	Limitations	57
4	Risk-Aware, Real-Time, Reachability-Based Motion Planning	58
4.1	Introduction	58
4.1.1	Related Work	58
4.1.2	Chapter Contributions	60
4.2	Notation	61
4.3	Preliminaries	62
4.3.1	Open Loop Vehicle Dynamics	62
4.3.2	Trajectory Parameterization	64
4.3.3	Closed Loop Vehicle Dynamics	64
4.3.4	Ego Vehicle, Environment, and Sensing	65
4.4	Online Planning Under Uncertainty	66
4.4.1	Obstacle Uncertainty	66
4.4.2	Risk-Aware Vehicle Safety	67
4.4.3	Online Optimization	67
4.5	Offline Reachability Analysis	68
4.6	An Implementable Alternative to (Opt-R)	69

4.6.1	Chance Constraint Relaxation	70
4.6.2	Tractable Online Optimization	72
4.6.3	Constraint Gradient and Parallelization	73
4.6.4	Online Operation	73
4.7	Experiments and Results	74
4.7.1	Tightness and Generality of Risk Approximation	75
4.7.2	Simulations	75
4.7.3	Hardware	79
4.8	RADIUS 2.0: DIAMETER – Generating a tighter risk over-approximation .	81
4.8.1	RADIUS’s Conservatism	81
4.8.2	DIAMETER’s Risk Over-Approximation Overview	82
4.8.3	DIAMETER’s Chance Constraint Relaxation	82
4.8.4	DIAMETER’s Tractable Online Optimization	88
4.8.5	DIAMETER’s Constraint Gradient and Parallelization	89
4.8.6	Evaluating DIAMETER’s Risk-Overapproximation	89
4.9	Conclusion	90
5	Let’s Make a Splat: Risk-Aware Trajectory Optimization in a Normalized Gaussian Splat	92
5.1	Introduction	92
5.1.1	Related Work	94
5.1.2	Chapter Contributions	97
5.2	Proposed Method Overview	98
5.2.1	Notation	98
5.2.2	Overview of Optimization	98
5.3	Robot Representation	99
5.3.1	Arm Kinematics	99
5.3.2	Arm Occupancy	100
5.4	Detailed Description of SPLANNING	102
5.4.1	3D Scene Representation	102
5.4.2	Bounding the Probability of Collision	104
5.4.3	Evaluating the Collision Bound	105
5.4.4	Risk-Aware Motion Planning	106
5.5	Experimental Results	107
5.5.1	Normalized 3DGS Evaluation	107
5.5.2	Simulation Environment	109
5.5.3	Collision Probability Evaluation	110
5.5.4	Planner Performance Evaluation	112
5.5.5	Real-World Demonstrations	114
5.6	Extending SPLANNING to Mobile Manipulation	116
5.6.1	Mobile Manipulator Kinematics	116
5.6.2	Mobile Manipulator Forward Occupancy	117
5.6.3	Risk-Aware Motion Planning for a Mobile Manipulator	119
5.7	Conclusion	121

6 Conclusion: Towards End-to-End Safe Motion Planning for High-Dimensional Robots	123
6.1 Summary of Contributions	124
6.2 Future Work	125
 APPENDICES	 127
A.1 Proof of Theorem 5	127
A.2 Proof of Theorem 6	133
A.3 PHLAME Results Appendix	138
B.1 Proof of Lemma 12	141
B.2 Proof of Theorem 15	142
B.3 Proof of Lemma 19	146
B.4 Experiment Hyperparameters	146
C.1 Proof of Lemma 32	151
C.2 Additional Details	151
C.3 Additional Experiments	154
D.1 Trajectory Design	157
D.2 Reachability Analysis	157
D.3 Splanning Proofs	159
D.4 Baseline Implementation Details	163
D.5 Normalized 3D Gaussian Splatting	164
 BIBLIOGRAPHY	 176

LIST OF FIGURES

FIGURE

2.1	PHLAME works by first taking in some initial guess of a trajectory (trajectory of i th state shown in dark blue) which does not have to be dynamically feasible and evolves it into some dynamically feasible final trajectory dark green). Both trajectories start and end at $\mathbf{x}(0) = \mathbf{x}_0$ and $\mathbf{x}(T) = \mathbf{x}_f$ respectively. Notice that at the initial trajectory Digit (a high dimensional humanoid robot) has a dynamically infeasible set of configurations during it's stepping trajectory and that at the end of the PHLAME solve that trajectory is made into a dynamically feasible one where Digit is able to step over the box.	6
2.2	Scaling trend of the mean evaluation times (in μs) of the right-hand side of the AGHF using Algorithm 2 as the number of bodies (N) increases from 2 to 22. The systems with N between 2 and 6 are penduli systems (i.e., 2-link pendulum, 3-link pendulum, etc.). The $N = 7$ system is the Kinova Gen3 and the $N = 22$ system is the pinned Digit V3 biped. Each robot's AGHF RHS was evaluated at a 1000 random robot configurations. Notice the polynomial line of best fit lacks a significant N^3 term, suggesting that the right-hand side computation time scales approximately quadratically with N in practice.	20
2.3	Scaling trend of the mean evaluation times (in μs) of the AGHF Jacobian using Algorithm 3 as the number of bodies (N) increases from 2 to 22. The systems with N between 2 and 6 are penduli systems (i.e., 2-link pendulum, 3-link pendulum, etc.). The $N = 7$ system is the Kinova Gen3 and the $N = 22$ system is the pinned Digit V3 biped. Each robot's AGHF Jacobian was evaluated at a 1000 random robot configurations. Notice the polynomial line of best fit lacks a significant N^4 term, suggesting that the Jacobian computation time scales approximately cubically with N in practice.	23
2.4	A bar plot comparing the mean solve times for three different trajectory optimization algorithms: PHLAME, AGHF [72] and Crocoddyl [79]. For $N \leq 5$ the results correspond to the 1-5 link pendulum, for $N = 7$ to the Kinova Gen3 Arm and for $N = 22$ for the Pinned Digit biped robot. Each experiment was run ten times using the best solver parameter set. For $N \geq 5$ we do not show results for the Original AGHF implementation as no results were obtained in a reasonable amount of time. Overall, we see that PHLAME shows better empirical scalability and solve time than the other methods. The values of $\mathbf{x}_0$ and $\mathbf{x}_f$ are provided in https://github.com/roahmlab/PHLAME-comparisons	28

2.5	Side-by-side figures showing the convergence of action functional and $\int_0^T u(t) ^2 dt$ for one of the trajectories found in the kinova no obstacle trajectory optimization problems. The lines transitions from dark blue to dark green as the trajectory converges to an extremal solution of (2.7)	29
2.6	Snapshots of the forward simulated solution obtained by PHLAME for Pinned Digit doing a yoga stretch. The intermediate poses in green , the initial in turquoise and the final in gold	30
2.7	Side-by-side figures showing the convergence of action functional and the evolution of $\int_0^T u(t) ^2 dt$ for the Digit yoga stretch scenario.	31
2.8	Two figures show trajectories generated by PHLAME for the Kinova Gen3 to get from the initial poses in turquoise to the final poses in gold while avoiding the obstacles in red . The intermediate poses in the paths it takes are shown in green	32
3.1	This chapter introduces BLAZE, a Phase 1 - Phase 2 Affine Geometric Heat Flow (AGHF) framework, to rapidly solve optimal control problems while respecting robot constraints and avoiding obstacles. It begins with an initial trajectory (shown in orange with the color gradient illustrating the evolution in time starting from darkest and going to lightest) that may violate constraints (e.g., the second and fourth pose of the arm are in collision with the boxes and outlined in red). If the initial trajectory is infeasible, BLAZE enters Phase 1, where it evolves the trajectory into a trajectory that satisfies all constraints (e.g., in the blue trajectory, the Kinova arm has been moved out of collision with the boxes). Once the trajectory satisfies all constraints, Phase 2 begins, optimizing the motion to minimize a user-specified cost function while maintaining feasibility (optimized trajectory shown green). BLAZE optimizes the trajectory to reach a target configuration while avoiding the obstacles while considering the full dynamical model of the arm. Note that optimal control (including Phase 1 and Phase 2) for this 14 dimensional state space model is completed within 3s while satisfying input, state, and collision avoidance constraints.	36
3.2	A bar plot comparing the mean solve times for four different trajectory optimization algorithms: BLAZE, RAPTOR, Crocoddyl and Aligator. For $N \leq 5$ the results correspond to the 1-5 link pendulum, $N = 7$ corresponds to the Kinova Gen3 Arm, $N = 14$ corresponds to a bimanual system of two Kinova Gen3 Arms, $N = 21$ corresponds to a trimanual system of three Kinova Gen3 Arms. Each experiment was run ten times. Overall, we see that BLAZE shows better scalability and solve times than the other methods as the system dimension increases.	51
3.3	A plot showing the evolution of the Action Functional versus the AGHF evolution parameter s for each of the the hard 2-second horizon task-based scenario experiments. The plot shows that all experiments ran to full convergence, each of them truncating at the particular s where the action functional converged to a minimum.	53

3.4	This figure shows a visualization of BLAZE's solution for one of the 2-second horizon task-based scenarios (scenario 3). In each subfigure, the color gradient illustrates the evolution in time where the start configuration is in the darkest shade and the end configuration in the lightest. The initial trajectory for the scenario is shown in the top image in yellow and collides with the obstacles along the path. The proposed algorithm is able to push this initial guess out of collision and generate the optimal collision-free solution shown in green in 3.73s for this scenario, whereas the other comparison methods cannot find a solution within the allotted time for this scenario.	55
3.5	This figure shows a series of still frames of the Kinova arm following the trajectory generated by BLAZE. Each subfigure depicts the arm at a different point in time. The initial guess collides with the obstacle. The proposed algorithm is able to push this initial guess out of collision and generate the optimal collision-free solution in 2.45s, whereas Aligator does it in 57.21s.	56
4.1	An illustration of the motion planning framework RADIUS that is developed in this chapter. RADIUS first performs offline reachability analysis using a closed-loop full-order vehicle dynamics to construct a series of control-parameterized, zonotope reachable sets (shown as dark gray boxes) that over-approximate all possible behaviors of the ego vehicle over the planning horizon. During on-line planning, given some user-defined risk of collision (ϵ) for the motion plan, RADIUS constructs a trajectory by solving an optimization problem that selects subsets of pre-computed zonotope reachable sets that are certified to have a no greater than ϵ risk of colliding with any obstacles. In this figure, the moving obstacles are shown in white and the $3\text{-}\sigma$ regions of the corresponding probability distributions for the obstacle locations are shown as the purple and blue ellipses where the probability density from low to high is illustrated from blue to purple. The subsets of the dark gray zonotope reachable sets corresponding to the trajectory parameter shown in green ensure a collision-free path that is guaranteed to have a no greater than ϵ risk of colliding with all obstacles, while the two trajectory parameters and their corresponding reachable sets shown orange may have a greater than ϵ risk of collision with the moving obstacles.	59
4.2	An illustration of chance constraint relaxation. Given arbitrary $(i, j) \in \mathcal{I} \times \mathcal{J}$, in (a) the risk of collision between the ego vehicle and the i -th obstacle during time interval $\mathcal{T}_j$ is relaxed as the integration of probability density function $q_{i,j}$ (shown in purple and blue) over zonotope $\xi_j(z_0, p) \oplus \langle 0, G^{\text{obs}} \rangle \subset \mathcal{W}$ (shown in green). In (b), $\xi_j(z_0, p) \oplus \langle 0, G^{\text{obs}} \rangle$ is over-approximated by a collection of right-angled triangles colored in white and black depending on if a triangle has a nontrivial intersection with $\xi_j(z_0, p) \oplus \langle 0, G^{\text{obs}} \rangle$ or not. In (c), Interval Arithmetic is used to generate an over- and under-approximation of $q_{i,j}$ over each right-angled triangle. And in (d), the integration of the over-approximation of $q_{i,j}$ over each right-angled triangle is computed in closed form.	69

4.3	Example of a single planning iteration of a simulated scenario in which RADIUS is able to navigate the ego vehicle (black) to the provided waypoint (black cross) through a lane change maneuver solved by one planning iteration, while CCPBA and REFINE execute contingency braking maneuvers as they are unable to find feasible solutions. Cantelli MPC results in a crash, as it has no fail-safe maneuver, so it instead maintains its velocity and continues searching for a feasible solution to execute the lane change. Forward reachable sets are shown in green. Obstacles are shown in white and are marked by their indices to make them trackable among different time instances. Probability distributions are illustrated as purple and blue ellipses where the probability density from low to high is illustrated from blue to purple. Note that for visualization purposes we have buffered the probability distributions shown in this figure by the footprints of the obstacles. Lastly, the convex hull that REFINE uses to over-approximate the $5\text{-}\sigma_{i,j}$ regions of the obstacle probability distributions are shown as gray boxes around the obstacles for all $(i, j) \in \mathcal{I} \times \mathcal{J}$. Note that the distances on the x-axis and y-axis are in [m].	77
4.4	Example of a simulated unguarded left turning trial where the ego vehicle (black) navigates through a gap in the oncoming traffic to successfully execute the unguarded left turn. Forward reachable sets are shown in green. Obstacles are shown in white and are marked by their indices to make them trackable among different time instances. Probability distributions are illustrated as purple and blue ellipses where the probability density from low to high is illustrated from blue to purple.	79
5.1	SPLANNING constructs risk-aware trajectories in a Gaussian Splatting map in real-time. The top image shows a real-world scene that a 7DOF serial manipulator must plan through, starting at the blue configuration (left) and ending at the green configuration (right). The scene is represented as a normalized 3D Gaussian Splat. Then, in real-time, SPLANNING solves an optimization problem that constrains the probability that the robot's forward occupancy (bottom center, purple) collides with the scene.	93
5.2	SPLANNING optimizes trajectories in a Normalized 3D Gaussian Splat given a start configuration (blue) and goal configuration (green). Offline, a Normalized 3D Gaussian Splat is constructed to represent the scene geometry (Sec. 5.4.1, bottom right). Online, a family of parameterized trajectories (App. D.1) is partitioned into a finite set of intervals (App. D.2, top left). Then, for each time interval, the Spherical Forward Occupancy ($\mathcal{SFO}$) (purple) is computed as an overapproximation of the robot's swept volume (Sec. 5.3.2, bottom left). Finally, during online trajectory optimization (Sec. 5.4.4, bottom middle), a novel constraint (Sec. 5.4.2–5.4.3) bounds the probability that the $\mathcal{SFO}$ intersects with the scene, as represented by a Normalized 3D Gaussian Splat.	95

5.3	SPLANNING generates safe trajectories in densely cluttered environments in simulation. The left panel shows discrete time steps of sequential trajectories that brings the arm from the start configuration (blue) safely to the goal configuration (green). The right panel shows an intermediate planning step where the Spherical Forward Occupancy (purple) avoids the obstacles represented by a Normalized 3D Gaussian Splat.	110
5.4	The collision constraint representation of CATNIPS, Splat-Nav, and SPLANNING are compared. CATNIPS transforms a NeRF into a 3D occupancy grid by relating it to a Poisson Point Process; after convolving this grid with a robot kernel, the center of the robot is checked for collision with the grid. Splat-Nav deterministically evaluates whether the confidence ellipsoids of each Gaussian intersect with a spherical robot. Finally, SPLANNING integrates a Normalized 3D Gaussian Splat over an overapproximation of the robot arm to form a risk constraint.	111
5.5	Two Precision-Recall curves are presented. Treating the collision constraints from SPLANNING, CATNIPS, Splat-Nav as classifiers, where a positive indicates collision and a negative indicates no collision, the top plot shows the Precision-Recall over individual spheres while the bottom plot show Precision-Recall over configurations. Highlighted markers indicate nominal parameters $\alpha = \beta = 0.025$, $\sigma = 0.99$, and $\sigma = 1$ for SPLANNING, CATNIPS, and Splat-Nav respectively.	122

LIST OF TABLES

TABLE

2.1	A table showing mean evaluation times (in μs) of the right-hand side of the AGHF using PHLAME and AGHF [72] as the number of bodies (N) increases from 2 to 22. For $N \leq 5$ the results correspond to the 1-5 link pendulum, for $N = 7$ to the Kinova Gen3 Arm and for $N = 22$ for the Pinned Digit biped robot. For each method each robot's AGHF RHS was evaluated at a 1000 random robot configurations each. For $N > 5$ the MATLAB method used in [72] was unable to symbolically generate the AGHF right-hand side without running out of memory. We denote the results where [72] was unable to generate the right-hand side and find a solution as DNF (Did Not Find). Regardless, we see that PHLAME's right-hand side evaluation time is much faster and scales better than the original AGHF [72].	27
2.2	Comparison of success rate, control effort norm ($\int_0^T \ u_{fb}(t)\ ^2 dt$), and solve time for Kinova Gen3 trajectory optimization experiments without obstacles.	30
2.3	A table showing the success rate, norm of the control effort ($\int_0^T \ u_{fb}(t)\ ^2 dt$) and solve time in seconds for a set of Digit V3 trajectory optimization experiments with no obstacles	31
2.4	Comparison of success rate, control effort norm ($\int_0^T \ u_{fb}(t)\ ^2 dt$), and solve time for Kinova Gen3 trajectory optimization experiments with obstacles. The 'Intersection' rows report metrics for trials where both methods succeeded; the 'Successes' rows report metrics for successful trials of each method.	33
2.5	A table showing the success rate, norm of the control effort ($\int_0^T \ u_{fb}(t)\ ^2 dt$) and Solve Time in seconds for Digit trajectory optimization experiment where Digit steps over different sized obstacles when the threshold for success ϵ is set to 0.05. We denote the results where one of the methods was unable to generate a successful solution as DNF (Did Not Find).	33
2.6	Comparison of success rate, control effort norm ($\int_0^T \ u_{fb}(t)\ ^2 dt$), and solve time for Digit trajectory optimization experiments over obstacles with a success threshold $\epsilon = 0.25$. The 'Intersection' rows report metrics for trials where both methods succeeded; the 'Successes' rows report metrics for successful trials of each method.	34
3.1	Comparison of Success Rate, Objective Cost and Solve Time for Kinova Gen3 trajectory optimization experiments with obstacles. Note that the Objective Cost and Solve Time numbers are just presented for scenarios where both methods were successful.	52

3.2	Objective Cost and Solve Time for BLAZE compared with Aligator after its 1st and 3rd iterations on Kinova Gen3 obstacle-avoidance tasks, where Aligator is permitted to return intermediate suboptimal solutions. Results are aggregated over all 20 scenarios, rather than only successful trials as in Table 3.1 due to a large proportion of the Aligator optimizations after the first iteration not satisfying the success criteria.	52
3.3	Detailed comparison of Objective Cost and Solve Time for the 2-second horizon task-based scenarios. All the problems consider input, state and cuboid obstacle constraints. BLAZE is depicted in green , Aligator is depicted in gold , RAPTOR is depicted in purple and PHLAME (which is a version of BLAZE that does not utilize a Phase1-Phase2 algorithm) is depicted in pink . In this table "F" denotes that a scenario was considered a failure as per the definition introduced in Section 3.6.2.2.	54
4.1	Results for the risk of collision estimation error of RADIUS and the Cantelli Inequality when the obstacle location is represented by three types of probability distributions. Note that CCPBA can only handle Gaussian distributions so does not have results for the other types of distributions.	76
4.2	Simulation results comparing RADIUS to CCPBA, Cantelli MPC and REFINE on a 1000[m] stretch of dense highway. A successful trial means the ego vehicle was able to successfully travel 1000[m] without crashing or coming to a stop. . .	78
4.3	Simulation results over a single planning iteration for RADIUS executing a left turn at different risk thresholds. Average Time To Goal (ATTG) and Maximum Time To Goal (MTTG) are only taken over successful trials.	80
4.4	Mean and maximum errors in estimated collision risk (relative to Monte-Carlo ground truth) for each method, evaluated over four different test cases with different obstacle distributions: Gaussian-S (only a small fraction of Gaussian probability mass lies inside each zonotope), Gaussian, Beta, and Bimodal distributions, where each are allowed to have more substantial probability mass in each zonotope. CCPBA is only applicable to Gaussian cases.	91
5.1	Reconstruction evaluation on Replica and TUM-RGBD. Standard 3DGS slightly outperforms the normalized variant. However, the performance is comparable, and only Normalized 3DGS is suitable for risk-aware motion planning.	107
5.2	The RMSE depth error (m) is computed per evaluation image, then averaged over each image in the dataset. The standard 3DGS slightly outperforms the Normalized variant, but only Normalized 3DGS is suitable for risk-aware planning.	108
5.3	Number of successes for PHLAME, SPARROWS, ARMTD, MPOT, CHOMP, and TrajOpt in the Kinova planning experiment with 10, 20, and 40 randomly-placed obstacles. For each set of experiments, 100 randomly-generated scenes were tested. The first number reported is the number of successes (higher is better). The second number, orange , indicates trials that failed to reach the goal state but did not result in crashes. Red indicates the number of failures due to collision (lower is better). The most and second most successes are annotated.	112
5.4	The peak memory usage of SPLANNING was measured for each execution, then averaged across the 100 randomly-generated scenes for each number of obstacles.	112

5.5	Mean per-step planning time for SPLANNING, SPARROWS, and ARMTD in Kinova planning experiments with 10, 20, and 40 obstacles with a 0.5s time limit. For SPLANNING, the first planning horizon is excluded from each timing measurement as the system warms up.	114
5.6	Mean runtime for constraint and constraint gradient evaluation for SPLANNING, SPARROWS, and ARMTD in Kinova planning experiments with 10, 20, and 40 obstacles. For SPLANNING, the first planning constraint evaluation is excluded from each timing measurement as the system warms up.	114
5.7	Hardware experiments commanded a robot to sets of three high-level waypoints then back to the start. Each trial listed corresponds to one such set of motions. A trial is counted a success only if all four motions succeed; otherwise, the first failure is reported. Asterisks* indicate the number of trials where the robot gently grazed an obstacle, but motion was not impeded. RRT and Optimization Time (s) were averaged across all executions for each configuration.	115
A.1	Parameter Values for Crocoddyl unconstrained	138
A.2	Parameter Values for Aligator with obstacles	139
A.3	Parameter Values for PHLAME for pendulum swingup	139
A.4	Parameter Values for PHLAME for Kinova without obstacles	139
A.5	Parameter Values for PHLAME for Digit without obstacles	139
A.6	Parameter Values for PHLAME Kinova with Constraints	140
A.7	Parameter Values for PHLAME for Digit with Constraints	140
B.1	Parameter settings for RAPTOR in the swing-up experiments.	147
B.2	Parameter settings for Aligator for the Scalability Experiment	148
B.3	Parameter settings for Aligator for the Sphere Obstacle Avoidance experiment .	148
B.5	Parameter settings for Crocoddyl for Scalability experiment	148
B.4	Parameter settings for Aligator for the 2s Task-Based scenario experiment . . .	149
B.6	Parameter settings for BLAZE Scalability Experiment	150
B.7	Parameter settings for BLAZE for Sphere Obstacle avoidance experiment. . . .	150
B.8	Parameter settings for BLAZE for 2s Task-Based scenario experiment.	150
C.1	Results comparing the computation times for generating an approximation for the risk of collision using RADIUS, CCPBA, and the Cantelli Inequality when the obstacle uncertainty is represented as a Gaussian distribution.	153
C.2	Single planning iteration results using RADIUS, CCPBA and Cantelli MPC. “Success” encompasses the trials where each method was able to successfully execute the lane change. “Other Action” encompasses the trials where a method did not complete the lane change maneuver, but instead either executed a safe stop maneuver or decided to keep driving in lane.	155
C.3	Single planning iteration results using the RADIUS variants considered in the ablation study. “Other Action” encompasses the trials where each method did not complete the lane change maneuver, but instead either executed a safe stop maneuver or decided to keep driving in the lane. RADIUS (No Analytical Gradient) is abbreviated as RADIUS (NAG), Monte-Carlo RTD (Discrete) is abbreviated as MCRTD(D) and Monte-Carlo RTD (Continuous) is abbreviated as MCRTD(C).	156

LIST OF APPENDICES

A PHLAME Appendices	127
Proof of Theorem 5	127
Proof of Theorem 6	132
PHLAME Results Appendix	138
B BLAZE Appendices	141
Proof of Lemma 12	141
Proof of Theorem 15	142
Proof of Lemma 19	146
Experiment Hyperparameters	146
C RADIUS Appendices	151
Proof of Lemma 32	151
Additional Details	151
Additional Experiments	154
D SPLANNING Appendices	157
Trajectory Design	157
Reachability Analysis	157
Splanning Proofs	159
Baseline Implementation Details	162
Normalized 3D Gaussian Splatting	164

ABSTRACT

Safe motion planning is essential for deploying autonomous robots in real-world environments, whether navigating cluttered warehouses, crowded public spaces, or unstructured outdoor settings. To perform reliably in these scenarios, robots must generate plans in real time, offer formal safety guarantees without undue conservatism, handle environmental uncertainty robustly, and tightly integrate visual perception with motion planning so they can react to changes as they occur.

There are several challenges associated with achieving safe autonomous navigation of mobile robots. First, generating dynamically feasible trajectories for high-dimensional systems is computationally demanding, often requiring significant processing time to produce high-quality solutions. Many safe motion planners treat these trajectories as high-level plans (or waypoints) and rely on a lower-level planner to enforce safety during execution. Consequently, overall performance hinges on the quality of the high-level plan so suboptimal plans can force frequent replanning or emergency halts. Second, accounting for environmental uncertainty, especially in dynamic settings with uncertain agent motions remains difficult. Planners typically use worst-case bounds on uncertainty, inflating safety margins and yielding overly conservative behaviors. Third, standard perception pipelines convert images into occupancy grids or decompose the scene into a set of convex primitives to simplify planning. These either discard fine geometric detail which may be useful for downstream task planning, or require significant and expensive preprocessing.

This dissertation proposes a cohesive suite of tools to address each challenge. First I present a collection of novel optimization-based trajectory generators (Chapters 2 and 3) for high-dimensional systems, built on the Affine Geometric Heat Flow (AGHF) PDE, which reformulates trajectory optimization as a curve-shortening process solved by forward-simulating an ODE system to its extremal curve. By combining pseudospectral collocation with spatial vector algebra, this framework dramatically reduces function evaluations and produces high-quality, dynamically feasible trajectories in seconds for systems with dozens of degrees of freedom (e.g., 44-dimensional legged robots)

Second, I introduce a risk-aware planning methodology (Chapter 4) that rigorously quantifies and bounds uncertainty via chance-constrained optimization and reachability analysis.

This method runs in real-time both in simulation and on hardware, and offers formal safety guarantees while avoiding the excessive conservatism of traditional uncertainty-aware safe planners.

Third, I propose a novel Gaussian splatting representation (Chapter 5) that allows probabilistic planning in radiance fields. By reformulating radiance field representations into a normalized Gaussian framework, and deriving tight upper bounds on collision probability this approach unifies perception and motion planning: the robot is able to use a single representation to reason about obstacle geometry while maintaining dense visual information that is useful for downstream task planning. Additionally, I present how this method can be applied on both fixed-base and mobile manipulators. Experiments in cluttered scenes and on a real-world fixed base manipulator demonstrate significant improvements in planning speed and collision avoidance.

Together, these contributions move us toward a unified architecture for safe, real-time motion planning of high-dimensional robotic systems in uncertain environments while closing the loop between perception and action with robust safety guarantees.

CHAPTER 1

Introduction

1.1 Motivation

Safe motion planning has become a critical requirement for autonomous robots that must operate reliably in environments ranging from densely packed warehouses to bustling public spaces and irregular outdoor terrain. In these settings, a robot cannot simply follow a precomputed path; it must generate new plans quickly, ensure safety without becoming so cautious that it stalls, and deal effectively with uncertainty in its sensors and surroundings. Yet most existing systems address these needs with separate modules whose limitations can undermine overall performance and safety.

In the sense-plan-act paradigm [4], the robot first acquires information about the robot's state and its environment using its sensors in the sensing phase, and ultimately executes motor commands in the action phase. Sitting between these stages, the planning module must produce a high-level plan that a mid-level planner or controller can reliably track. Typically it is this mid-level planner that is enforcing fine-grained safety constraints. This high-level plan may take the form of a full trajectory or a sequence of waypoints, but its quality directly affects the success of downstream execution and the quality of the final motion plan generated by the mid-level planner.

Trajectory generation methods grounded in optimal control offer a rigorous way to compute such high-level plans, but they typically rely on solving large nonlinear programs that demand substantial computation. In practice, this means planning can take seconds or even minutes, forcing roboticists to simplify robot models by either 1) ignoring dynamics or use reduced-order approximations or 2) invest substantial engineering effort into warm-starting solvers with hand-tuned initial guesses. Alternatively, many systems use sampling-based planners, such as RRT, to generate a sparse set of waypoints for the robot's subsequent mid-level planner to follow. While RRTs excel at quickly finding some feasible path, they suffer from poor coverage in high dimensions, lack explicit notions of trajectory smoothness or dy-

dynamic feasibility, and produce waypoints that require expensive post-processing to smooth or make more dynamically feasible. These limitations force a trade-off between speed and quality that current methods struggle to reconcile.

Accounting for uncertainty inherent in the dynamic environments the robots must operate in introduces its own set of challenges. Mobile robots must fuse noisy sensor data into predictions of how obstacles might move, then incorporate those predictions into their motion plans. Many existing methods struggle to run in real time: sampling-based chance-constrained planners require thousands of samples to estimate the risk of colliding with these uncertain obstacles accurately, leading to long planning times and frequent pauses. Other approaches make strong assumptions about form of the uncertainty models, most commonly assuming they are Gaussian models. This simplification fails to capture the multimodal or heavy-tailed uncertainty distributions exhibited by dynamic obstacles. Alternatively, some planners use worst-case bounds on uncertainty and tend to inflate safety margins, leading to overly cautious and conservative behavior, which can slow robot progress or even prevent the robot from completing its task. Together, these issues leave robots either too slow to react, insufficiently expressive in their risk models, or so conservative that they cannot operate effectively in cluttered, uncertain settings.

Perception pipelines further compound these issues by divorcing scene understanding from motion planning. Raw data from cameras or lidar sensors is often converted into occupancy grids or decomposed into primitive shapes to simplify collision checking. Such representations discard geometric detail that could help navigate cluttered environments or grasp small objects. Alternative approaches construct high-fidelity meshes or signed distance fields, but incur heavy preprocessing overhead. The result is a perception system that either slows down the planning process due to heavy preprocessing or sacrifices critical information, making it difficult for the robot to operate safely in real world conditions.

Together, these shortfalls in real-time computation, uncertainty handling, and perception integration reveal a need for approaches that can plan quickly with full knowledge of robot dynamics, quantify and incorporate uncertainty without overt conservatism, operate in real time, and maintain rich geometric detail for obstacle avoidance and other downstream planning tasks like manipulation. This dissertation proposes a cohesive suite of tools to address each of these shortcomings and move towards a unified end-to-end motion planner that takes in visual inputs and generates safe motion plans.

1.2 Contributions

This dissertation is organized around three sets of contributions, each addressing one of the key gaps in current safe motion planning methods and validated through simulation and hardware experiments. Broadly, my approach emphasizes first-principles, model-based optimization and analytic risk quantification rather than relying solely on black-box policy learning: by grounding planning in dynamics, geometry, and provable risk bounds we obtain interpretable behaviors that are amenable to certification. That said, the approach also accommodates learned components for perception, prediction, and uncertainty estimation. Yet these learned modules inform, rather than replace, the safety-critical planner, so safety decisions remain rooted in explainable optimization and analytic risk envelopes. This hybrid design would leverage the strengths of data-driven perception while preserving verifiability and more predictable generalization in safety-critical settings. This dissertation lays out a concrete roadmap towards an end-to-end motion planner that takes in visual inputs and generates safe motion plans, by developing three sets of complementary tools that can be combined in an end-to-end safe-motion planner.

The first set of contributions appears in Chapters 2 and 3. Here, I introduce two optimization-based trajectory generators grounded in the affine geometric heat flow partial differential equation. This method casts the trajectory optimization problem as a curve-shortening process solved via integrating forward a set of ordinary differential equations. Typically, generating trajectories in this way involves numerous expensive evaluations of the affine geometric heat flow PDE making scaling of this method to high-dimensional systems challenging. By leveraging pseudospectral collocation in conjunction with spatial vector algebra, this framework reduces the number of function evaluations of the affine geometric heat flow PDE by orders of magnitude. As a result, high-quality, dynamically feasible motions for robots with forty or more degrees of freedom can be produced in seconds, which contrasts sharply with conventional nonlinear program solvers that demand lengthy solve times or careful warm starts.

The second set of contributions is presented in Chapter 4. Here, I develop a risk-aware planning methodology that rigorously quantifies and bounds collision probability using chance-constrained optimization in concert with reachability analysis. This chapter introduces two different parallelizable, closed-form over-approximations of collision risk for any twice-differentiable obstacle location distribution and provides the analytical gradient of these risk approximations with respect to the planner’s control parameters. These over-approximations fit into a risk-aware planner that retains formal safety guarantees while avoiding the extreme conservatism of prior approaches, enabling the robot to fluidly avoid

moving obstacles with uncertainty in their locations while still having bounded safety guarantees. They are amenable to user-set levels of risk enabling more aggressive behaviors but still have guarantees on the level of safety given the risk threshold.

The third set of contributions appears in Chapter 5 and bridges perception and planning through a probabilistic framework built on radiance field representations. By reformulating neural radiance fields into a Normalized Gaussian Splatting scene representation, I present tight analytic bounds on collision risk that preserve the dense visual detail needed for downstream tasks. This unified representation allows a robot to reason about both geometry and appearance in a single model, yielding significant improvements in planning speed and collision avoidance in cluttered scenes without incurring heavy reconstruction overhead. This chapter discusses how this framework can be applied to both fixed-base and mobile manipulation, proposing a set of tools for risk-aware motion planning with safety guarantees in a gaussian splat.

CHAPTER 2

Rapid Trajectory Optimization with Pseudospectral Techniques and the Affine Geometric Heat Flow Equation

2.1 Introduction

To perform effectively in real-world applications, robots must generate dynamically feasible trajectories across a diverse range of tasks—including ground vehicle navigation, manipulation, and legged locomotion [46, 30, 82, 20, 69, 80, 55]. Optimal control plays a fundamental role in enabling these capabilities, providing the mathematical framework necessary for planning and executing complex movements under various physical constraints. For robotic applications, an optimal control algorithm must satisfy several critical requirements: (1) computational efficiency to enable online planning and replanning, (2) scalability to handle high-dimensional systems like humanoids and manipulators, (3) ability to incorporate nonlinear dynamics and physical constraints like torque limits which are that are critical for exploiting the full performance envelope of modern actuators [108, 109], and (4) reliable convergence with minimal sensitivity to initial conditions. Despite significant advances in optimal control theory and algorithms, existing methods face fundamental challenges in simultaneously meeting these requirements. This chapter addresses these challenges by proposing a novel algorithm (PHLAME¹²) that enhances computational efficiency while maintaining dynamic feasibility, leveraging recent advances in pseudospectral methods and spatial vector algebra applied to the Affine Geometric Heat Flow (AGHF) Partial Differential Equation (PDE).

¹Published in IEEE Robotics and Automation Letters, 10(4):4148–4155, 2025

²Code found at <https://github.com/roahmlab/PHLAME>

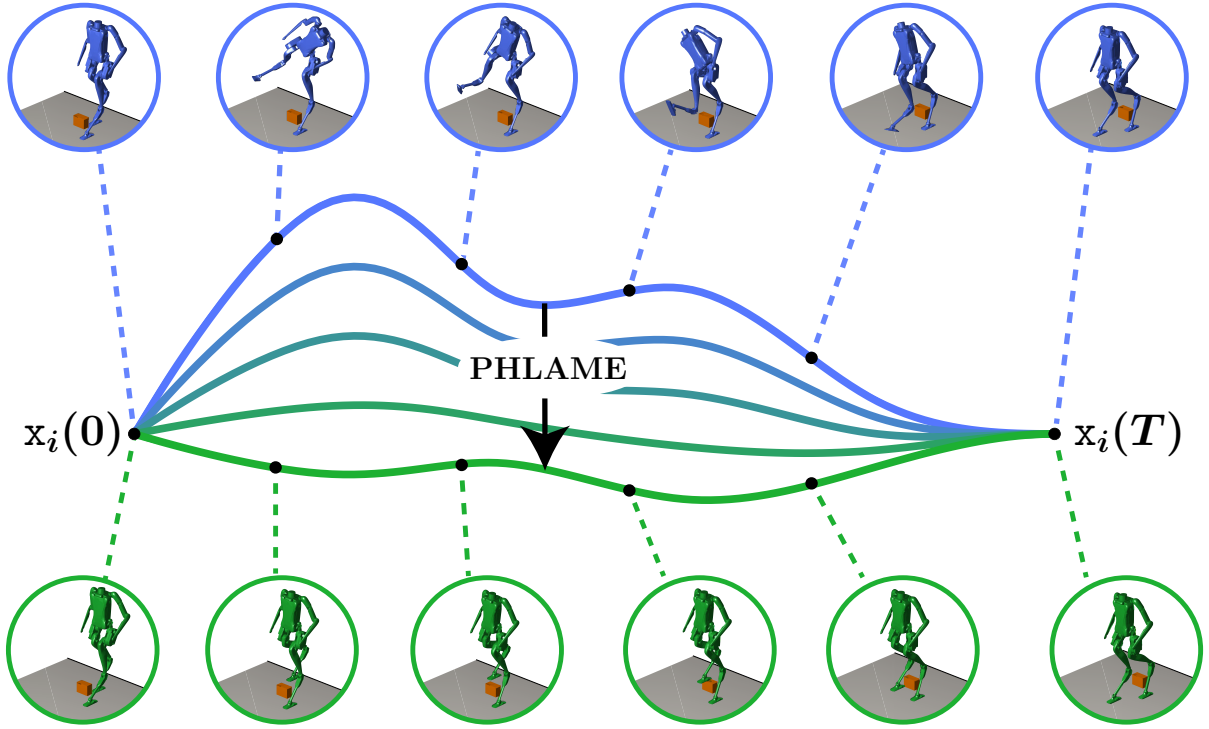


Figure 2.1: PHLAME works by first taking in some initial guess of a trajectory (trajectory of i th state shown in **dark blue**) which does not have to be dynamically feasible and evolves it into some dynamically feasible final trajectory **dark green**). Both trajectories start and end at $\mathbf{x}(0) = \mathbf{x}_0$ and $\mathbf{x}(T) = \mathbf{x}_f$ respectively. Notice that at the initial trajectory Digit (a high dimensional humanoid robot) has a dynamically infeasible set of configurations during it’s stepping trajectory and that at the end of the PHLAME solve that trajectory is made into a dynamically feasible one where Digit is able to step over the box.

2.1.1 Related Work

Existing optimal control approaches have made significant theoretical and algorithmic advances, yet still struggle to simultaneously meet all these requirements. To understand these challenges, we examine the two primary approaches to optimal control: Dynamic Programming (DP) methods and Variational methods.

Dynamic programming methods [9] leverage Bellman’s Principle of Optimality to compute optimal value functions and policies. Global approaches that solve the Hamilton-Jacobi-Bellman (HJB) equation provide optimality guarantees regardless of initialization but face fundamental computational barriers. In continuous time, these methods require discretizing the entire state space, meaning that as the state dimension increases, the number of grid points this nonlinear PDE must be evaluated at scales exponentially. This makes these methods computationally intractable beyond 5-dimensional systems and increasingly un-

stable numerically as dimensions grow. To address these limitations, Differential Dynamic Programming (DDP) variants such as Crocoddyl [79] apply dynamic programming principles locally along a trajectory, achieving faster convergence through iterative forward-backward passes. While more computationally tractable than global methods, DDP approaches can struggle with convergence when initialized far from local optima and face challenges incorporating inequality constraints, though recent work like Aligator [51] has begun addressing these limitations.

Variational methods, which derive necessary conditions for optimality using calculus of variations, offer an alternative approach. Direct methods in this category discretize the continuous optimal control problem into a nonlinear program (NLP). For example, direct collocation methods like C-FROST [43] and TROPIC [34] approximate trajectories using either linear interpolation, which is used in schemes such as trapezoidal collocation, or polynomial basis functions, which are employed in Hermite-Simpson collocation. While these methods effectively handle complex constraints [97, 98], they require significant computation time for high-dimensional systems, often taking tens of minutes to converge for high dimensional systems. Moreover, the quality of their solutions can be sensitive to both discretization choices and initial trajectory guesses. Another direct method, RAPTOR [124], parameterizes trajectories using Bézier curves, which allows it to solve optimization problems in a matter of seconds and makes it more robust to initial guesses. However, RAPTOR does not explicitly enforce dynamic constraints, and as a result, requires the robotic system to be fully actuated.

Indirect variational methods like Pontryagin’s Maximum Principle (PMP) [67] provide elegant theoretical solutions but exhibit high sensitivity to initial guesses of co-states, making them challenging to apply to complex robotic systems where good initialization is difficult to obtain.

This chapter presents a novel algorithm that addresses these limitations through an alternative PDE-based formulation called the Affine Geometric Heat Flow (AGHF). First introduced in [72], the AGHF poses trajectory generation as the solution to a PDE that evolves an initial trajectory that may not be dynamically feasible into a final trajectory that is dynamically feasible while minimizing control input magnitudes. Unlike traditional PDE-based optimal control methods like HJB whose solution domain scales with the state space dimension, the AGHF solution has a two-dimensional domain regardless of system dimension. As a result, the AGHF PDE offers a significant advancement in computational speed, without compromising the dynamic feasibility of motion planning and is also able to incorporate path constraints. Because the AGHF solution has a two-dimensional domain, it is usually solved by using the Method of Lines (MOL) [93]. The MOL begins by discretizing

the domain of the solution of AGHF along one dimension to generate a set of nodes. Then at each node, it represents the PDE as if it is an Ordinary Differential Equation (ODE). This system of ODEs can then be solved by using well understood numerical ODE solvers.

The nodes in the MOL are usually chosen in an evenly spaced fashion, and the solution quality gets better as more nodes are used. The number of evaluations of the PDE function scales linearly with the number of nodes so having a fine grid requires many evaluations of the AGHF to compute a solution. This issue is further exacerbated for high dimensional systems because evaluating the AGHF requires evaluating the dynamics and derivatives of the dynamics of the system whose trajectory is being optimized. This has made applying the AGHF PDE to perform trajectory optimization untenable for high dimensional systems [72].

To address these challenges, this chapter proposes PHLAME which applies pseudospectral collocation in conjunction with spatial vector algebra to rapidly solve the AGHF PDE.

2.1.2 Chapter Contributions

The main contributions of this work are four-fold: First, we propose a pseudospectral method that reduces the number of AGHF evaluations and nodes when compared to the classical MOL, which allows PHLAME to scale up to high dimensional robotic systems (Section 2.4.2). Second, we provide an analytical expression for the AGHF in terms of the rigid body dynamics equation and an algorithm to rapidly evaluate this analytical AGHF expression using spatial vector algebra based rigid body dynamics algorithms (Sections 2.4.1 2.4.4.1). Third, we provide an analytical expression for the jacobian of the AGHF and an algorithm to rapidly compute it using spatial vector algebra based rigid body dynamics algorithms (Sections 2.4.3 2.4.4.2). Finally, this chapter demonstrates the performance of PHLAME for trajectory optimization for a number of different dynamical systems in the presence of obstacles and without obstacles and illustrates its performance when compared to a variety of state of the art methods (Section 2.6).

The remainder of the chapter is arranged as follows: Section 2.2 presents the background and introduces the relevant notation for the chapter. Section 2.3 introduces the AGHF and discusses the underlying theory associated with the AGHF. Section 2.5 details how to incorporate constraints into the AGHF to enable actions like obstacle avoidance.

2.2 Preliminaries

This section introduces the notation used throughout this chapter. This paper is focused on performing trajectory optimization for robot systems whose dynamics can be written as follows:

$$H(q(t))\ddot{q}(t) + C(q(t), \dot{q}(t)) = Bu(t), \quad (2.1)$$

where $q(t) \in \mathbb{R}^N$ is the configuration of the robot at time t , $u(t) \in \mathbb{R}^m$ is the input applied to the robot at time t , $H(q(t))$ is the mass matrix, $C(q(t), \dot{q}(t))$ is the grouped Coriolis and gravity term and B is the actuation matrix. For convenience, let $\mathbf{x}(t)$ correspond to the vector of $q(t)$ and $\dot{q}(t)$. To be consistent with the notation in the rest of the paper we refer to the first N and last N components of $\mathbf{x}(t)$ as $\mathbf{x}_{P1}(t)$ and $\mathbf{x}_{P2}(t)$, respectively. Additionally, let $\mathbf{0}$ be a $N \times 1$ vector of zeros. Using these definitions, we can represent the dynamics of the robot (2.1) as a control affine system:

$$\dot{\mathbf{x}}(t) = F_d(\mathbf{x}(t)) + F(\mathbf{x}(t))u(t), \quad (2.2)$$

where

$$F_d(\mathbf{x}(t)) = \begin{bmatrix} \mathbf{x}_{P2}(t) \\ -H^{-1}(\mathbf{x}_{P1}(t))C(\mathbf{x}_{P1}(t), \mathbf{x}_{P2}(t)) \end{bmatrix} \quad (2.3)$$

$$F(\mathbf{x}(t)) = \begin{bmatrix} 0_{N \times m} \\ H^{-1}(\mathbf{x}_{P1}(t))B \end{bmatrix} \quad (2.4)$$

For convenience, we assume without any loss of generality that we are interested in the evolution of the system for $t \in [0, T]$.

To ensure the convergence of the AGHF PDE, we make the following assumption on the differentiability and smoothness of the system dynamics and existence of a feasible solution:

Assumption 1. *Both F_d and F are C^2 , Lipschitz continuous, and F has constant rank almost everywhere in $\mathbb{R}^n$. Additionally, we assume the existence of a feasible solution to the motion planning problem for the system.*

Note that the dynamics of rigid body robotic systems are smooth and Lipschitz continuous when their domain is restricted to a compact set.

The objective of this paper is to develop an algorithm to construct a trajectory beginning from some user-specified initial condition, $\mathbf{x}_0$, and ending in some user-specified terminal condition, $\mathbf{x}_f$, while avoiding obstacles and satisfying the dynamics in (2.2) for all $t \in [0, T]$ while minimizing the square control. If we let the zero superlevel set of a function g represent

the inequality constraints, then one can formulate the trajectory design problem as the solution to the following optimization problem:

$$\begin{aligned}
& \inf_{u \in L^2} \int_0^T \|u(t)\|_2^2 dt && \text{(OCP)} \\
& \text{s.t. } \dot{\mathbf{x}}(t) = F_d(\mathbf{x}(t)) + F(\mathbf{x}(t))u(t), && \forall t \in [0, T], \\
& g(\mathbf{x}(t)) \leq 0 && \forall t \in [0, T], \\
& \mathbf{x}(0) = \mathbf{x}_0, \\
& \mathbf{x}(T) = \mathbf{x}_f,
\end{aligned}$$

where L^2 denotes the space of square integrable functions. Note if a particular $\mathbf{x}(\cdot)$ satisfies each of the constraints in the (OCP), then we call $\mathbf{x}(\cdot)$ a feasible trajectory to (OCP). To numerically solve this optimization problem, the methods discussed in Section 2.1 are typically used. As mentioned in Section 2.1, these methods either have high computational costs, are highly sensitive to initial guesses, or may have difficulty dealing with non-smooth elements like obstacles or non-convex constraints. The proposed approach aims to address these challenges by enabling the rapid generation of trajectories for high-dimensional systems while incorporating multiple constraints using the Affine Geometric Heat Flow Partial Differential Equation to solve (OCP).

2.3 The Affine Geometric Heat Flow (AGHF) Partial Differential Equation

The Affine Geometric Heat Flow (AGHF) Partial Differential Equation (PDE) is a parabolic PDE that attempts to solve (OCP). At a high level, the AGHF equation works by deforming an initial trajectory that begins from $\mathbf{x}_0$ and ends at some final state $\mathbf{x}_f$ into a final trajectory that begins from $\mathbf{x}_0$ and ends at $\mathbf{x}_f$. The AGHF deforms that initial trajectory, which can be any trajectory including one that does not satisfy the dynamics, into a dynamically feasible final trajectory that minimizes some user-specified cost. This section summarizes the background knowledge and theory of the AGHF. A more detailed treatment of the subjects discussed here can be found in [72, 71, 73]. Note throughout this section, we assume that there are no inequality constraints in (OCP). The inequality constraint case is considered in Section 2.5.

2.3.1 Homotopies and Extracting Control Inputs

To describe the evolution of trajectories by the AGHF PDE, we begin by defining a homotopy: $x : [0, T] \times [0, s_{max}] \rightarrow \mathbb{R}^{2N}$, that is twice differentiable with respect to its first argument and differentiable with respect to its second argument. For convenience, we denote $x(t, s)$ by $x_s(t)$ and we denote $\frac{\partial x}{\partial t}(t, s)$ by $\dot{x}(t, s)$ or $\dot{x}_s(t)$. Next, define the *Lagrangian* as:

$$L(x_s(t), \dot{x}_s(t)) = (\dot{x}_s(t) - F_d(x_s(t)))^T G(x_s(t)) (\dot{x}_s(t) - F_d(x_s(t))) \quad (2.5)$$

where $G : \mathbb{R}^{2N} \rightarrow \mathbb{R}^{2N \times 2N}$ is a user-specified matrix. Similar to in Section 2.2, for succinctness we refer to the first N and last N components of x_s as x_{P1} and x_{P2} , respectively. Additionally, let the first and second time derivatives of these states be defined as $\dot{x}_{P1}$, $\dot{x}_{P2}$ and $\ddot{x}_{P1}$, $\ddot{x}_{P2}$ respectively. In Section 2.3.2, we describe how to select G to ensure that the AGHF minimizes the squared control effort as in the cost function in (OCP). Finally, we define the *Action Functional*:

$$\mathcal{A}(x_s) = \int_0^T L(x_s(t), \dot{x}_s(t)) dt. \quad (2.6)$$

Using these definitions, we can write down the AGHF PDE:

Definition 2. *The Affine Geometric Heat Flow is a parabolic partial differential equation that is defined as:*

$$\frac{\partial x}{\partial s}(t, s) = G^{-1}(x(t, s)) \left(\frac{d}{dt} \frac{\partial L}{\partial \dot{x}_s}(x_s(t), \dot{x}_s(t)) - \frac{\partial L}{\partial x_s}(x_s(t), \dot{x}_s(t)) \right) \quad (2.7)$$

with the following boundary conditions:

$$x_s(0) = \mathbf{x}_0, \quad \forall s \in [0, s_{max}] \quad (2.8)$$

$$x_s(T) = \mathbf{x}_f, \quad \forall s \in [0, s_{max}]. \quad (2.9)$$

When solving the AGHF PDE, one begins by specifying an initial curve $x_{init} : [0, T] \rightarrow \mathbb{R}^{2N}$ and setting it such that $x_0 = x_{init}$. As the AGHF PDE evolves forward in s , one can prove that the action functional is minimized. In addition, if during that evolution the AGHF converges to a curve where the right hand side of the AGHF PDE is equal to 0, then one has found a curve that extremizes the action functional. Such a curve is called a *steady state solution*. We formalize these observations in the following lemma that was originally proved in [72, Lemma 1] and which we repeat here for convenience:

Lemma 3. *Let x satisfy the AGHF PDE. Then, $\frac{d\mathcal{A}(x_s)}{ds} \leq 0$ for all s . In addition, if the right hand side of the AGHF PDE when evaluated at x_{s^*} is equal to 0 for some $s^* \in [0, s_{max})$, then $\frac{d\mathcal{A}(x_{s^*})}{ds} = 0$.*

2.3.2 Ensuring $\mathcal{A}$ Coincides with the Control Input by Designing G

To ensure that the action functional being minimized coincides with minimizing the square of the control input as in (OCP), we must design G carefully. We do this by applying the following lemma that was originally proven in [72, Theorem 1]:

Lemma 4. *Suppose (OCP) is feasible and let G be defined as follows:*

$$G(x_s(t)) = (\bar{F}(x_s(t))^{-1})^T K \bar{F}(x_s(t))^{-1} \quad (2.10)$$

where

$$K = \begin{bmatrix} kI_{N \times N} & 0_{N \times N} \\ 0_{N \times N} & I_{N \times N} \end{bmatrix} \in \mathbb{R}^{2N \times 2N} \quad (2.11)$$

for $k > 0$ and

$$\bar{F}(x_s(t)) = \begin{bmatrix} F_c(x_s(t)) & F(x_s(t)) \end{bmatrix} \in \mathbb{R}^{2N \times 2N}, \quad (2.12)$$

where $F_c \in \mathbb{R}^{2N \times (2N-m)}$ is some differentiable in x matrix such that $\bar{F}$ is invertible for all x in $\mathbb{R}^{2N}$. Note that such an F_c can be obtained using the Gram-Schmidt procedure. Let $u_s : [0, T] \rightarrow \mathbb{R}^m$ be the extracted control inputs at some $s \in [0, s_{max}]$ given by:

$$u_s(t) = \begin{bmatrix} 0_{N \times N} & I_{N \times N} \end{bmatrix} \bar{F}(x_s(t))^{-1} (\dot{x}_s(t) - F_d(x_s(t))). \quad (2.13)$$

Then

$$\mathcal{A}(x_s) = \int_0^T k \|\dot{x}_{P1} - x_{P2}\|_2^2 + \|u_s(t)\|_2^2 dt. \quad (2.14)$$

In short, at each s , the Action Functional with G as described by Lemma 4 corresponds to the squared control input of the trajectory x_s plus the error between the velocity states (x_{P2}) and the derivative of the position states ($\dot{x}_{P1}$). For sufficiently large k , this penalizes errors in the dynamics, reducing them as the trajectory evolves. For feasible trajectories of (OCP) that satisfy the dynamics, this error is zero, and the Action Functional with G corresponds solely to the squared control input generating that feasible trajectory.

As a result, if x_s was a feasible trajectory of (OCP) for each s , then Lemmas 3 and 4 would ensure that the AGHF was minimizing the square of the control input during its evolution.

Though we do not describe it here, [72, Theorem 1] under Assumption 1 proves that for sufficiently large k and s_{max} , the control extracted from the solution to the AGHF PDE can be used to generate a trajectory that is arbitrarily close to a feasible trajectory of (OCP). In fact [72, Theorem 1] proves an explicit bound on how close the trajectory generated by using (2.13) is to a feasible trajectory of (OCP).

2.4 Solving the AGHF Rapidly for High Dimensional Systems

The computationally intensive part of the AGHF method is solving (2.7), which is a parabolic PDE. In contrast to traditional PDEs used for optimal control (e.g., the Hamilton-Jacobi-Bellman PDE), the AGHF PDE has a complexity that scales polynomially with increasing state dimension rather than exponentially. The favorable scaling properties of the AGHF are owed to the fact that the domain of (2.7) is always **two-dimensional** and the dimension of the range of the function scales **linearly** with the state dimension. However, evolving the AGHF quickly demands being able to evaluate the right hand side of (2.7) rapidly. This can be difficult for high dimensional systems as the system dynamics and its derivatives must be evaluated each time the AGHF is called. If the AGHF must be evaluated at many time nodes, as in the classical MOL algorithm, then it can be even more challenging to construct a technique to rapidly solve the AGHF PDE.

This section describes how our method for solving the AGHF addresses these issues. Section 2.4.1 describes how to leverage spatial vector algebra to rapidly compute the right hand side of (2.7). Section 2.4.2 describes how to apply a pseudospectral MOL to reduce the number of time nodes that need to be considered to generate an accurate solution. Notably this pseudospectral MOL approach also allows us to accurately compute derivatives of time. This enables our method to avoid having to compute derivatives using finite difference, which dramatically reduces the number of function evaluations.

2.4.1 Computing AGHF PDE Partial Derivatives Analytically

This subsection derives analytical expressions that can be used to evaluate (2.7) in terms of the rigid body dynamics equation (2.1), and how to leverage spatial vector algebra to rapidly compute these expressions. We summarize the relevant results in the following theorem whose proof can be found in Appendix A.1.

Note, for succinctness, we have dropped the dependence on (t, s) for the following equations (i.e. $x(t, s)$ is denoted by x). Additionally, in similar fashion to the notation introduced

in Section 2.2, we denote the first N and last N components of x as x_{P1} and x_{P2} , respectively. Lastly, we also drop the dependence on the x terms for the dynamics functions (i.e., $H(x_{P1})$ is denoted by just H and so on)

Theorem 5. *Consider a system with dynamics as in (2.1). The AGHF PDE (2.7) using the G described in Lemma 4 can be written as follows:*

$$\frac{\partial x}{\partial s} = \Omega \left(x, \dot{x}, \ddot{x}, k \right) = \Omega_1 - (\Omega_2 - \Omega_3 + \Omega_4), \quad (2.15)$$

where

$$\Omega_1 = 2 \left[\begin{array}{c} \ddot{x}_{P1} - \dot{x}_{P2} \\ (H^T H)^{-1} ((\dot{H}^T H + H^T \dot{H}) \dot{x}_{P2} + H^T H \ddot{x}_{P2} + \dot{H}^T C + H^T \dot{C}) \end{array} \right] \quad (2.16)$$

$$\Omega_2 = \left[\begin{array}{c} -\frac{1}{k} I_{N \times N} \frac{\partial F D_0^T}{\partial x_{P1}} H^T (2C + 2H \dot{x}_{P2}) \\ -(H^T H)^{-1} \frac{\partial F D_0^T}{\partial x_{P2}} H^T (2C + 2H \dot{x}_{P2}) \end{array} \right] \quad (2.17)$$

$$\Omega_3 = \left[\begin{array}{c} \mathbf{0} \\ 2k(H^T H)^{-1}(\dot{x}_{P1} - x_{P2}) \end{array} \right] \quad (2.18)$$

$$\Omega_4 = \left[\begin{array}{c} 2\frac{1}{k} \left[\frac{\partial H}{\partial x_{P1}} (\dot{x}_{P2} - F D_0) \right]^T H (\dot{x}_{P2} - F D_0) \\ \mathbf{0} \end{array} \right], \quad (2.19)$$

and $F D_0 = -H^{-1}C$.

Note that $\frac{\partial H}{\partial x_{P1}}$ is a $N \times N \times N$ tensor, so computing Ω_4 requires a matrix-tensor multiplication. Algorithm 2 provides an efficient approach to avoid explicitly constructing the tensor $\frac{\partial H}{\partial x_{P1}}$ and performing matrix-tensor multiplication directly. It also details how to efficiently evaluate the analytical expressions in Theorem 5 using spatial vector algebra and rigid body dynamics algorithms. Section 2.4.4 discusses the computational efficiency of these algorithms and the overall AGHF evaluation.

2.4.2 Pseudospectral Method for Solving the AGHF

The goal of this section is to describe how to apply a pseudospectral MOL to solve the AGHF as in (2.15). Throughout the remainder of this section, we assume without loss of generality that we have scaled the time domain of the dynamics in (OCP) so that the initial time is -1 and the final time is 1 instead of 0 and T , respectively. This assumption is made with any loss of generality because one can shift and scale the dynamics in time to satisfy the assumption. Note, we make this assumption to simplify the presentation of the pseudospectral method that relies on Chebyshev polynomials whose domain is $[-1, 1]$.

At a high level, the pseudospectral MOL begins by representing the solution of the AGHF PDE as a linear combination of Chebyshev polynomials of $t \in [-1, 1]$ at each value of s . Taking inspiration from pseudospectral methods, we represent the Chebyshev function by its values at certain discrete points in t , which are called the collocation nodes. Notably, computing the values of the derivatives of the function at these same discrete nodes can be done by applying a matrix called the differentiation matrix. As a result, the AGHF PDE at each of the collocation nodes can be written down as a system of ordinary differential equations. Once the values of the solution are known at each of the collocation nodes at some final s , then one can apply Chebyshev interpolation to construct the steady state solution of the AGHF.

Recall in Section 2.1, we described how the regular method of lines works by discretizing the PDE in one dimension to generate a set of collocation nodes, and then approximating derivatives at these collocation nodes using finite difference. The benefits of applying a pseudospectral method of lines as opposed to the regular method of lines, lies in the ability for the pseudospectral method to represent the solution to the PDE as a polynomial, which simultaneously enables it to give high accuracy derivatives at each of the nodes. In particular, by using a polynomial basis set (i.e., the Chebyshev Polynomials), the number of nodes to achieve a high accuracy solution is significantly less than the number of nodes required by the classical MOL algorithm [10, Chapter 2, Chapter 4].

The remainder of this subsection describes how to perform the transformation from the AGHF PDE to a system of ordinary differential equations. To begin, let $p \in \mathbb{N}$ and define the *Chebyshev nodes* as:

$$t_i = -\cos\left(\frac{\pi i}{p}\right), \quad (2.20)$$

for each $i \in \{0, \dots, p\}$. If we fix a particular $p \in \mathbb{N}$ and have the values of a continuously differentiable function at all of the Chebyshev nodes, then we can compute the approximate the values of the derivative of that function at the Chebyshev nodes by using the differentiation matrix, $D : \mathbb{R}^{p+1} \rightarrow \mathbb{R}^{p+1}$ [107, (21.2)]. Note that one that just needs to multiply this

matrix by the vector of function values at the Chebyshev nodes to compute the approximate value of the derivative of the function at the Chebyshev nodes. In fact, the differentiation matrix of size $p+1$ generates the exact derivative at the Chebyshev nodes for polynomials of degree p or less. Note that for suitably smooth functions one can compose this differentiation matrix to compute higher order derivatives (i.e., D^3 can be applied to compute the third derivative of a function at the Chebyshev nodes).

Next, we transform the AGHF PDE (2.15) into a system of ODEs. To do this, fix $p \in \mathbb{N}$ and for each s denote the value of the solution at a particular Chebyshev node t_i as $\xi_i(s) = x^T(t_i, s)$ and let

$$\xi(s) = \begin{bmatrix} \xi_0(s) \\ \vdots \\ \xi_p(s) \end{bmatrix} \in \mathbb{R}^{(p+1) \times 2N}. \quad (2.21)$$

By using the definition of the differentiation matrix, notice that the i th row of $D\xi(s)$ is an approximation of $\frac{\partial x^T}{\partial t}(t_i, s)$. For convenience let us denote the i th row of $D\xi(s)$ as $[D\xi]_i(s)$.

With these definitions, we can write down the AGHF PDE (2.15) at each of the nodes as system of ODEs:

$$\frac{d\xi}{ds}(s) = \begin{bmatrix} \frac{d\xi_0}{ds}(s) \\ \vdots \\ \frac{d\xi_p}{ds}(s) \end{bmatrix} \in \mathbb{R}^{(p+1) \times 2N} \quad (2.22)$$

where

$$\frac{d\xi_i}{ds}(s) = \Omega(\xi_i^T(s), [D\xi]_i^T(s), [D^2\xi]_i^T(s), k) \quad (2.23)$$

for each $i \in \{0, \dots, p\}$. For notational convenience, we have abused notation and have not transposed $\frac{d\xi_i}{ds}(s)$ on the left hand side of the previous equation.

This system of ordinary differential equations can be simultaneously solved by using an appropriate differential equation solver. Though we do not prove it here, one can show that under certain regularity assumptions regarding the numerical method used to solve the ODEs that the solution computed by the pseudospectral method of lines converges to the true solution of the PDE [10, Chapter 9 and 12].

Finally, note for each $p \in \mathbb{N}$, t_0 and t_p correspond to -1 and 1 , respectively. As a result, one can ensure that the initial and final state of the AGHF solution for all s satisfies the boundary conditions by setting $\xi_0(s) = x_0$ and $\xi_n(s) = x_f$. Note this allows us to reduce the number of system of ODEs by $4N$.

The PHLAME algorithm is summarized in Algorithm 1. It first requires one to specify some initial curve x_0 with initial and terminal points as $\mathbf{x}_0$ and $\mathbf{x}_f$ respectively. Along with the initial curve one must specify the number of pseudospectral nodes p , the penalty term

Algorithm 1 PHLAME

Require: $x_0 : [0, T] \rightarrow \mathbb{R}^{2N}$ s.t. $x_0(-1) = \mathbf{x}_0$, $x_0(1) = \mathbf{x}_f$, $p \in \mathbb{N}$, k and s_{max} .

- 1: $\xi_i^T(0) \leftarrow x_0(t_i)$ for $i \in \{1, \dots, p-1\}$ (2.20).
 - 2: **Compute** $\xi(s_{max})$ using an ODE Solver.
 - 3: **Compute** $u(t_i)$ using $\xi(s_{max})$ and (2.13).
-

k to be used in G , and the final s in the domain of the homotopy s_{max} . Algorithm 1 then sets the values of the initial pseudospectral nodes, $\xi(0)$, equal to the initial curve (Line 1). Then an ODE solver (e.g., Runge Kutta or Adams Bashforth Method [10]) can be used to simulate solution of the AGHF PDE at each of the collocation nodes (Line 2). To evaluate the dynamics within the ODE, one can apply Algorithm 2. Finally, one can extract the control input at each of the collocation nodes by using (2.13) (Line 3). Note that one could also perform Chebyshev interpolation to compute the control input for all $t \in [-1, 1]$ [107].

2.4.3 Computing the PHLAME Jacobian Partial Derivatives Analytically

To solve the system of ODEs in (2.22), we leverage a differential equation solver that uses an implicit method [87]. This implicit method requires the derivative of (2.23). Most ODE solvers that use implicit methods approximate the derivative numerically. This can reduce accuracy and increase the number of function evaluations, slowing down the process. To avoid this and further speed up PHLAME, we compute and provide the analytical Jacobian of the system of ODEs with respect to $\xi(s)$. Computing this Jacobian, requires one to compute the Jacobian of $\frac{d\xi_i}{ds}(s)$ with respect to ξ_i . We summarize the form of this Jacobian as a function of the first- and second-order derivatives of the rigid body dynamics in Theorem 6, whose proof can be found in Appendix A.2. To efficiently compute the required second-order derivatives, we leverage some of the algorithms highlighted in [96]. Algorithm 3 shows how we rapidly compute all the necessary terms to evaluate the Jacobian. Once again, for notational convenience we have abused notation and left out the transpose for $\frac{d\xi_i}{ds}(s)$.

Theorem 6. *Let $\xi_i(s) = x^T(t_i, s)$, $[D\xi]_i(s) = \dot{x}^T(t_i, s)$ and $[D^2\xi]_i(s) = \ddot{x}^T(t_i, s)$. Then the Jacobian of $\frac{d\xi_i}{ds}(s)$ with respect to $\xi_i(s)$, $J_{\Xi_i}(s)$ is given by:*

$$\begin{aligned} J_{\Xi_i} &= \frac{d\left(\frac{d\xi_i}{ds}(s)\right)}{d\xi_i(s)} = \frac{d\Omega(\xi_i^T(s), [D\xi]_i^T(s), [D^2\xi]_i^T(s), k)}{d\xi_i(s)} \\ &= \frac{d\Omega}{d\xi_i} \frac{d\xi_i}{d\xi_i} + \frac{d\Omega}{d[D\xi]_i} \frac{d[D\xi]_i}{d\xi_i} + \frac{d\Omega}{d[D^2\xi]_i} \frac{d[D^2\xi]_i}{d\xi_i} \end{aligned} \tag{2.24}$$

where

$$\begin{aligned} \frac{d\Omega}{d\xi_i} &= \frac{d\Omega_1}{dx} - \left(\frac{d\Omega_2}{dx} - \frac{d\Omega_3}{dx} + \frac{d\Omega_4}{dx} \right) \\ &= \frac{d\Omega}{d\xi_i} \left(\frac{\partial H}{\partial x_{P1}}, \frac{\partial^2 H}{\partial x_{P1}^2}, \frac{\partial \dot{H}}{\partial x_{P1}}, \frac{\partial FD_0}{\partial x_{P1}}, \frac{\partial FD_0}{\partial x_{P2}}, \frac{\partial C}{\partial x_{P1}}, \frac{\partial C}{\partial x_{P2}}, \frac{\partial^2 FD_0}{\partial x_{P1}^2}, \frac{\partial^2 FD_0}{\partial x_{P2}^2}, \frac{\partial^2 FD_0}{\partial x_{P1}\partial x_{P2}} \right) \end{aligned} \quad (2.25)$$

$$\begin{aligned} \frac{d\Omega}{d[D\xi]_i} &= \frac{d\Omega_1}{d\dot{x}} - \left(\frac{d\Omega_2}{d\dot{x}} - \frac{d\Omega_3}{d\dot{x}} + \frac{d\Omega_4}{d\dot{x}} \right) \\ &= \frac{d\Omega}{d[D\xi]_i} \left(H, C, \dot{H}, FD_0, \frac{\partial H}{\partial x_{P1}}, \frac{\partial FD_0}{\partial x_{P1}}, \frac{\partial FD_0}{\partial x_{P2}}, \frac{\partial C}{\partial x_{P1}}, \frac{\partial C}{\partial x_{P2}} \right) \end{aligned} \quad (2.26)$$

$$\frac{d\Omega}{d[D^2\xi]_i} = 2I_{2N \times 2N} \quad (2.27)$$

2.4.4 Rapidly Evaluating the AGHF (2.15) and its Jacobian (2.24)

This section presents several algorithms to rapidly evaluate the AGHF in (2.15) and its Jacobian (2.24) using spatial vector algebra based rigid body dynamics algorithms and discusses the computational efficiency of these algorithms.

2.4.4.1 Rapidly Evaluating the AGHF (2.15)

To efficiently evaluate the AGHF at each of the collocation nodes, one can use Theorem 5. Algorithm 2 shows how to leverage spatial vector algebra and some state-of-the-art dynamics algorithms [17] to rapidly compute the expressions introduced in Algorithm 2. By using recursive algorithms based on spatial vector algebra to compute the necessary dynamics terms in (2.15), this approach provides a substantial speedup over [72]. For example, algorithms like the Recursive Newton-Euler Algorithm (RNEA) achieve $\mathcal{O}(N)$ complexity for systems with N bodies, efficiently propagating forces and accelerations throughout the robot's kinematic chain and enabling faster, more scalable evaluation of the dynamics terms. This combination of recursive methods allows for a more rapid evaluation of the AGHF's right-hand side compared to the original approach in [72], especially for higher dimensional systems. Table 2.1 in Section 2.6, shows how well this algorithm scales with increasing system dimension compared to [72].

In Algorithm 2, we begin by computing H and $\dot{H}$ using a modified version of the Composite Rigid Body Algorithm (CRBA) (Line 1). This modified version leverages the chain rule

Algorithm 2 Leveraging Spatial Vector Algebra to Compute Ω (2.15)

Require: $x, \dot{x}, \ddot{x}, k$

- 1: $H, \dot{H} \leftarrow \text{CRBA_D}(x_{P1}, \dot{x}_{P1})$
 - 2: $C \leftarrow \text{RNEA}(x_{P1}, x_{P2}, \mathbf{0})$
 - 3: $\frac{\partial C}{\partial x_{P1}}, \frac{\partial C}{\partial x_{P2}} \leftarrow \text{RNEA_D}(x_{P1}, x_{P2}, \mathbf{0})$
 - 4: $\dot{C} \leftarrow \frac{\partial C}{\partial x_{P1}} \dot{x}_{P1} + \frac{\partial C}{\partial x_{P2}} \dot{x}_{P2}$
 - 5: $\frac{\partial FD_0}{\partial x_{P1}}, \frac{\partial FD_0}{\partial x_{P2}}, H^{-1}, FD_0 \leftarrow \text{ABA_D}(x_{P1}, x_{P2}, \mathbf{0})$
 - 6: **Compute** Ω_1 (2.16), Ω_2 (2.17) and Ω_3 (2.18)
 - 7: `model_gravity` $\leftarrow 0$
 - 8: $\frac{\partial H}{\partial x_{P1}}(\dot{x}_{P2} - FD_0) \leftarrow \text{RNEA_D}(x_{P1}, \mathbf{0}, \dot{x}_{P2} - FD_0)$
 - 9: $\omega_4 \leftarrow 2\frac{1}{k} \left[\frac{\partial H}{\partial x_{P1}}(\dot{x}_{P2} - FD_0) \right]^T H(\dot{x}_{P2} - FD_0)$
 - 10: $\Omega_4 \leftarrow [\omega_4^T \quad \mathbf{0}^T]^T$
 - 11: **Compute** Ω (2.15)
-

to compute the time derivatives of the various spatial quantities as we traverse the rigid body tree, yielding the time derivative of the mass matrix ($\dot{H}$). This modified version, we call `CRBA_D`, is a worst-case $\mathcal{O}(N^3)$ algorithm. We then use the Recursive Newton Euler Algorithm (`RNEA`) to rapidly compute C ($\mathcal{O}(N)$) (Line 2) and an extended version (`RNEA_D`) that computes `RNEA`'s derivatives with respect to $q, \dot{q}$ and $\ddot{q}$ to compute the derivatives of C with respect to x_{P1} and x_{P2} ($\mathcal{O}(N^2)$ worst-case) (Line 3). These are all used to compute $\dot{C}$ (Line 4).

Next, we use the algorithm introduced in [16] to compute multiple partial derivatives of the Forward Dynamics when $u = \mathbf{0}$ (Line 5) leveraging the Articulated Body Algorithm (`ABA`). We denote this as `ABA_D`. This is a worst-case $\mathcal{O}(N^3)$ algorithm [95]. Utilizing the earlier results, we compute Ω_1, Ω_2 and Ω_3 (Line 6). Next, we compute $\frac{\partial H}{\partial x_{P1}}(\dot{x}_{P2} - FD_0)$ efficiently by setting the gravity term used by our dynamics model (`model_gravity`) to zero (Line 7) and using `RNEA_D` with zero velocity and setting the acceleration to $(\dot{x}_{P2} - FD_0)$, which avoids explicitly computing the tensor $\frac{\partial H}{\partial x_{P1}}$ (Line 8). Lastly, we perform a matrix-vector multiplication to compute ω_4 (Line 9) and stack the vector to obtain Ω_4 (Line 10). With all the terms computed we can compute Ω using (2.15) (Line 11).

Combining these operations with the $\mathcal{O}(N^3)$ matrix-matrix multiplications needed to compute (2.15), results in a worst-case $\mathcal{O}(N^3)$ algorithm for computing the AGHF right-hand side. Figure 2.2 shows the mean and standard deviation of the computation time of (2.15) as the number of bodies (N) of the system increases from 2 (Double Pendulum) to 22

(Digit). While the worst-case computational complexity is $\mathcal{O}(N^3)$, the results from Figure 2.2 indicate that the algorithm scales more efficiently in practice. Specifically, the polynomial line of best fit lacks a significant N^3 term, suggesting that the computational time scales approximately quadratically with the number of bodies N within the observed range.

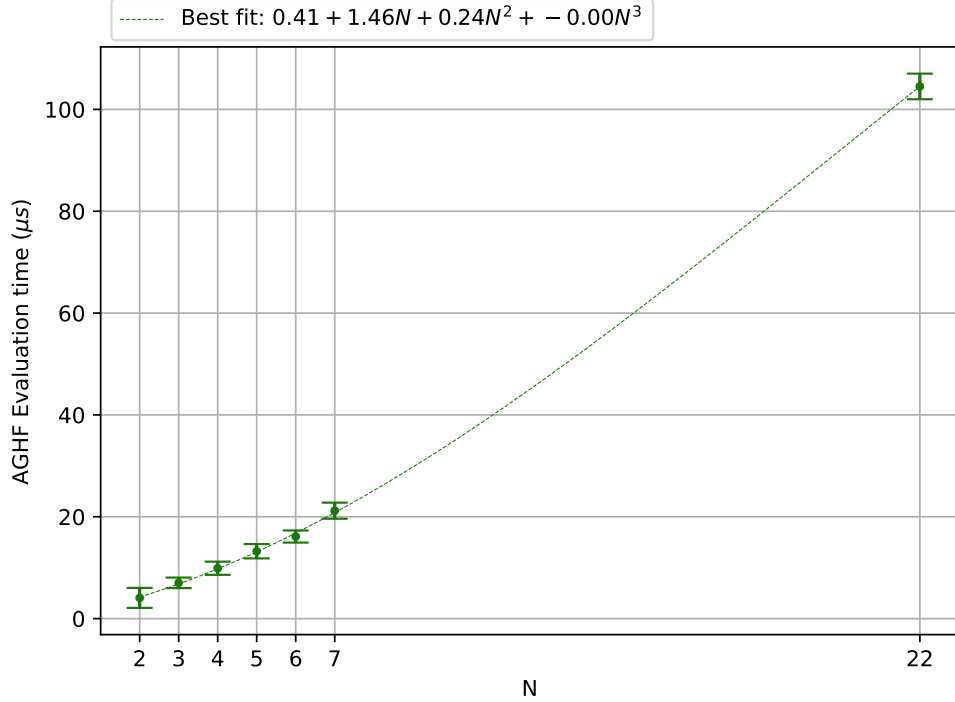


Figure 2.2: Scaling trend of the mean evaluation times (in μs) of the right-hand side of the AGHF using Algorithm 2 as the number of bodies (N) increases from 2 to 22. The systems with N between 2 and 6 are penduli systems (i.e., 2-link pendulum, 3-link pendulum, etc.). The $N = 7$ system is the Kinova Gen3 and the $N = 22$ system is the pinned Digit V3 biped. Each robot’s AGHF RHS was evaluated at a 1000 random robot configurations. Notice the polynomial line of best fit lacks a significant N^3 term, suggesting that the right-hand side computation time scales approximately quadratically with N in practice.

2.4.4.2 Rapidly Evaluating the AGHF Jacobian (2.24)

To efficiently evaluate the AGHF Jacobian at each of the collocation nodes, one can use Theorem 6. Algorithm 3 shows how to leverage spatial vector algebra and state-of-the-art dynamics algorithms [17] to rapidly compute this Jacobian in (2.24). We begin by first computing H , $\dot{H}$, $\frac{\partial H}{\partial x_{P1}}$ and $\frac{\partial^2 H}{\partial x_{P1}^2}$ (Line 1) using a modified **CRBA** algorithm where we compute $\dot{H}$ and the first and second derivatives of H with respect to x_{P1} using the chain rule. This modified version, we call **CRBA_2D**, is a worst-case $\mathcal{O}(N^4)$ algorithm. Similar to

Section 2.4.4.1, we use `ABA_D` (worst-case $\mathcal{O}(N^3)$ [95]) to compute the partial derivatives of the Forward Dynamics (Line 2). We then use `RNEA` to compute C ($\mathcal{O}(N)$) (Line 3) and `RNEA_D` to compute the derivatives of C (Line 4) once more ($\mathcal{O}(N^2)$ worst-case). Next we use these derivatives to compute $\dot{C}$ (Line 5). Next, the function `get_Hdot_D` computes $\frac{\partial \dot{H}}{\partial x_{P1}}$ by applying the chain rule to $\frac{\partial^2 H}{\partial x_{P1}^2}$ and $\dot{x}_{P1}$ (Line 6). We then use `RNEA_2D`, which computes the second derivatives of the Inverse Dynamics (ID), with the acceleration passed in as zero to get the second derivatives of C (Line 7). Next, `RNEA_2D` with the acceleration set to FD_0 (which is the acceleration of the system, with $u = 0$) allows us to compute the derivatives of the inverse dynamics (Line 8). The inverse dynamics derivatives computed using the acceleration from FD_0 are needed to rapidly compute the second derivatives of FD_0 using the algorithm proposed in [96], `ABA_2D` (Line 9). The function `get_Cdot_D`, similar to `get_Hdot_D`, computes the partial derivatives of $\dot{C}$ using the chain rule with the second derivatives of C and $\dot{x}_{P1}$ and $\dot{x}_{P2}$ (Line 10). With all these terms computed we then evaluate $J_{\Xi_i}(s)$ (2.24) (Line 11). Overall, combining these operations with the $\mathcal{O}(N^4)$ tensor-matrix multiplications needed to compute (2.24), results in an $\mathcal{O}(N^4)$ algorithm for computing $J_{\Xi_i}(s)$. Figure 2.3 shows the mean and standard deviation of the computation time of $J_{\Xi_i}(s)$ as the number of bodies (N) of the system increases from 2 (Double Pendulum) to 22 (Digit). While the worst-case computational complexity is $\mathcal{O}(N^4)$, the results from Figure 2.3 indicate that the algorithm scales more efficiently in practice. Specifically, the polynomial line of best fit lacks a significant N^4 term, suggesting that the Jacobian computational time scales approximately cubically with the number of bodies N within the observed range.

2.5 Incorporating constraints into the AGHF

This section explains how constraints are incorporated into the AGHF by adding them to the Lagrangian. The constraint terms are designed so that any violations increase the magnitude of the action functional. As we minimize the action functional, PHLAME naturally converges to solutions that satisfy the constraints. Below we discuss the form and properties of the added constraint term and show how it augments Lagrangian and the AGHF PDE. Note that similar to Theorems 5 and 6, in the subsequent subsections we denote $x(t, s)$ by x .

2.5.1 Constraint Lagrangian

We incorporate constraints into the AGHF by using a penalty term in the Lagrangian in a similar fashion to [32]. This penalizes the PDE when it evolves towards undesirable states. By adding the penalty term, our original Lagrangian from (2.5) is augmented in the following

Algorithm 3 Leveraging Spatial Vector Algebra to Compute $J_{\Xi_i}(s)$ (2.24)

Require: $x, \dot{x}, \ddot{x}, k$

- 1: $H, \dot{H}, \frac{\partial H}{\partial x_{P1}}, \frac{\partial^2 H}{\partial x_{P1}^2} \leftarrow \text{CRBA_2D}(x_{P1}, \dot{x}_{P1})$
 - 2: $\frac{\partial FD_0}{\partial x_{P1}}, \frac{\partial FD_0}{\partial x_{P2}}, H^{-1}, FD_0 \leftarrow \text{ABA_D}(x_{P1}, x_{P2}, \mathbf{0})$
 - 3: $C \leftarrow \text{RNEA}(x_{P1}, x_{P2}, \mathbf{0})$
 - 4: $\frac{\partial C}{\partial x_{P1}}, \frac{\partial C}{\partial x_{P2}} \leftarrow \text{RNEA_D}(x_{P1}, x_{P2}, \mathbf{0})$
 - 5: $\dot{C} \leftarrow \frac{\partial C}{\partial x_{P1}} \dot{x}_{P1} + \frac{\partial C}{\partial x_{P2}} \dot{x}_{P2}$
 - 6: $\frac{\partial \dot{H}}{\partial x_{P1}} \leftarrow \text{get_Hdot_D}(\dot{x}_{P1}, \frac{\partial^2 H}{\partial x_{P1}^2})$
 - 7: $\frac{\partial^2 C}{\partial x_{P1}^2}, \frac{\partial^2 C}{\partial x_{P2}^2}, \frac{\partial^2 C}{\partial x_{P1} \partial x_{P2}} \leftarrow \text{RNEA_2D}(x_{P1}, x_{P2}, \mathbf{0})$
 - 8: $\frac{\partial^2 ID}{\partial x_{P1}^2}, \frac{\partial^2 ID}{\partial x_{P2}^2}, \frac{\partial^2 ID}{\partial x_{P1} \partial x_{P2}} \leftarrow \text{RNEA_2D}(x_{P1}, x_{P2}, FD_0)$
 - 9: $\frac{\partial^2 FD_0}{\partial x_{P1}^2}, \frac{\partial^2 FD_0}{\partial x_{P2}^2}, \frac{\partial^2 FD_0}{\partial x_{P1} \partial x_{P2}} \leftarrow \text{ABA_2D}(\frac{\partial FD_0}{\partial x_{P1}}, \frac{\partial FD_0}{\partial x_{P2}}, H^{-1}, \frac{\partial H}{\partial x_{P1}}, \frac{\partial^2 ID}{\partial x_{P1}^2}, \frac{\partial^2 ID}{\partial x_{P2}^2}, \frac{\partial^2 ID}{\partial x_{P1} \partial x_{P2}})$
 - 10: $\frac{\partial \dot{C}}{\partial x_{P1}}, \frac{\partial \dot{C}}{\partial x_{P2}} \leftarrow \text{get_Cdot_D}(\dot{x}_{P1}, \dot{x}_{P2}, \frac{\partial^2 C}{\partial x_{P1}^2}, \frac{\partial^2 C}{\partial x_{P2}^2}, \frac{\partial^2 C}{\partial x_{P1} \partial x_{P2}}, \frac{\partial^2 C}{\partial x_{P2} \partial x_{P1}})$
 - 11: **Compute** $J_{\Xi_i}(s)$ (2.24)
-

way:

Definition 7. Let k_{cons} be some large positive integer that penalizes constraint violation, and let $g_j(x)$ be the j -th inequality constraint evaluated at x . Finally, let L be the Lagrangian from (2.5) with additional terms to enforce the constraints for all $j \in \mathcal{J}$. L_{cons} is given by:

$$L_{\text{cons}}(x, \dot{x}, g_j(x)) = L(x, \dot{x}) + \sum_{j \in \mathcal{J}} b(g_j(x)) \quad (2.28)$$

where

$$b(g_j(x)) = k_{\text{cons}} \cdot (g_j(x))^2 \cdot S(g_j(x)), \quad (2.29)$$

where $S : \mathbb{R} \rightarrow \mathbb{R}$ is defined as follows:

1. $S : \mathbb{R} \rightarrow \mathbb{R}$ is a positive, differentiable function,
2. $S(g_j(x)) = 0$ when $g_j(x) \leq 0$ and
3. $S(g_j(x)) = 1$ when $g_j(x) > 0$.

An example of an S satisfying this definition is:

$$S(g_j(x)) = \frac{1}{2} + \frac{1}{2} \tanh(c_{\text{cons}} \cdot g_j(x)) \quad (2.30)$$

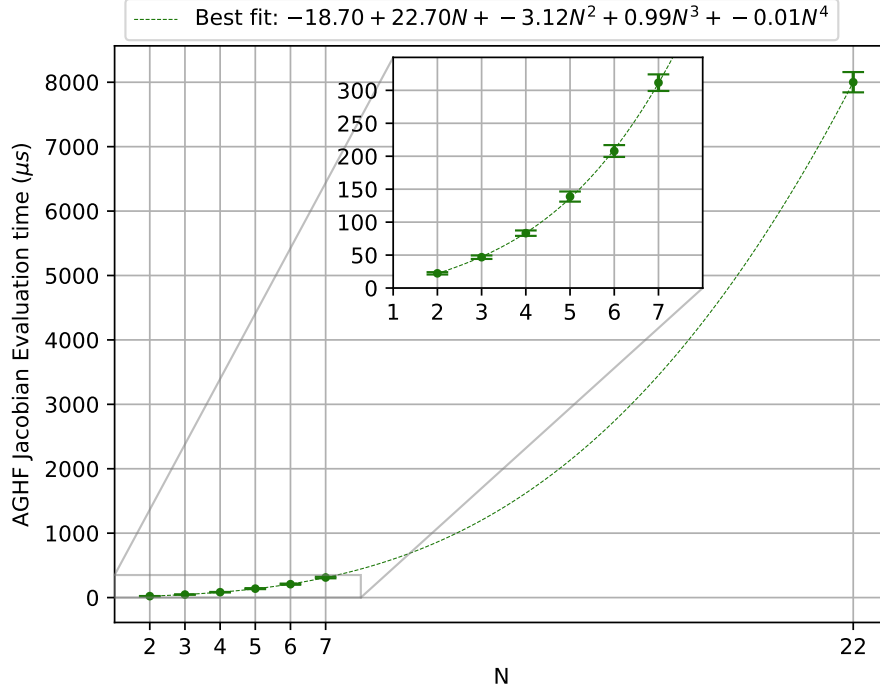


Figure 2.3: Scaling trend of the mean evaluation times (in μs) of the AGHF Jacobian using Algorithm 3 as the number of bodies (N) increases from 2 to 22. The systems with N between 2 and 6 are penduli systems (i.e., 2-link pendulum, 3-link pendulum, etc.). The $N = 7$ system is the Kinova Gen3 and the $N = 22$ system is the pinned Digit V3 biped. Each robot’s AGHF Jacobian was evaluated at a 1000 random robot configurations. Notice the polynomial line of best fit lacks a significant N^4 term, suggesting that the Jacobian computation time scales approximately cubically with N in practice.

where c_{cons} is a hyper-parameter that determines how fast $S(g_j(x))$ transitions from 0 to 1, once the constraint is violated. We introduce the form of the AGHF with L_{cons} in the following definition:

Definition 8. Let L_{cons} be used as the Lagrangian to construct the AGHF PDE. Then the constrained AGHF PDE is given by:

$$\frac{\partial x}{\partial s} = G^{-1}(x) \left(\frac{d}{dt} \frac{\partial L}{\partial \dot{x}} - \frac{\partial L}{\partial x} - \sum_{j \in \mathcal{J}} \frac{\partial b(g_j(x_s(t), \dot{x}_s(t)))}{\partial x_s(t)} \right) \quad (2.31)$$

The proof of convergence of the constrained AGHF is given in the proof of [31, Lemma 4.1].

2.6 Experiments and Results

This section evaluates the speed and performance of PHLAME on a variety of scenarios and robot platforms and compares PHLAME against the state of the art trajectory optimization algorithms. We begin by explaining the experimental setup. Next, we discuss various implementation details of the comparison methods and conclude the section by summarizing the different evaluations.

2.6.1 Experimental Setup

2.6.1.1 Robot Platforms and Comparison Methods

We perform evaluations on the following robot platforms: the 1 to 5 link pendulums in two dimensions, the Kinova Gen3 7DOF arm, and a pinned version of Digit V3 by Agility Robotics with the closed kinematic chains removed. Note we have removed the kinematic chain because a version of the RNEA algorithm that incorporates such a chain has not been implemented within the spatial vector algebra and rigid body dynamics package that we utilize – Pinocchio [17]. We compare the performance of PHLAME to the original AGHF [72], Crocoddyl [79], and Aligator [51]. Note we refer to Crocoddyl and Aligator as the trajectory optimization methods as in contrast to the PHLAME and the original AGHF approach they rely on optimization to synthesize a trajectory. We choose to compare to these methods in particular because these recent DDP methods have been shown to outperform the collocation-based methods [117] in terms of computation time and the other indirect and dynamic programming methods mentioned in Section 2.1 do not scale well to high-dimensional robots. Details on the scenarios used for the comparisons ($\mathbf{x}_0$, $\mathbf{x}_f$, obstacles description etc.) are available online at https://github.com/roahmlab/PHLAME_comparisons.

2.6.1.2 Determining Success or Failure

Evaluating whether a solution generated by a numerical method is satisfactory is non-trivial. In particular, each of these numerical methods compute an open loop input, $u^* : [0, T] \rightarrow \mathbb{R}^m$ and associated trajectory of the system, $x^* : [0, T] \rightarrow \mathbb{R}^{2N}$; however, it is unclear whether these solutions can actually be successfully followed by the robot. For instance, when one forward integrates the system using the control input synthesized by one of the evaluated algorithms, one may find that it violates the constraints. One could check constraint satisfaction by applying this open-loop control directly into the high dimensional robotic system and checking whether constraints are satisfied; however, this is also impractical because small numerical integration errors can compound significantly over time, causing the actual tra-

jectory to deviate substantially from the planned path. As a result, to fairly assess whether constraints are satisfied, we instead integrate forward using the following feedback controller:

$$u_{fb}(t) = u^*(t) + k_p(x_{P1}^*(t) - q(t)) + k_v(x_{P2}^*(t) - \dot{q}(t)), \quad (2.32)$$

where k_p and k_v are parameters that we fix in each experiment and q and $\dot{q}$ correspond to the position and velocity of the robot whose dynamics we are integrating forward using (2.32). Note that when we report the control effort for a particular experiment and algorithm, we compute the 2-norm of u_{fb} .

In the case of an experiment without obstacles, a solution obtained by a method is successful if the error between the final state of the forward integrated solution and $\mathbf{x}_f$ is less than some ϵ in the infinity norm. Note ϵ is a constant threshold that is set to 0.05 for all experiments. In the case of experiments with obstacles, a trial is successful if the previous condition is met and additionally, the joint position of the forward integrated solution at a time resolution of $\delta t = 10^{-2}$ s is not inside any of the obstacles at each joint frame.

2.6.1.3 Solver Setup

All experiments were run on a Ubuntu 22.04 machine with an Intel Xeon Platinum 8170M @ 208x 3.7GHz CPU. Each of the evaluated numerical methods requires an initial and final desired state and an initial guess for the initial trajectory x_0 along with the selection of solver specific parameters. Note the trajectory optimization methods also require an initial control input $u_0 : [0, T] \rightarrow \mathbb{R}^m$. Throughout this section, the term experiment refers to a tuple of experimental parameters consisting of the robotic platform, $\mathbf{x}_0$, $\mathbf{x}_f$, and a set of obstacles (if present).

To perform a fair comparison, we ran a grid search over the parameter space to obtain the best possible solver parameters. Each grid search was ran in parallel with a timeout per parameter set. The timeout for the systems with ≤ 5 DOF was 3 minutes and for the rest, 5 minutes. Then, we re-ran the experiments sequentially with the best solver specific parameters per method to obtain timings. The best solver specific parameter is defined as the one that yields the lowest solve time while producing a success for the experiment as defined above.

We next describe the parameters that make up the grid search for each method. Note that these parameters and how they are varied are summarized in Appendix A.3.

As part of the grid search for Crocodyl and Aligator, we considered different types of initial guesses to pass to the optimizer based on the examples given in their publicly available code. For Aligator, we considered the following initial guesses: (1) “Zeros,” which

corresponds to an initial guess that is zero for all time for x_0 and u_0 ; (2) “Line and RNEA,” which corresponds to an initial guess where x_0 is a line connecting $\mathbf{x}_0$ and $\mathbf{x}_f$ and u_0 is equal to applying RNEA using x_0 and $\dot{x}_0$ (here, $\dot{x}_0$ is obtained by fitting a chebyshev polynomial to x_0 and taking it’s derivative); (3) “Rollout and Zero,” which corresponds to an initial guess where u_0 is zero and x_0 is the result of forward simulating the dynamical system using zero input; and (4) “Rollout and Constant,” which corresponds to an initial guess for u_0 that is constant and is equal to applying RNEA to the initial $\mathbf{x}_0$ while assuming that $\ddot{q}(0) = 0$ and x_0 is equal to forward simulating the system using that control input. For Crocoddyl, we considered the following initial guesses: “Zeros” and “Line and RNEA,” as defined in the Aligator case, and “Constants,” which corresponds to setting $x(t) = \mathbf{x}_0$ for all t and u_0 equal to a constant that is equal to applying RNEA to the initial $\mathbf{x}_0$ while assuming that $\ddot{q}(0) = 0$ and x_0 . For PHLAME, we only considered the initial guess “Line”, which corresponds to an initial guess where x_0 is a line connecting $\mathbf{x}_0$ and $\mathbf{x}_f$. Note that for our method we do not need to specify an initial guess u_0 .

Aligator and Crocoddyl have several parameters that are specific to their implementation. First, each method relies upon discretizing time and allow a user to specify a time discretization, δ_t . Second, each method allows one to include a running cost. We choose this running cost to be the 2-norm of the input added to the 2-norm of the state with weights w_u and w_x , respectively. Third, each method allows one to include a terminal cost with weight w_{xf} . We choose this to be the 2-norm of the difference of the final state from $\mathbf{x}_f$. For Aligator, we also consider the parameters μ_{init} and ϵ_{tol} , where the former corresponds to the initial value of the augmented Lagrangian penalty parameter and the latter to the solver tolerance. Lastly, for Aligator we also add an equality constraint that enforces the desired final state $\mathbf{x}_f$ and inequality constraints that enforce that the joint frames are not inside any of the sphere obstacles.

As for PHLAME. First, p is the degree of the polynomial that represents the solution. Second, k is a penalty that ensures dynamic feasibility and was first introduced in (2.11). Third, s_{max} corresponds to the maximum “time” that the PDE has to evolve. Fourth, only for the experiments that have obstacles (inequality constraints) we also consider the parameters k_{cons} , introduced in (2.29) and c_{cons} , introduced in (2.30) which control the weight of the constraint satisfaction and the sharpness of the activation function respectively.

In summary, for Crocoddyl in each experiment we perform a grid search of Initial guess, δ_t, w_u, w_x , and w_{xf} . For Aligator we search over the Initial guess, $\delta_t, w_u, w_x, w_{xf}, \mu_{init}$ and ϵ_{tol} . For unconstrained PHLAME, p, s_{max} and k and for constrained PHLAME p, s_{max}, k, c_{cons} and k_{cons} .

Number of Bodies (N)	Evaluation Time [μ s]	
	PHLAME	AGHF [72]
2	4.071 ± 1.963	46.819 ± 11.559
3	7.029 ± 1.035	593.054 ± 286.744
4	9.895 ± 1.297	44582.9 ± 7963.98
5	13.234 ± 1.411	359506 ± 36460.7
6	16.122 ± 1.190	DNF
7	21.195 ± 1.580	DNF
22	104.507 ± 2.516	DNF

Table 2.1: A table showing mean evaluation times (in μ s) of the right-hand side of the AGHF using PHLAME and AGHF [72] as the number of bodies (N) increases from 2 to 22. For $N \leq 5$ the results correspond to the 1-5 link pendulum, for $N = 7$ to the Kinova Gen3 Arm and for $N = 22$ for the Pinned Digit biped robot. For each method each robot’s AGHF RHS was evaluated at a 1000 random robot configurations each. For $N > 5$ the MATLAB method used in [72] was unable to symbolically generate the AGHF right-hand side without running out of memory. We denote the results where [72] was unable to generate the right-hand side and find a solution as DNF (Did Not Find). Regardless, we see that PHLAME’s right-hand side evaluation time is much faster and scales better than the original AGHF [72].

2.6.2 PHLAME AGHF Evaluation vs Original AGHF [72]

This section compares evaluating the right hand side of the AGHF PDE using the original Matlab based implementation [72] and using Algorithm 2. We compare the evaluation time as the number of bodies increases from 2 to 22. The systems with N between 2 and 6 are penduli systems (i.e., 2-link pendulum, 3-link pendulum, etc.). The $N = 7$ system is the Kinova Gen3 and the $N = 22$ system is the pinned Digit V3 biped. We evaluate the AGHF at a 1000 random configurations for all of these systems. Table 2.1 shows the evaluation times of the AGHF using PHLAME and the original AGHF [72].

The results demonstrate the significant speedup achieved by Algorithm 2 in evaluating the AGHF. This, combined with the substantial reduction in the number of nodes needed to accurately solve the AGHF, allows PHLAME to efficiently scale to high-dimensional systems while preserving fast solution times. The next section discusses how PHLAME’s overall solve time scales with increasing number of bodies (N).

2.6.3 Scalability with Increasing State Dimension in Unconstrained Systems

This section compares the methods Crocoddyl, PHLAME and AGHF [72] to solve a fixed time swing up problem for a 1-, 2-, 3-, 4-, and 5-link pendulum model, a fixed time and final

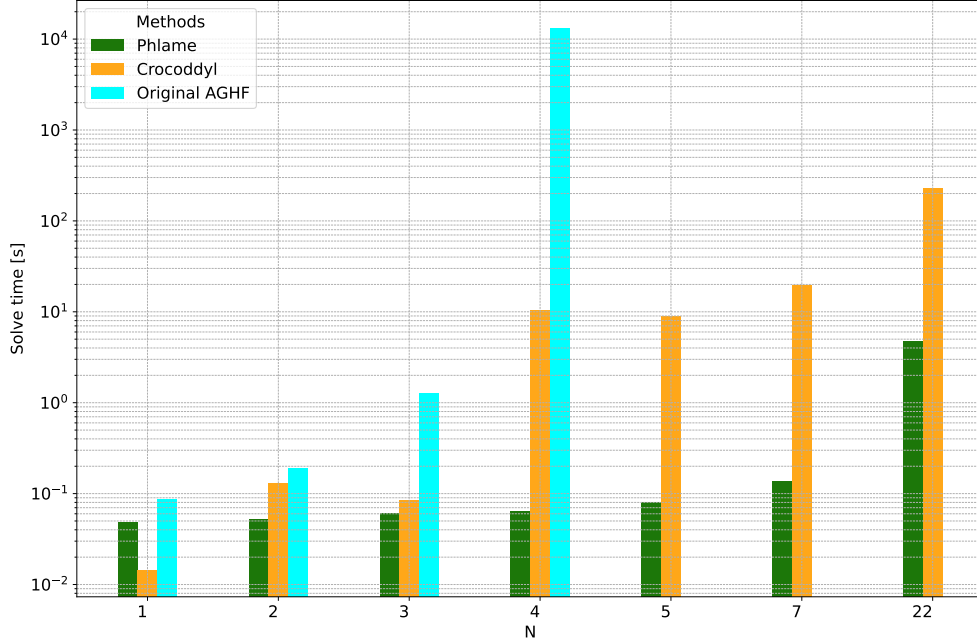
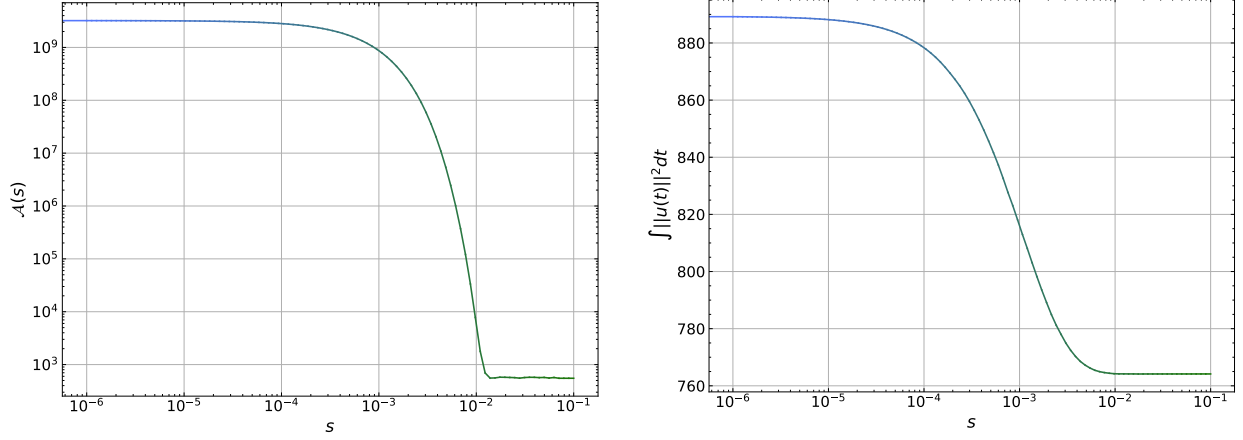


Figure 2.4: A bar plot comparing the mean solve times for three different trajectory optimization algorithms: PHLAME, AGHF [72] and Crocoddyl [79]. For $N \leq 5$ the results correspond to the 1-5 link pendulum, for $N = 7$ to the Kinova Gen3 Arm and for $N = 22$ for the Pinned Digit biped robot. Each experiment was run ten times using the best solver parameter set. For $N \geq 5$ we do not show results for the Original AGHF implementation as no results were obtained in a reasonable amount of time. Overall, we see that PHLAME shows better empirical scalability and solve time than the other methods. The values of $\mathbf{x}_0$ and $\mathbf{x}_f$ are provided in https://github.com/roahmlab/PHLAME_comparisons.

state specified trajectory optimization for the Kinova arm, and a fixed time and final state trajectory optimization having the pinned Digit execute a step. As described in Section 2.6.1, we run a grid search in parallel to obtain the best parameter sets for PHLAME and Crocoddyl per problem; the values of the grid search for Crocoddyl without obstacles can be seen in Table A.1. The parameters w_u, w_{x_f}, w_x, dt and initial guess correspond to solver specific parameters. For PHLAME, the explored parameter grids for the penduli, Kinova and Digit are in Tables A.3, A.4, and A.4 respectively. For AGHF, we use the same parameters as for PHLAME, but with 100 nodes to discretize the trajectory as that was the minimum we could use to yield a good solution.

Figure 2.4 compares solve time with the best parameter set using the described feedback controller with gains $k_p = 100, k_v = 100$ for Digit and $k_p = 10, k_v = 10$ for all other robotic platforms. The results demonstrate how PHLAME is able to generate solutions faster than these state-of-the-art methods. As the system dimension increases, PHLAME has a much lower solve time than the comparison methods.

2.6.4 Trajectory Optimization for the Kinova Arm without Obstacles



(a) Evolution of the action functional ($\mathcal{A}(s)$) as the PHLAME evolves the trajectory along s for one of the kinova no obstacles scenarios. (b) Evolution of $\int_0^T \|u(t)\|^2 dt$ as the PHLAME evolves the trajectory along s for one of the kinova no obstacles scenarios.

Figure 2.5: Side-by-side figures showing the convergence of action functional and $\int_0^T \|u(t)\|^2 dt$ for one of the trajectories found in the kinova no obstacle trajectory optimization problems. The lines transitions from **dark blue** to **dark green** as the trajectory converges to an extremal solution of (2.7)

We consider ten different scenarios with distinct $\mathbf{x}_0, \mathbf{x}_f$. We select five out of the ten scenarios to perform grid search over to find the best solver specific parameters. Tables A.1 and A.4 summarize the grids for Crocoddyl and PHLAME, respectively. Then, for each solver method, we find the single solver parameter set that yields the most successes across the five scenarios and the lowest time to solve when using a feedback controller with gains of $k_p = k_v = 10$. The best parameter set for PHLAME was $s_{max} = 0.1, k = 10^9, p = 9$. The best parameter set for Crocoddyl was $w_x = 0.0001, w_u = 0.0001, \delta t = 0.0001, w_{xf} = 1$, and Initial Condition = “Zeros”. We then run all the scenarios sequentially ten times with the best parameter set and apply the aforementioned feedback controller.

The results are shown in Table 2.2. For all the scenarios, both methods had a 100% success rate and were able to generate dynamically feasible trajectories. PHLAME was faster than Crocoddyl; however, Crocoddyl’s trajectory had a smaller controller input. Note that PHLAME’s control input still satisfied the actuation limits of the Kinova arm.

Figure 2.5 illustrates the evolution of the Action Functional and the $\int_0^T \|u(t)\|^2 dt$ for one of the kinova arm trajectory optimization problems. Note that the action functional

Metric	PHLAME	Crocoddyl
Success Rate [%]	100	100
$\int_0^T u_{fb}(t) ^2 dt$	430.5 ± 204.8	120.7 ± 50.98
Solve Time [s]	0.2102 ± 0.02454	213.8 ± 82.1

Table 2.2: Comparison of success rate, control effort norm ($\int_0^T ||u_{fb}(t)||^2 dt$), and solve time for Kinova Gen3 trajectory optimization experiments without obstacles.

decreases significantly, ensuring the trajectory’s dynamic feasibility while reducing control inputs, ultimately converging to an extremal solution of the AGHF (2.7), which minimizes the Action Functional.

2.6.5 Trajectory Optimization for Pinned Digit without Obstacles

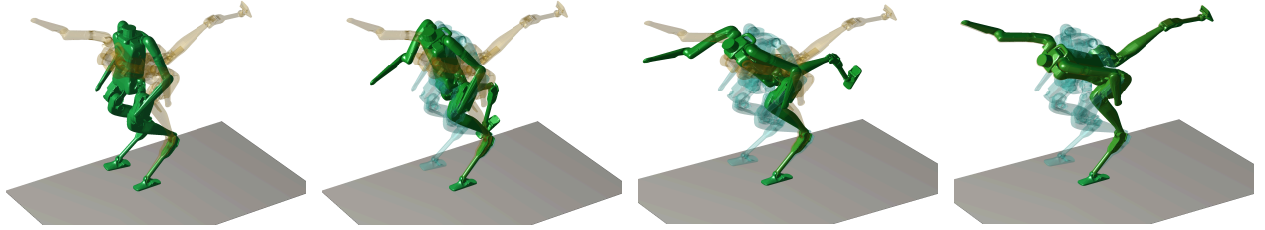
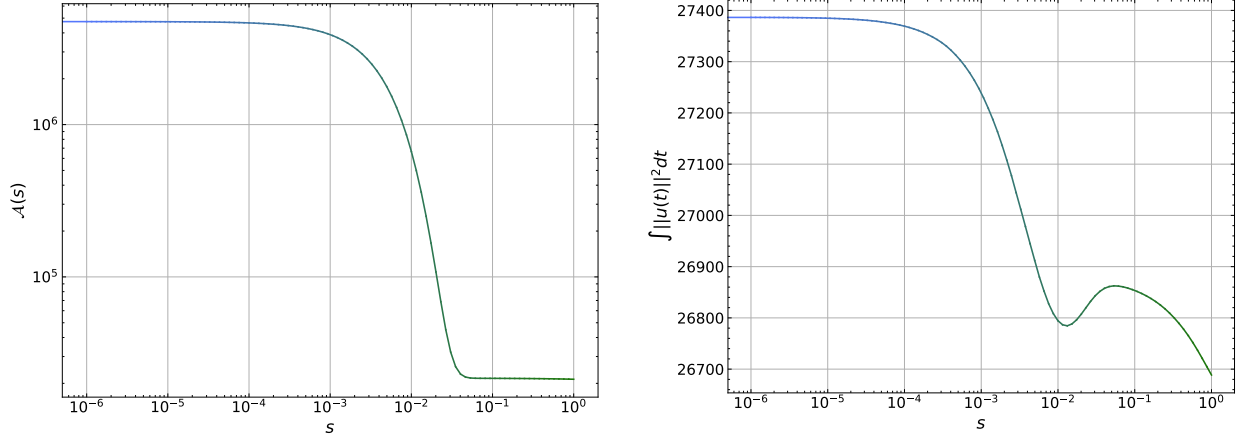


Figure 2.6: Snapshots of the forward simulated solution obtained by PHLAME for Pinned Digit doing a yoga stretch. The intermediate poses in **green**, the initial in **turquoise** and the final in **gold**.

We consider two scenarios that correspond to taking a step and doing a yoga pose (Figure 2.6). In both experiments one of the feet is pinned to the ground. As before, we first run the grid search with the parameters in Tables A.1 and A.5 for Crocoddyl and PHLAME, respectively. The best parameter set for PHLAME was $s_{max} = 1, k = 10^7, p = 6$. The best parameter set for Crocoddyl was $w_x = 0.001, w_u = 0.0001, \delta t = 0.001, w_{xf} = 1000$, and Initial Condition = “Rollout and Constant”.

Then, for each solver method, we find the single solver parameter set that yields the lowest time to solve when using a feedback controller with gains of $k_p = k_v = 100$ in the step scenario. After that, we ran all the scenarios sequentially ten times with the best parameter set and the mentioned controller gains. Table 2.3 shows the results of these trajectory optimization problems. Both methods had a 100% success rate, with PHLAME being able to generate trajectories for the 22 DOF Digit biped in about 3s, two orders of magnitude faster than Crocoddyl. However, Crocoddyl’s trajectory had a smaller $\int_0^T ||u_{fb}(t)||^2 dt$. For the Pinned Digit Bipod with the closed kinematic chain removed the an additional two sets



(a) Convergence of the action functional ($\mathcal{A}(s)$) to a local minimum as the PHLAME evolves along s for the Digit yoga stretch scenario. (b) Evolution of $\int_0^T \|u(t)\|^2 dt$ as the PHLAME evolves along s for the Digit yoga stretch scenario.

Figure 2.7: Side-by-side figures showing the convergence of action functional and the evolution of $\int_0^T \|u(t)\|^2 dt$ for the Digit yoga stretch scenario.

of joints are made actuated to account for the removal of the closed loop kinematic chain. However, since these actuated joints are not present in the original Digit robot model, we cannot ascertain whether the computed trajectory adheres to the control limits of the actual hardware.

Metric	PHLAME	Crocoddyl
Success Rate [%]	100	100
$\int_0^T \ u_{fb}(t)\ ^2 dt$	34570 ± 8418	5140 ± 1853
Solve Time [s]	3.331 ± 1.437	225.2 ± 7.62

Table 2.3: A table showing the success rate, norm of the control effort ($\int_0^T \|u_{fb}(t)\|^2 dt$) and solve time in seconds for a set of Digit V3 trajectory optimization experiments with no obstacles

2.6.6 Trajectory Optimization for the Kinova Arm with Obstacles

In these experiments, we consider ten different scenarios where each one consists of a different $\mathbf{x}_0$, $\mathbf{x}_f$ similar to the one in Section 2.6.4. However, in each of these scenarios there are 5 obstacles in the arm's path that it must plan around to reach the goal. We also start with an initial guess x_0 that is not in collision with any of the obstacles. As before, the average results for the best parameter sets per experiment are shown in Table 2.4 where

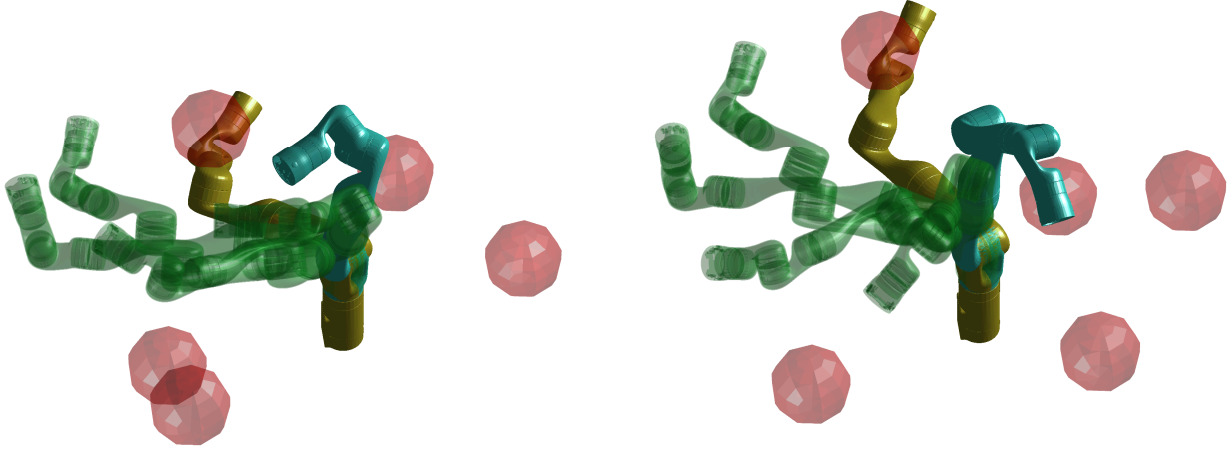


Figure 2.8: Two figures show trajectories generated by PHLAME for the Kinova Gen3 to get from the initial poses in **turquoise** to the final poses in **gold** while avoiding the obstacles in **red**. The intermediate poses in the paths it takes are shown in **green**.

the controller gains were $k_p = k_v = 10$. The best parameter set for PHLAME was $s_{max} = 0.1, k = 10^9, p = 8, k_{cons} = 10^9, c_{cons} = 1$. The best parameter set for Aligator was $w_x = 0.001, w_u = 0.0001, \delta t = 0.001, w_{xf} = 1$, and Initial Condition = “Line and RNEA”, $\epsilon_{tol} = 0.001, \mu_{init} = 10^{-7}$. In this setting PHLAME is able to successfully avoid the obstacles and meet the goal in all the scenarios while generating trajectories much faster than Aligator.

Figure 2.8 illustrates one of these scenarios. Aligator had fewer successes than PHLAME, but had a smaller control input. Similar to the no obstacle case in Section 2.6.4, the control inputs generated by PHLAME still satisfied the control limits.

2.6.7 Trajectory Optimization for Pinned Digit with Obstacles

In this section we consider four experiments that consist of the pinned Digit taking a single step over a single half-sphere obstacle of four different radii, with the smallest and largest spheres having radii of 1cm and 20.3cm respectively. Once again we start with an initial guess x_0 that is not in collision with any of the obstacles. The average results for the best parameter sets are shown in Table 2.5 where the controller gains were $k_p = k_v = 100$. The best parameter set for PHLAME was $s_{max} = 1.0, k = 10^7, p = 6, k_{cons} = 10^5, c_{cons} = 200$. For Aligator none of the parameter sets yielded a success when the allowable final state error $\epsilon = 0.05$ as with all the other experiments.

Here we see that PHLAME is able to successfully generate trajectories to get Digit to

Metric	PHLAME	Aligator
Success Rate [%]	100	60
$\int_0^T u_{fb}(t) ^2 dt$ (Successes)	902.2 ± 122.8	194.3 ± 78.95
$\int_0^T u_{fb}(t) ^2 dt$ (Intersection)	903.8 ± 77.49	194.3 ± 78.95
Solve Time [s] (Successes)	0.2968 ± 0.0492	75.84 ± 9.405
Solve Time [s] (Intersection)	0.3261 ± 0.03808	75.84 ± 9.405

Table 2.4: Comparison of success rate, control effort norm ($\int_0^T ||u_{fb}(t)||^2 dt$), and solve time for Kinova Gen3 trajectory optimization experiments with obstacles. The 'Intersection' rows report metrics for trials where both methods succeeded; the 'Successes' rows report metrics for successful trials of each method.

step over the obstacles in less than 6s on average. Whereas Aligator is unable to succeed in any of the trials due to it's final state being too far from the goal state.

Method name	Success Rate	$\int_0^T u_{fb}(t) ^2 dt$	Solve time [s]
PHLAME	4/4	43633 ± 904.6	5.285 ± 0.2262
Aligator	0/4	DNF	DNF

Table 2.5: A table showing the success rate, norm of the control effort ($\int_0^T ||u_{fb}(t)||^2 dt$) and Solve Time in seconds for Digit trajectory optimization experiment where Digit steps over different sized obstacles when the threshold for success ϵ is set to 0.05. We denote the results where one of the methods was unable to generate a successful solution as DNF (Did Not Find).

Because Aligator had no successes with the regular error threshold for success ($\epsilon = 0.05$), we reran the same experiment of taking a step, but with a larger error threshold ($\epsilon = 0.25$). The results for this experiment are shown in Table 2.6. The results show that PHLAME still outperforms Aligator in terms of time to solve the optimal control problem. The best parameter set for Aligator was $w_x = 1.0, w_u = 0.01, \delta t = 0.01, w_{xf} = 10^{-6}$, and Initial Condition = "Zeros", $\epsilon_{tol} = 0.001, \mu_{init} = 10^{-8}$.

Metric	PHLAME	Aligator
Success Rate [%]	100	25
$\int_0^T u_{fb}(t) ^2 dt$ (Successes)	43633 ± 904.6	68306 ± 0
$\int_0^T u_{fb}(t) ^2 dt$ (Intersection)	42964 ± 0	68306 ± 0
Solve Time [s] (Successes)	5.285 ± 0.2262	187.6 ± 1.571
Solve Time [s] (Intersection)	5.335 ± 0.2984	187.6 ± 1.571

Table 2.6: Comparison of success rate, control effort norm ($\int_0^T ||u_{fb}(t)||^2 dt$), and solve time for Digit trajectory optimization experiments over obstacles with a success threshold $\epsilon = 0.25$. The ‘Intersection’ rows report metrics for trials where both methods succeeded; the ‘Successes’ rows report metrics for successful trials of each method.

2.7 Conclusion and Limitations

This work proposes PHLAME, a method for rapid trajectory optimization leveraged by solving the AGHF PDE. The AGHF PDE poses the trajectory generation problem as the solution to a PDE that evolves an initial trajectory that may not be dynamically feasible into some final trajectory that is dynamically feasible while minimizing the magnitude of the control inputs of the final trajectory. PHLAME applies pseudospectral collocation in conjunction with spatial vector algebra to rapidly solve the AGHF PDE enabling it to scale to high-dimensional robot systems. This chapter demonstrates that PHLAME can generate trajectories for high-dimensional systems much faster than state-of-the-art trajectory optimization methods, with PHLAME generating trajectories for the Digit biped on the order of 4 seconds.

PHLAME has its limitations: because the action functional (2.14) incorporates dynamic constraints through a penalty term, the AGHF solution does not minimize the control inputs to the fullest extent possible. Additionally, we have demonstrated PHLAME on legged systems by assuming that ground contact can always be maintained (i.e., we assumed there was infinite friction). Future work will extend this formulation to systems while checking to see if contact can be maintained.

CHAPTER 3

Rapid Motion Planning with the AGHF PDE for Arbitrary Objective Functions and Constraints

3.1 Introduction

Optimal Control is a powerful tool for motion planning and control of advanced robotic systems. For robust deployment in trajectory-based robotics-algorithms [55, 69, 20, 63, 82, 38], an optimal control algorithm should be (1) computationally efficient to enable online planning, (2) capable of handling nonlinear dynamics and constraints inherent in real-world tasks, (3) scalable to high-dimensional platforms like humanoids and manipulators, and (4) reliably convergent to feasible solutions despite their initialization. However, balancing these criteria is challenging.

3.1.1 Related Work

Current methods to address these challenges can be grouped into two primary approaches – Dynamic Programming (DP) and Variational methods. DP leverages Bellman’s Principle of Optimality to derive value functions and optimal policies. In particular, solving the Hamilton-Jacobi-Bellman (HJB) Partial Differential Equation (PDE) offers global optimality guarantees, but necessitates discretizing the entire state space, making it computationally infeasible for systems beyond 5-6 dimensions [37, 13]. Differential Dynamic Programming (DDP) methods, such as Crocoddyl [79], mitigate these issues by applying DP techniques locally along a given trajectory. Through iterative forward-backward passes, DDP variants achieve more rapid convergence, providing a practical compromise between optimality and computational tractability for high-dimensional robotic systems. Although DDP methods are more computationally efficient than global approaches, they can exhibit poor convergence



Figure 3.1: This chapter introduces BLAZE, a Phase 1 - Phase 2 Affine Geometric Heat Flow (AGHF) framework, to rapidly solve optimal control problems while respecting robot constraints and avoiding obstacles. It begins with an initial trajectory (shown in **orange** with the color gradient illustrating the evolution in time starting from darkest and going to lightest) that may violate constraints (e.g., the second and fourth pose of the arm are in collision with the boxes and outlined in **red**). If the initial trajectory is infeasible, BLAZE enters Phase 1, where it evolves the trajectory into a trajectory that satisfies all constraints (e.g., in the **blue** trajectory, the Kinova arm has been moved out of collision with the boxes). Once the trajectory satisfies all constraints, Phase 2 begins, optimizing the motion to minimize a user-specified cost function while maintaining feasibility (optimized trajectory shown **green**). BLAZE optimizes the trajectory to reach a target configuration while avoiding the obstacles while considering the full dynamical model of the arm. Note that optimal control (including Phase 1 and Phase 2) for this 14 dimensional state space model is completed within 3s while satisfying input, state, and collision avoidance constraints.

when initialized far from a local optimum.

Variational methods derive necessary optimality conditions via calculus of variations, offering an alternative route to solving control problems. Within this framework, direct methods discretize a continuous optimal control problem into a nonlinear program (NLP). For example, direct collocation methods [43, 34] approximate trajectories using polynomial functions. Although effective at handling complex constraints, these methods can be computationally intensive for high-dimensional systems and sensitive to discretization choices and initial guesses.

Recently, another PDE-based method using the Affine Geometric Heat Flow (AGHF) PDE has been proposed [72, 3]. Notably these methods have been shown to be able to generate trajectories for high dimensional systems faster than existing methods (e.g., on the order of seconds for systems with more than a 40 dimensional state space model). These methods pose the trajectory optimization problem as the solution to a PDE that evolves

an initial trajectory that may not be dynamically feasible into a final trajectory that is dynamically feasible while minimizing control input magnitudes. In contrast to the HJB PDE, the AGHF solution is defined over a two-dimensional domain irrespective of the system’s dimension. This enables the AGHF PDE to achieve significant computational speedups while preserving dynamic feasibility in motion planning and incorporating various kinds of constraints. Though these AGHF methods show promise for rapidly generating trajectories for high-dimensional systems, currently they have been restricted to considering only one kind of cost function – the integral of the squared norm of the control inputs along the trajectory. Additionally, to find solutions rapidly these methods often require an initial guess that does not violate any constraint other than the dynamic feasibility constraint.

3.1.2 Chapter Contributions

To address these limitations, this chapter proposes BLAZE [2], a method that builds a generalized AGHF formulation that accommodates arbitrary cost functions, enabling the generation of diverse trajectories that were previously unattainable. This method enables one to rapidly compute dynamically feasible trajectories for complex, highly dynamic tasks for high dimensional systems by leveraging spatial vector algebra, a pseudospectral method and a Phase 1-Phase 2 algorithm for the solving the AGHF PDE (as illustrated in Figure 3.1). The Phase 1–Phase 2 algorithm initially transforms any infeasible trajectories into feasible ones during Phase 1. In Phase 2, it then optimizes these feasible trajectories to minimize cost while satisfying all constraints. The contributions of this chapter are three-fold: First, this chapter illustrates how to formulate the AGHF Action Functional to solve optimal control problems with arbitrary cost functions (Section 3.3.3). Second, this chapter describes how to implement a Phase 1-Phase 2 style method that enables the AGHF initial guess to violate the constraints while still generating feasible solutions rapidly (Section 3.5). Third, this chapter describes how to efficiently enforce input constraints in the AGHF (Section 3.4.2). The utility of these contributions is illustrated by comparing the performance of the proposed method to state of the art trajectory optimization techniques and a hardware demonstration of the proposed method on the Kinova Gen3 robot.

The remainder of the chapter is arranged as follows: Section 3.2 presents the background and introduces the relevant notation for the chapter. Section 3.3 introduces the AGHF and discusses the underlying theory associated with the AGHF. Section 3.4 details how to incorporate constraints into the AGHF to enable actions like obstacle avoidance. Section 3.6 evaluates the proposed algorithm’s efficiency through simulation comparisons with several state-of-the-art methods and validates its performance on a Kinova Gen3 hardware platform.

3.2 Preliminaries

This section introduces the notation used throughout this chapter. This chapter is focused on performing trajectory optimization for robot systems whose dynamics can be written as follows:

$$H(q(t))\ddot{q}(t) + C(q(t), \dot{q}(t)) = Bu(t), \quad (3.1)$$

where $q(t) \in \mathbb{R}^N$ is the configuration of the robot at time t , $u(t) \in \mathbb{R}^m$ is the input applied to the robot at time t , $H(q(t))$ is the mass matrix, $C(q(t), \dot{q}(t))$ is the grouped Coriolis and gravity term and B is the actuation matrix. For convenience, let $\mathbf{x}(t)$ correspond to the vector of $q(t)$ and $\dot{q}(t)$. To be consistent with the notation in the rest of the chapter we refer to the first N and last N components of $\mathbf{x}(t)$ as $\mathbf{x}_{P1}(t)$ and $\mathbf{x}_{P2}(t)$, respectively. Using these definitions, we can represent the dynamics of the robot (3.1) as a control affine system:

$$\dot{\mathbf{x}}(t) = F_d(\mathbf{x}(t)) + F(\mathbf{x}(t))u(t), \quad (3.2)$$

where

$$F_d(\mathbf{x}(t)) = \begin{bmatrix} \mathbf{x}_{P2}(t) \\ -H^{-1}(\mathbf{x}_{P1}(t))C(\mathbf{x}_{P1}(t), \mathbf{x}_{P2}(t)) \end{bmatrix} \quad (3.3)$$

$$F(\mathbf{x}(t)) = \begin{bmatrix} 0_{N \times m} \\ H^{-1}(\mathbf{x}_{P1}(t))B \end{bmatrix} \quad (3.4)$$

For convenience, we assume without loss of generality that we are interested in the evolution of the system for $t \in [0, T]$. Lastly, let L^2 refer to the set of square integrable functions defined for $t \in [0, T]$. Throughout this chapter, we also utilize the following definition that is used throughout the calculus of variations because it allows one to describe how functions behave as they go to infinity:

Definition 9 (Coercive Functions [27, Section 8.2]). *A function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is called coercive if there exist constants $\alpha > 0, \beta \geq 0$ such that $f(x) \geq \alpha\|x\|_\infty - \beta$ for all x .*

The objective of this chapter is to develop an algorithm to construct a trajectory beginning from some user-specified initial condition, $\mathbf{x}_0$, and ending in some user-specified terminal condition, $\mathbf{x}_f$, while satisfying state and input constraints and the dynamics in (3.2) for all $t \in [0, T]$ all while minimizing an arbitrary cost function. For convenience, let the zero superlevel set of functions g_j for each $j \in \mathcal{J}$ and h_i for each $i \in \mathcal{I}$ represent a collection of state and input inequality constraints where $\mathcal{J} \subset \mathbb{N}$ and $\mathcal{I} \subset \mathbb{N}$ are each finite sets. Using

these definitions, one can formulate the trajectory optimization problem:

$$\begin{aligned}
& \inf_{u \in L^2} \int_0^T c(\mathbf{x}(t), \dot{\mathbf{x}}(t), u(t)) dt & (\text{OCP}) \\
& \text{s.t.} \quad \dot{\mathbf{x}}(t) = F_d(\mathbf{x}(t)) + F(\mathbf{x}(t))u(t), \quad \forall t \in [0, T], \\
& \quad g_j(\mathbf{x}(t), \dot{\mathbf{x}}(t)) \leq 0 & \forall t \in [0, T], \forall j \in \mathcal{J}, \\
& \quad h_i(u(t)) \leq 0 & \forall t \in [0, T], \forall i \in \mathcal{I}, \\
& \quad \mathbf{x}(0) = \mathbf{x}_0, \\
& \quad \mathbf{x}(T) = \mathbf{x}_f,
\end{aligned}$$

The goal of this chapter is to develop a method to rapidly generate trajectories for high-dimensional systems while incorporating multiple constraints using the Affine Geometric Heat Flow Partial Differential Equation to solve (OCP). To ensure the convergence of the AGHF, we make the following assumption:

Assumption 10 (Continuity of Dynamics). *Both F_d and F are C^2 , globally Lipschitz continuous (with constants Y_1 and Y_2 respectively), and F has constant rank almost everywhere in $\mathbb{R}^n$. Suppose c , g_j , and h_i are C^2 in all of their arguments for all $j \in \mathcal{J}$ and $i \in \mathcal{I}$. Additionally, assume that there exists a trajectory that satisfies the constraints of (OCP).*

Note that the dynamics of rigid-body robotic systems are smooth and Lipschitz continuous when restricted to a compact domain. In addition, we do not assume that the trajectory that satisfies the constraints of (OCP) is given to the user. Rather we only assume its existence.

3.3 The Affine Geometric Heat Flow Partial Differential Equation

The Affine Geometric Heat Flow (AGHF) is a parabolic PDE framework for trajectory optimization that deforms an arbitrary initial trajectory (including one that does not satisfy the dynamics) into a dynamically feasible one while minimizing the squared control input of the trajectory. In this section, we review the foundational AGHF theory and introduce novel theorems with proofs that generalize the AGHF framework to optimal control problems with arbitrary cost functions as posed in (OCP). A more detailed treatment of the background knowledge on the AGHF can be found in [72, 71, 73, 3]. *Throughout this section, we assume that there are no inequality constraints in (OCP). The inequality constraint case is considered in Section 3.4.*

3.3.1 Homotopies and the Action Functional

To describe the evolution of trajectories by the AGHF PDE, we begin by defining a homotopy: $x : [0, T] \times [0, s_{max}] \rightarrow \mathbb{R}^{2N}$, that is twice differentiable with respect to its first argument and differentiable with respect to its second argument. For convenience, we denote $x(t, s)$ by $x_s(t)$ and we denote $\frac{\partial x}{\partial t}(t, s)$ by $\dot{x}(t, s)$ or $\dot{x}_s(t)$. In practice this homotopy can be any twice differentiable initial guess for a trajectory that starts at $\mathbf{x}_0$ and terminates at $\mathbf{x}_f$. As is done in Section 3.2, we refer to the first N and last N components of x_s as x_{P1} and x_{P2} , respectively. Additionally, let the first and second time derivatives of these states be defined as $\dot{x}_{P1}$, $\dot{x}_{P2}$ and $\ddot{x}_{P1}$, $\ddot{x}_{P2}$, respectively.

Next, define the *Lagrangian* $L : \mathbb{R}^{2N} \times \mathbb{R}^{2N} \rightarrow \mathbb{R}$, which we assume is C^2 and twice differentiable with respect to any of its arguments. Subsection 3.3.3 describes how to select L to ensure that the AGHF minimizes the cost function in (OCP). Finally, define the *Action Functional*:

$$\mathcal{A}(x_s) = \int_0^T L(x_s(t), \dot{x}_s(t)) dt. \quad (3.5)$$

Using these definitions, we can define the AGHF PDE:

Definition 11 (Affine Geometric Heat Flow). *The Affine Geometric Heat Flow is a parabolic partial differential equation defined as:*

$$\begin{aligned} \frac{\partial x}{\partial s}(t, s) = M^{-1}(x(t, s)) & \left(\frac{d}{dt} \frac{\partial L}{\partial \dot{x}_s}(x_s(t), \dot{x}_s(t)) \right. \\ & \left. - \frac{\partial L}{\partial x_s}(x_s(t), \dot{x}_s(t)) \right), \end{aligned} \quad (3.6)$$

with the following boundary conditions:

$$\begin{aligned} x_s(0) &= \mathbf{x}_0, \quad \forall s \in [0, s_{max}], \\ x_s(T) &= \mathbf{x}_f, \quad \forall s \in [0, s_{max}]. \end{aligned} \quad (3.7)$$

where $M : \mathbb{R}^{2N} \rightarrow \mathbb{S}_+^{2N}$ is a user specified matrix valued function and $\mathbb{S}_+^{2N}$ is the set of positive semi-definite matrices.

Note that this AGHF formulation differs from standard approaches in that here we allow $M(x(t, s))$ to be an arbitrary positive semi-definite and invertible matrix. This enables us to consider general cost functions. The traditional AGHF formulation designs M specifically to penalize evolution toward dynamically infeasible states while prioritizing control effort reduction. However, as we show in this chapter, for general cost functions, M , should be defined differently.

When solving the AGHF PDE, one begins by specifying an initial curve $x_{init} : [0, T] \rightarrow \mathbb{R}^{2N}$. As the AGHF PDE evolves forward in s , one can prove that the action functional is minimized. In addition, if during that evolution the AGHF converges to a curve where the right-hand side of the AGHF PDE is equal to 0, then one has found a curve that extremizes the Action Functional. Such a curve is called a *steady state solution*. We formalize these observations in the following Lemma that describes the convergence properties of the AGHF and whose proof can be found in Appendix B.1.

Lemma 12 (Action Functional Along the AGHF Homotopy). *Let x_s satisfy the AGHF PDE (3.6). Then, $\frac{dA(x_s)}{ds} \leq 0$ for all s . In addition, if the right hand side of the AGHF PDE when evaluated at x_{s^*} is equal to 0 for some $s^* \in [0, s_{max})$, then $\frac{dA(x_{s^*})}{ds} = 0$.*

In other words, given the Action Functional (3.5), as the AGHF PDE evolves forward in s , the Action Functional is minimized. In addition, if during that evolution the AGHF converges to a curve where the right hand side of the AGHF PDE is equal to 0, then one has found a *steady state solution* to the AGHF.

3.3.2 Solving the AGHF Rapidly

Unlike the Hamilton-Jacobi-Bellman (HJB) PDE, which suffers from the curse of dimensionality due to its exponential scaling with state dimension, the AGHF PDE scales polynomially [32]. This favorable scaling arises because the solution to the AGHF PDE has a two-dimensional domain and has a range whose dimension grows linearly with the state dimension. Because the AGHF solution has a two-dimensional domain, the AGHF PDE can be solved using the Method of Lines (MOL) [93].

The MOL discretizes the AGHF solution domain along one dimension (typically t) into a set of nodes, reformulating the PDE at each node as a system of coupled Ordinary Differential Equations (ODEs). These coupled ODEs can then be solved in s using numerical ODE solvers. During each ODE solver step, the right-hand side (RHS) of the AGHF PDE must be evaluated at every node. For high-dimensional systems, this process becomes computationally expensive, as it requires repeatedly computing the system dynamics and their derivatives.

In classical MOL AGHF implementations [72], the nodes are uniformly spaced, and the solution accuracy improves as more nodes are introduced. However, because the number of function evaluations scales linearly with the number of nodes, a finer discretization requires frequent evaluations of the AGHF RHS, significantly increasing computational cost. Recent approaches [3] address this by employing a pseudospectral MOL that strategically reduces

the number of required nodes while maintaining accuracy. This pseudospectral formulation enables precise computation of time derivatives. Additionally, one can accelerate the evaluation of the AGHF's RHS by utilizing spatial vector algebra and rigid body dynamics algorithms [3].

3.3.3 Ensuring $\mathcal{A}$ Coincides with the (OCP) Cost Function

To ensure that the Action Functional, $\mathcal{A}$, being minimized coincides with minimizing an arbitrary cost function, $c(\mathbf{x}(t), \dot{\mathbf{x}}(t), u(t))$, of the (OCP) we must design L carefully. Before defining this action functional we first introduce one additional definition:

Definition 13 (Control Extraction). *Suppose x_s corresponds to a continuously differentiable trajectory at any $s \in [0, s_{max}]$. Let $u_s : [0, T] \rightarrow \mathbb{R}^m$ be the extracted control input given by:*

$$u_s(t) = \begin{bmatrix} 0_{N \times N} & I_{N \times N} \end{bmatrix} \bar{F}(x_s(t))^{-1} (\dot{x}_s(t) - F_d(x_s(t))). \quad (3.8)$$

where

$$\bar{F}(x_s(t)) = \begin{bmatrix} F_c(x_s(t)) & F(x_s(t)) \end{bmatrix} \in \mathbb{R}^{2N \times 2N}, \quad (3.9)$$

and $F_c(x_s(t)) \in \mathbb{R}^{2N \times (2N-m)}$ is a differentiable matrix such that $\bar{F}$ is invertible for all $x_s(t)$ in $\mathbb{R}^{2N}$.

This definition describes how to synthesize a control input given a specific trajectory. For notational convenience, we have omitted the explicit dependency of u_s on x_s and $\dot{x}_s$. Note that F_c in the above definition can be obtained using the Gram-Schmidt procedure. We give an explicit example of how to construct F_c when describing our experimental results in Section 3.6. Before moving on, we call attention to one important consideration: it remains unclear whether applying this extracted control input to the dynamical system (3.2) would reproduce the original trajectory x_s used in its creation. This question is answered in Theorem 15.

Given Definition 13, we can define a Lagrangian and associated Action Functional that encode solutions to (OCP) as we describe next:

Definition 14 (Lagrangian and Action Functional for (OCP)). *Let $u_s : [0, T] \rightarrow \mathbb{R}^m$ be the extracted control input at some $s \in [0, s_{max}]$ using Definition 13. Define L in terms of the extracted control input $u_s(t)$ as:*

$$L(x_s(t), \dot{x}_s(t)) = k_d \|\dot{x}_{P1}(s, t) - x_{P2}(s, t)\|_2^2 + c(x_s(t), \dot{x}_s(t), u_s(t)) \quad (3.10)$$

where $k_d > 0$. Then the corresponding Action Functional is given by

$$\mathcal{A}(x_s) = \int_0^T \left(k_d \|\dot{x}_{P1}(s, t) - x_{P2}(s, t)\|_2^2 + c(x_s(t), \dot{x}_s(t), u_s(t)) \right) dt \quad (3.11)$$

In summary, for each s , the Action Functional with L as described by Definition 14 consists of the integral of an arbitrary objective function, $c(\mathbf{x}(t), \dot{\mathbf{x}}(t), u(t))$, plus the error between the velocity states, x_{P2} , and the derivative of the position state, $\dot{x}_{P1}$. For trajectories that satisfy the system dynamics (3.2), this error term vanishes, reducing the Action Functional to the integral of $c(\mathbf{x}(t), \dot{\mathbf{x}}(t), u(t))$. Notably, as Lemma 12 shows, as the AGHF evolves the Action Functional decreases, which coincides with the objective function in (OCP) for dynamically feasible trajectories.

However, it is unclear whether the solution generated by the AGHF PDE eventually satisfies the system dynamics (3.2). The next result whose proof can be found in Appendix B.2 resolves this concern:

Theorem 15 (AGHF Generates Feasible Trajectories). *Consider a system governed by the dynamics in (3.2), with an initial state $\mathbf{x}_0$ and desired final state $\mathbf{x}_f$. Suppose Assumption 10 is satisfied, F_c in Definition 13 is a continuous function, and the cost function in the Action Functional (3.11) is coercive. Then there exists C_1, C_2 , and $C_3 > 0$, such that for any $k_d > 0$, there exists an open set $\Omega_{k_d} \subset \{x \in C^1([0, T] \rightarrow \mathbb{R}^{2N}) \mid x(0) = \mathbf{x}_0, x(T) = \mathbf{x}_f\}$ such that as long as the initial curve to the AGHF PDE satisfies $x_{init} \in \Omega_{k_d}$ then for sufficiently large s_{max} the integrated path $\tilde{x}$ using the extracted control input with $x_{s_{max}}$ plugged into the dynamics (3.2) satisfies the following error bound for any $t \in [0, T]$:*

$$\|\tilde{x}(t) - x_{s_{max}}(t)\|_2 \leq \sqrt{\frac{3TC_1C_2^2}{k_d} \exp\left(3T(Y_1^2T + Y_2^2C_3)\right)} \quad (3.12)$$

Theorem 15 proves that for sufficiently large k_d and s_{max} , the control input extracted from the solution to the AGHF PDE can be used to generate a trajectory that is arbitrarily close to a dynamically feasible trajectory of (3.2) and provides an explicit bound on how close the trajectory obtained by applying (3.8) is to a feasible trajectory of (OCP). With this result, we have an AGHF formulation that tries to minimize the objective function in (OCP) and enables one to generate dynamically feasible trajectories.

Remark 16 (Relationship to Previous Lagrangians). *Note that in Definition 14 if L is*

replaced by

$$L(x_s(t), \dot{x}_s(t)) = (\dot{x}_s(t) - F_d(x_s(t)))^T G(x_s(t)) \cdot (\dot{x}_s(t) - F_d(x_s(t))), \quad (3.13)$$

where G is given by

$$G = \begin{bmatrix} kI_{N \times N} & 0_{N \times N} \\ 0_{N \times N} & H^T H \end{bmatrix} \in \mathbb{R}^{2N \times 2N} \quad (3.14)$$

and $M = G$, then the resulting Action Functional and AGHF correspond to the standard formulation used in the literature [72, 3], which minimizes the squared control input (i.e $c(x_s(t), \dot{x}_s(t), u_s(t))$ is $\|u_s(t)\|_2^2$). However, the formulation in Definition 14 along with Theorem 15 generalizes this framework to accommodate a broader class of optimal control problems.

3.4 Incorporating Arbitrary Constraints into the AGHF

The previous section assumed that there were no constraints in (OCP) other than the dynamics constraints. This section discusses how to enforce general state and input constraints.

3.4.1 Constrained Lagrangian

We incorporate constraints into the AGHF by using a penalty term in the Lagrangian:

Definition 17 (Constrained Lagrangian). *Let k_{cons} be some large positive real number, and let $g_j(x(t), \dot{x}(t))$ be the j -th inequality constraint evaluated at $x(t)$ and $\dot{x}(t)$. Let the Constrained Lagrangian denoted by L_{cons} be defined as:*

$$L_{cons}(x(t), \dot{x}(t)) = L(x(t), \dot{x}(t)) + \sum_{j \in \mathcal{J}} b(g_j(x(t), \dot{x}(t))) \quad (3.15)$$

where

$$b(g_j(x(t), \dot{x}(t))) = k_{cons} \cdot (g_j(x(t), \dot{x}(t)))^2 \cdot S(g_j(x(t), \dot{x}(t))), \quad (3.16)$$

where $S : \mathbb{R} \rightarrow \mathbb{R}$ is defined as follows:

1. $S : \mathbb{R} \rightarrow \mathbb{R}$ is a positive, differentiable function,
2. $S(g_j(x(t), \dot{x}(t))) = 0$ when $g_j(x(t), \dot{x}(t)) \leq 0$ and

3. $S(g_j(x(t), \dot{x}(t))) = 1$ when $g_j(x(t), \dot{x}(t)) > 0$.

The function S ensures that the penalty term is activated only when the AGHF trajectory moves towards constraint violation. An example of such a function is described during the description of the experiments in Section 3.6. With a sufficiently large k_{cons} , the penalty term steers the trajectory away from infeasible regions, preventing convergence to solutions that violate constraints.

Using the Constrained Lagrangian, one can construct a corresponding AGHF by applying Definition 11:

Lemma 18 (AGHF for a Constrained Lagrangian). *Consider a Constrained Lagrangian, L_{cons} , as in Definition 17. Then the AGHF PDE is given by:*

$$\begin{aligned} \frac{\partial x}{\partial s} = M^{-1}(x_s(t)) & \left(\frac{d}{dt} \frac{\partial L}{\partial \dot{x}_s}(x_s(t), \dot{x}_s(t)) - \frac{\partial L}{\partial x_s}(x_s(t), \dot{x}_s(t)) \right. \\ & \left. + \sum_{j \in \mathcal{J}} \left(\frac{d}{dt} \frac{\partial b(g_j(x_s(t), \dot{x}_s(t)))}{\partial \dot{x}_s(t)} - \frac{\partial b(g_j(x_s(t), \dot{x}_s(t)))}{\partial x_s(t)} \right) \right) \end{aligned} \quad (3.17)$$

Notice that L_{cons} , still satisfies the properties of the Lagrangian introduced in Section 3.3.1. As a result, the convergence properties shown in Lemma 12 apply to the Constrained Lagrangian. If the function b in the definition of L_{cons} is coercive, then the results of Theorem 15 also apply to ensure that a dynamically feasible trajectory is found. Note this does not guarantee that the inequality constraints within (OCP) are eventually satisfied. This is particularly an issue when the initial trajectory passed to the homotopy violates the constraints. We address this particular challenge in Section 3.5.

3.4.2 Incorporating Input Constraints in the Lagrangian

This section discusses how to rapidly compute the derivatives of the input penalties which are required to enforce input constraints. Before introducing this formulation, we briefly describe how alternative AGHF formulations have enforced input constraints. Existing approaches to enforce input constraints using the AGHF require augmenting the state by treating inputs as additional states, and enforcing the input constraints as state constraints[72]. For high-dimensional robotic systems, this approach substantially increases the dimension of state space of the system and, consequently, the dimension of the AGHF PDE.

To avoid this issue, we instead enforce the input constraints by appending L_{cons} with the term $\sum_{i \in \mathcal{I}} b(h_i(u_s(t)))$. However to construct the AGHF as in Lemma 18 for this even larger Constrained Lagrangian, one must be able to compute partial derivatives of $b(h_i(u_s(t)))$ with respect to $x_s(t)$ and $\dot{x}_s(t)$. This requires then applying the chain rule and computing

partial derivatives of $u_s(t)$ with respect to $x_s(t)$ and $\dot{x}_s(t)$. This can be challenging for high-dimensional robotic systems. Fortunately, one can leverage the following lemma, whose proof can be found in Appendix B.3:

Lemma 19. *Suppose dynamics are given by (3.1) where $B = I$ then the control extraction formula (3.8) is equivalent to inverse dynamics and u_s can be computed directly using inverse dynamics.*

From Lemma 19 it follows that one can compute u_s using Rigid Body Dynamics Algorithms, such as the Recursive Newton-Euler Algorithm (RNEA). This can be used to efficiently compute $u_s(t)$ using $x_s(t)$ and $\dot{x}_s(t)$ [33]. By employing an extended version of RNEA that recursively applies the chain rule [16] we can also rapidly evaluate the derivatives of the control inputs $\frac{\partial u_s}{\partial x_s}$ and $\frac{\partial u_s}{\partial \dot{x}_s}$.

3.5 Designing a Phase 1 - Phase 2 Algorithm using the AGHF

Similar to other trajectory optimization methods, AGHF requires x_{init} as an initial guess. As detailed in Section 3.3, the AGHF PDE transforms this initial trajectory x_{init} into an optimal final trajectory. If x_{init} does not satisfy the optimal control problem’s constraints, the AGHF solver must address constraint violations, leading to larger penalties that make the AGHF evaluate to larger and larger values, which can cause the ODE solver to take smaller steps to ensure solution accuracy, which increases the convergence time. Previous approaches [3] mitigate this issue by providing initial guesses that satisfy the constraints, facilitating faster AGHF convergence. However, designing such feasible initial trajectories is challenging for systems with many complex constraints such as input constraints.

To enhance the efficiency of the AGHF and enable rapid convergence, we propose a Phase 1- Phase 2 Algorithm using the AGHF. Phase 1-Phase 2 Methods are used in optimization to first drive an initial guess into a feasible region (Phase 1) before optimizing the objective function while maintaining feasibility (Phase 2) [89]. This approach aids an optimization process in beginning from a well-conditioned, constraint-satisfying state. This results in improved convergence properties and computational efficiency. To implement this within AGHF, we formulate two distinct Action Functionals – one for each phase.

In Phase 1, the goal is to guide the initial trajectory into the feasible set while promoting dynamic feasibility. Leveraging Definition 14, we construct an Action functional that primarily penalizes constraint violations to achieve this, with an additional term to encourage

dynamic feasibility. In Phase 2, we apply the generalized AGHF Action Functional with constraints (18) to maintain feasibility while minimizing the objective cost.

We first introduce the form of the Phase 1 Action Functional:

Definition 20. *Let the Phase 1 Action Functional for a general state constraint $g_j(x_s(t), \dot{x}_s(t))$ be defined by*

$$\mathcal{A}(x_s) = \int_0^T k_d \|\dot{x}_{P1}(s, t) - x_{P2}(s, t)\|_2^2 + \sum_{j \in \mathcal{J}} b(g_j(x_s(t), \dot{x}_s(t))) \quad (3.18)$$

and let the corresponding AGHF RHS be expressed as:

$$\begin{aligned} \frac{\partial x}{\partial s}(t, s) = & I_{2N \times 2N} \left(\frac{d}{dt} \frac{\partial L_d}{\partial \dot{x}_s}(x_s(t), \dot{x}_s(t)) - \frac{\partial L_d}{\partial x_s}(x_s(t), \dot{x}_s(t)) \right) \\ & + \sum_{j \in \mathcal{J}} \left(\frac{d}{dt} \frac{\partial b(g_j(x_s(t), \dot{x}_s(t)))}{\partial \dot{x}_s(t)} - \frac{\partial b(g_j(x_s(t), \dot{x}_s(t)))}{\partial x_s(t)} \right), \end{aligned} \quad (3.19)$$

where $I_{2N \times 2N}$ is the identity matrix and

$$L_d(x_s(t), \dot{x}_s(t)) = (\dot{x}_s(t) - F_d(x_s(t)))^T \begin{bmatrix} k_d I_{N \times N} & 0_{N \times N} \\ 0_{N \times N} & 0_{N \times N} \end{bmatrix} (\dot{x}_s(t) - F_d(x_s(t))). \quad (3.20)$$

Note that one can extend this definition to incorporate input constraints $h_i(u_s(t))$. Phase 2 utilizes the AGHF RHS from (18), initializing the trajectory evolution with the solution obtained from Phase 1. Given a sufficiently large penalty parameter k_{cons} and evolution time s_{max} , where $k_{cons} > k_d$, the Phase 1 trajectory evolves towards a trajectory that minimizes constraint violation while promoting some dynamic feasibility.

Phase 2 is initialized with the final trajectory from Phase 1 and proceeds by optimizing the objective function while maintaining feasibility. As shown in Section 3.4.1, the constraint penalty term ensures that for sufficiently large k_{cons} and s_{max} , where $k_{cons} > k_d$ if the trajectory starts in the feasible set, it will remain within the feasible set throughout the evolution. Consequently, as the AGHF PDE progresses, the trajectory is driven toward a minimizer of the specified cost function that is dynamically feasible while satisfying all constraints.

3.6 Experiments and Results

This section evaluates the speed and performance of BLAZE on a variety of scenarios and robot platforms. We begin by explaining the experimental setup. Next, we evaluate the

scalability of BLAZE when compared to the state of the art trajectory optimization algorithm. Then, we investigate the performance of the methods in scenarios where obstacles are present and then describe the translation of these results onto a real-world platform. We conclude the section by summarizing the different evaluations.

3.6.1 Implementation of BLAZE

Our numerical implementation of BLAZE follows the pseudospectral MOL approach adopted in [3] which is summarized in Section 3.3.2. Throughout these experiments, for BLAZE we choose the following activation function $S(g_j(x, \dot{x}))$ for state constraints that satisfies the properties highlighted in Definition 17:

$$S(g_j(x, \dot{x})) = \frac{1}{2} + \frac{1}{2} \tanh(c_{\text{cons}} \cdot g_j(x, \dot{x})) \quad (3.21)$$

where c_{cons} is a hyper-parameter that determines how fast $S(g_j(x))$ transitions from 0 to 1, once the constraint is violated. Note that we apply the same activation function for the input constraints as well, where in that case the activation is given by $S(g_j(u))$ instead. Note that the Constrained Lagrangian we choose in Phase 1 of our implementation requires that we satisfy input box-constraints for each of the examples described below. Whereas in Phase 2, we try minimize the square of the control effort. In either instance, note that the Lagrangian satisfies the coercive requirement due to the definition of the extracted control input (Definition 13).

3.6.2 Experimental Setup

We evaluate BLAZE and compare it to the DDP methods Crocoddyl [79] and Aligator [51], and RAPTOR [124] which uses **IPOPT** [113] as its backend optimizer. Note we compare against these methods because they each are able to perform optimal control while considering the full order dynamics of a robot. We solve a number of fixed time and fixed initial and final state optimization problems. In all the problems, we enforce input constraints and state constraints and in problems with obstacles we also enforce obstacle avoidance constraints. All the experiments were run on an Ubuntu 22.04 machine with an AMD EPYC 7742 64-Core @ 256x 2.25GHz CPU.

3.6.2.1 Selection of Parameters

The solution produced by any of the mentioned trajectory optimization algorithms depends heavily on the selection of initial guess and on the parameters of the solver. This section

explains the rationale used to choose the parameters.

For most experiments we use the parameters reported in the grid search results of [3] with the following caveat. When the experiments are qualitatively different from the ones considered in that paper (e.g., novel constraints or types of obstacles), we sequentially vary the parameters until finding parameters those that yield a feasible solution. If the previous procedure fails, we run a small grid search around the parameters reported in [3]. Appendix B.4 summarizes all the parameters chosen for the different methods for all the experiments and explain the meaning of each.

3.6.2.2 Evaluating Success or Failure

Each tested numerical method generates an open-loop control input, $u^* : [0, T] \rightarrow \mathbb{R}^m$ and corresponding system trajectory, $x^* : [0, T] \rightarrow \mathbb{R}^{2N}$. However, whether the robot can accurately track these computed solutions in practice is inobvious. Thus, to fairly evaluate constraint satisfaction and account for integration errors when applying the open-loop control, we integrate forward using the following feedback controller:

$$u_{fb}(t) = u^*(t) + k_p(x_{P1}^*(t) - q(t)) + k_v(x_{P2}^*(t) - \dot{q}(t)), \quad (3.22)$$

where $k_p = k_v = 100$ for each experiment, and $q(t)$ and $\dot{q}(t)$ represent the robot's position and velocity at time t , respectively. The system dynamics are then integrated forward using (3.22) to assess tracking accuracy and constraint adherence.

In obstacle-free experiments, a solution is considered *successful* if the infinity norm of the error between the forward-integrated final state and x_f is below a fixed threshold $\epsilon = 0.05$ and the forward-integrated solution does not violate any of the state or input constraints. In experiments with obstacles, a trial is considered *successful* if it meets the previous criteria and, additionally, the joint positions of the forward integrated solution, sampled at a time resolution of $\Delta t = 10^{-2}$ s, remain outside of the obstacles at each joint frame.

3.6.2.3 Aligning Initial Guesses

For all the experiments, each method is given the same initial guess x_{init} to ensure a fair comparison across methods. Because Crocoddyl and Aligator, both DDP-based methods, optimize over control inputs as decision variables, we provide them with an initial guess of $u_{init} : [0, T] \rightarrow \mathbb{R}^N$. This is obtained via inverse dynamics using position $q : [0, T] \rightarrow \mathbb{R}^N$, velocity $\dot{q} : [0, T] \rightarrow \mathbb{R}^N$ and acceleration $\ddot{q} : [0, T] \rightarrow \mathbb{R}^N$ extracted from x_{init} . By default x_{init} provides q and $\dot{q}$. To compute the acceleration trajectory $\ddot{q} : [0, T] \rightarrow \mathbb{R}^N$, we fit a Chebyshev polynomial to x_{init} and apply the Chebyshev differentiation matrix $D : \mathbb{R}^{p+1} \rightarrow$

$\mathbb{R}^{p+1}$ as described in [107, (21.2)]. Specifically, the differentiation matrix D approximates the derivative of the fitted polynomial, allowing the computation of $\ddot{q}$ from the polynomial coefficients. The resulting trajectories q , $\dot{q}$ and $\ddot{q}$ are then utilized in the inverse dynamics computation to generate u_{init} . In contrast, RAPTOR formulates trajectory optimization with Bezier polynomial coefficients as decision variables. To ensure a consistent initial guess, we fit a Bezier polynomial to x_{init} . x_{init} and use the fitted polynomial’s coefficients as RAPTOR’s initial input.

3.6.3 BLAZE Scalability with Increasing System Dimension

This section compares BLAZE, Crocoddyl, Aligator and RAPTOR in solving fixed-time trajectory optimization problems with fixed initial and final states across multiple systems. The goal is to understand how each method’s solve time scales with increasing system dimension. In each of these trials input and joint limits are enforced, but there are no obstacle avoidance constraints. The objective function is formulated as minimizing control effort along the trajectory.

We do this evaluation for a 1-, 2-, 3-, 4-, 5-link pendulum model, a 7DOF Kinova Gen3 arm, a double 7DOF Kinova Gen3 arm and a triple 7DOF Kinova Gen3. Figure 3.2 shows how each of the aforementioned methods scale with increasing N . We see that BLAZE is able to scale more favorably than the other methods and solves faster than all the other methods for most of the evaluated robot systems with $N > 1$. Refer to tables B.6, B.2 and B.5 for the specific parameter values used for BLAZE, Aligator and Crocoddyl respectively.

3.6.4 Kinova Gen3 Sphere Obstacle Avoidance

This section compares BLAZE and Aligator in solving multiple fixed-time trajectory optimizations to get the Kinova Gen3 from some initial state to some final state while avoiding sphere obstacles. Note, we do not run Crocoddyl or RAPTOR in these experiments, because they did not have any examples doing sphere obstacle avoidance. In these experiments, we design x_{init} such that it starts in collision, to evaluate how effective the proposed method is in moving the initial guess out of constraint violation during Phase 1 before solving Phase 2. Each of these methods is ran to convergence at a small optimality tolerance, ensuring fully converged solutions. We evaluate each of these methods on 20 different scenarios where either the obstacles are randomly placed and x_{init} is generated to be in collision. For 10 of these scenarios, there are 5 obstacles, while for the other 10 there are 10 obstacles. The objective function is formulated to minimize control effort along the trajectory. For each method, we gave a 60s time budget to be able to generate a trajectory. Being unable to

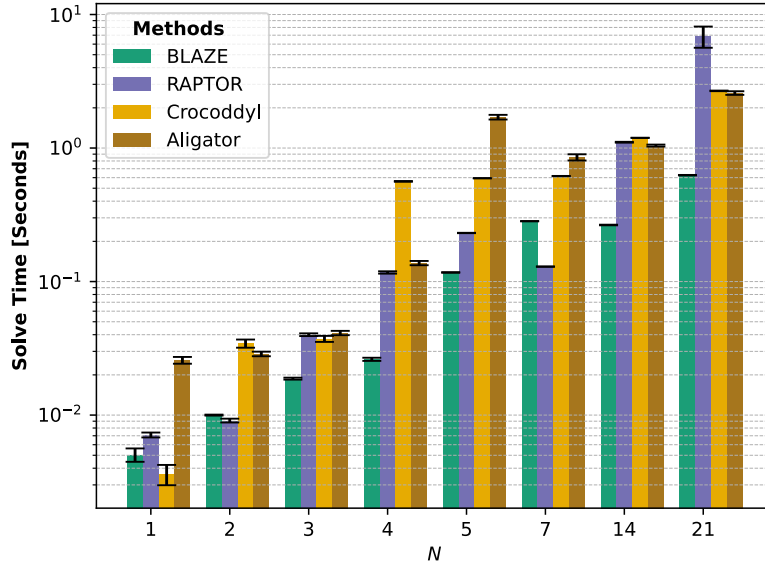


Figure 3.2: A bar plot comparing the mean solve times for four different trajectory optimization algorithms: BLAZE, RAPTOR, Crocoddyl and Aligator. For $N \leq 5$ the results correspond to the 1-5 link pendulum, $N = 7$ corresponds to the Kinova Gen3 Arm, $N = 14$ corresponds to a bimanual system of two Kinova Gen3 Arms, $N = 21$ corresponds to a trimanual system of three Kinova Gen3 Arms. Each experiment was run ten times. Overall, we see that BLAZE shows better scalability and solve times than the other methods as the system dimension increases.

generate the trajectory within that time or not satisfying the earlier success criteria from Section 3.6.2.2 is counted as an unsuccessful trial.

As illustrated in Table 3.1, BLAZE is able to generate trajectories much faster than Aligator when both methods are ran to full convergence, and is able to generate trajectories that do not collide with obstacles for all the scenarios. To explore how Aligator fared when it was allowed to return suboptimal solutions we reran the same experiment over the same scenarios but allow Aligator to return its suboptimal solutions after each iteration. Table 3.2 shows that BLAZE converges in less time than Aligator’s first iteration and achieves a lower objective cost even compared to Aligator’s third iteration.

3.6.5 Kinova Gen3 Hard Task-Based Scenarios

This section compares BLAZE, PHLAME [3] (which is a version of BLAZE that does not utilize a Phase1-Phase2 algorithm), Aligator, and RAPTOR under a similar setup as the previous subsection, with two key differences. First, obstacles are now cuboids arranged to

Method Name	BLAZE	Aligator
Success Rate [%]	100	65
Objective Cost	738.7 ± 157.1	635.9 ± 108.81
Solve Time [s]	0.75 ± 0.27	27 ± 11.06

Table 3.1: Comparison of Success Rate, Objective Cost and Solve Time for Kinova Gen3 trajectory optimization experiments with obstacles. Note that the Objective Cost and Solve Time numbers are just presented for scenarios where both methods were successful.

Method Name	BLAZE	Aligator it = 1	Aligator it = 3
Objective Cost	733.2 ± 166.3	884.8 ± 140.4	775.3 ± 120.5
Solve Time [s]	0.84 ± 0.33	0.91 ± 0.85	1.45 ± 0.84

Table 3.2: Objective Cost and Solve Time for BLAZE compared with Aligator after its 1st and 3rd iterations on Kinova Gen3 obstacle-avoidance tasks, where Aligator is permitted to return intermediate suboptimal solutions. Results are aggregated over all 20 scenarios, rather than only successful trials as in Table 3.1 due to a large proportion of the Aligator optimizations after the first iteration not satisfying the success criteria.

resemble more task-oriented scenarios for manipulators that would be deployed in domestic and industrial settings. These scenarios involve actions such as reaching into shelves, retracting arms out of shelves and under shelves, and placing items in confined bins, reflecting real-world manipulation tasks. Second, the time limit for each method is extended to 120s to accommodate scenarios with a greater number of obstacles. For these experiments each method must generate a 2-second long trajectory, which is a relatively short time to execute many of these scenarios due to how far the robot must move. This time constraint forces the methods to produce dynamic solutions while simultaneously satisfying state and input constraints, as well as obstacle avoidance, making the scenarios challenging. A trial is considered unsuccessful if a method fails to generate a trajectory within the allotted time or does not satisfy the earlier success criteria from Section 3.6.2.2.

Figure 3.3 shows the evolution of the Action Functional versus the AGHF evolution parameter s for all the hard task-based scenario experiments. Each curve corresponds to one scenario and exhibits the Action functional decreasing and fully converging to a steady-state value.

Table 3.3 summarizes the performance of the three optimal control algorithms for the 10 different realistic scenarios. In this table “F” denotes that a scenario either timed out after 120s or was considered a failure as per the success criteria introduced in Section 3.6.2.2. BLAZE demonstrates a higher success rate, completing 62.5% of trials compared to 25% for Aligator, 12.5% for RAPTOR and 0% for PHLAME. In these kinds of scenarios, PHLAME struggles as the initial trajectories given to all solvers violates the obstacle constraints, mak-

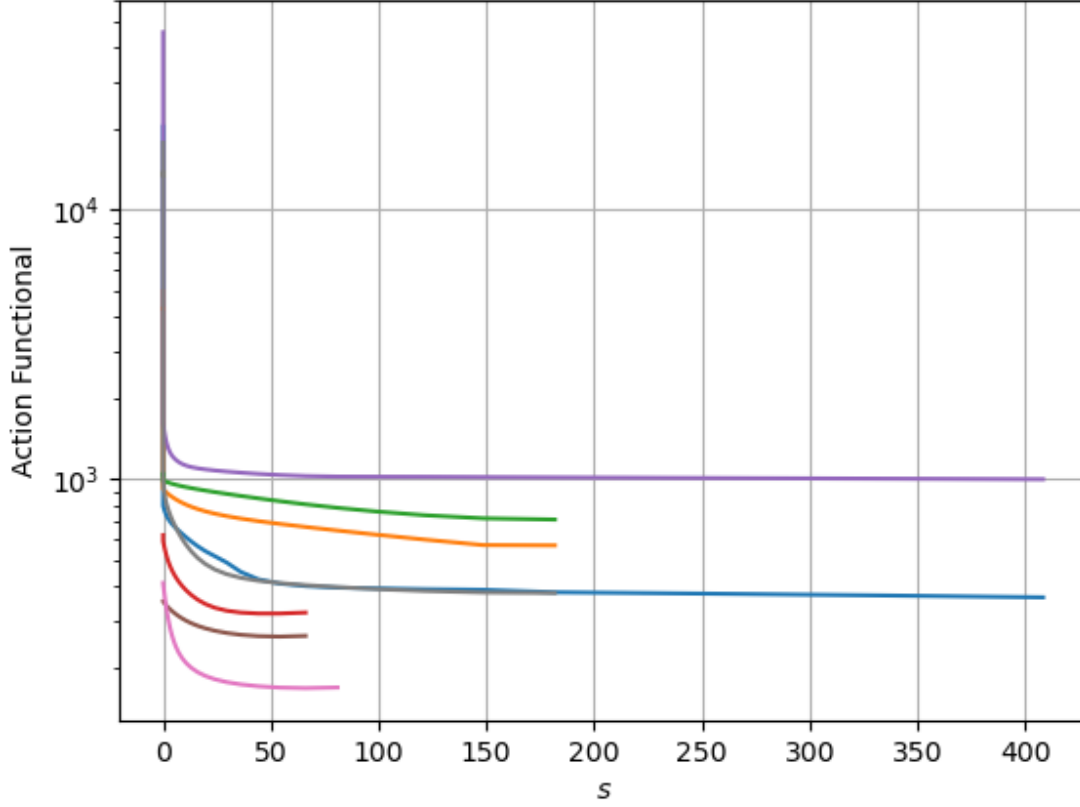


Figure 3.3: A plot showing the evolution of the Action Functional versus the AGHF evolution parameter s for each of the the hard 2-second horizon task-based scenario experiments. The plot shows that all experiments ran to full convergence, each of them truncating at the particular s where the action functional converged to a minimum.

ing it difficult for PHLAME to converge to a good solution in the given time and further highlighting the motivation for designing a Phase 1 - Phase 2 algorithm. Figure 3.4 illustrates a solution generated by BLAZE for one of the task-based scenarios.

Scenario	Objective Cost	Solve Time [s]
1	F	F
	F	F
	F	F
	F	F
2	777.45	4.03 $\pm$ 0.08
	291.35	57.21 $\pm$ 0.13
	F	F
	F	F
3	887.54	3.73 $\pm$ 0.06
	F	F
	F	F
	F	F
4	F	F
	F	F
	F	F
	F	F
5	F	F
	F	F
	F	F
	F	F
6	269.03	4.34 $\pm$ 0.06
	F	F
	354.64	0.32 $\pm$ 0.00
	F	F
7	166.99	6.92 $\pm$ 0.04
	187.68	68.46 $\pm$ 0.19
	F	F
	F	F
8	381.07	7.57 $\pm$ 0.06
	F	F
	F	F
	F	F

Table 3.3: Detailed comparison of Objective Cost and Solve Time for the 2-second horizon task-based scenarios. All the problems consider input, state and cuboid obstacle constraints. BLAZE is depicted in **green**, Aligator is depicted in **gold**, RAPTOR is depicted in **purple** and PHLAME (which is a version of BLAZE that does not utilize a Phase1-Phase2 algorithm) is depicted in **pink**. In this table "F" denotes that a scenario was considered a failure as per the definition introduced in Section 3.6.2.2.

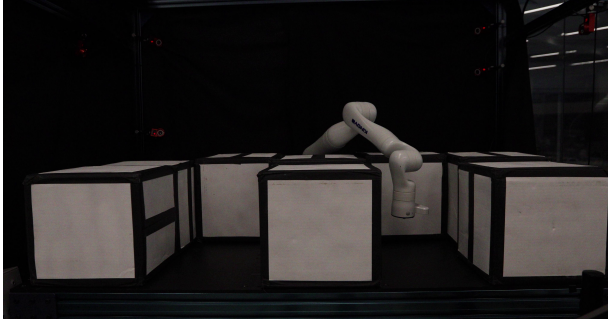


Figure 3.4: This figure shows a visualization of BLAZE’s solution for one of the 2-second horizon task-based scenarios (scenario 3). In each subfigure, the color gradient illustrates the evolution in time where the start configuration is in the darkest shade and the end configuration in the lightest. The initial trajectory for the scenario is shown in the top image in yellow and collides with the obstacles along the path. The proposed algorithm is able to push this initial guess out of collision and generate the optimal collision-free solution shown in green in 3.73s for this scenario, whereas the other comparison methods cannot find a solution within the allotted time for this scenario.

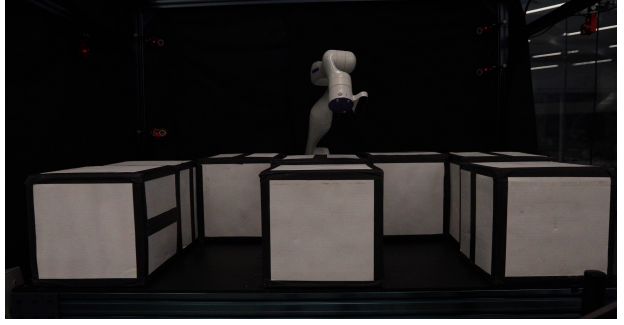
3.6.6 Hardware Experiments

We demonstrate the performance of BLAZE on hardware on the Kinova Gen3 robot on a number of challenging obstacle avoidance scenarios, where the robot must navigate from some initial position to some final position while adhering to the robot’s state and input constraints. Here, the dynamic feasibility of BLAZE as well as its ability to generate trajectories that satisfy the real hardware constraints is critical to enabling the robot to successfully execute the generated trajectory. On the robot, we leverage a passivity-based controller to track the trajectories we generate.

The supplemental material includes videos showcasing various challenging scenarios, such as a rapid pull-and-place maneuver where the robot retracts its arm from a simulated shelf and swiftly places it in a bin (hardware scenario 1, solved in 3.1 seconds); a fast retraction from a confined space while avoiding obstacles (hardware scenario 2, solved in 2.74 seconds); and a precise pick-and-place trajectory requiring navigation through tight grasp points (hardware scenario 3, solved in 2.45 seconds, as shown in Figure 3.5). Each of these trajectories is executed within a strict 2-second duration, requiring the robot to move dynamically while adhering to state and input constraints. These experiments highlight BLAZE’s ability to rapidly generate dynamically feasible trajectories that fully account for the robot’s full-order



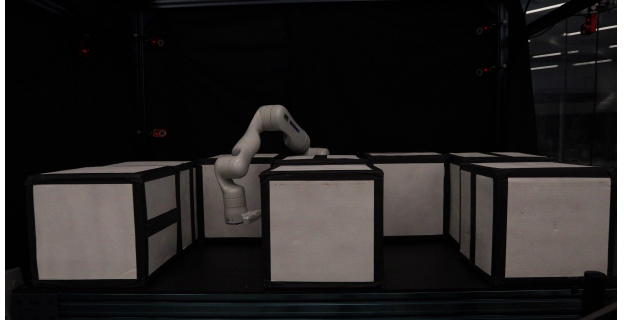
(a) Still frame at the initial time $t = 0\text{s}$.



(b) Still frame at time $t = 0.67\text{s}$.



(c) Still frame at time $t = 1.34\text{s}$.



(d) Still frame at the final time $t = 2.0\text{s}$.

Figure 3.5: This figure shows a series of still frames of the Kinova arm following the trajectory generated by BLAZE. Each subfigure depicts the arm at a different point in time. The initial guess collides with the obstacle. The proposed algorithm is able to push this initial guess out of collision and generate the optimal collision-free solution in 2.45s, whereas Aligator does it in 57.21s.

dynamics, demonstrating its effectiveness in real-world task execution.

3.7 Conclusion

This work proposes BLAZE, a generalized AGHF-based trajectory optimization framework that generalizes the AGHF PDE methods to arbitrary cost functions, significantly expanding the range of feasible trajectories. By leveraging the AGHF PDE, this method evolves an initial, potentially infeasible trajectory into a dynamically feasible solution while optimizing a specified objective function, allowing for rapid trajectory generation. A key contribution of this work is the Phase1-Phase2 framework, made possible by the generalized cost function formulation. This Phase1-Phase2 framework addresses a limitation of previous AGHF approaches – requiring constraint-satisfying initial guesses to ensure fast convergence. Phase1 leverages the generalized cost function to drive the trajectory into the feasible set before

Phase2 optimizes the objective while maintaining feasibility. This allows the initial guess to violate constraints while ensuring rapid convergence to a valid solution. Additionally, a new method for enforcing input constraints within AGHF is introduced, enabling trajectory optimization with realistic actuation limits without increasing the state dimension, as was required in previous AGHF-based approaches. Simulations and hardware experiments demonstrate that BLAZE can generate trajectories for high-dimensional robotic systems under multiple constraints, with constraint-violating initial guesses faster than state-of-the-art trajectory optimization methods.

3.8 Limitations

BLAZE has its limitations: similar to other AGHF formulations because the action functional (3.11) incorporates dynamic constraints through a penalty term, the AGHF solution does not minimize the objective function as dramatically as other optimal control methods. In particular, traditional optimal control methods may construct lower cost trajectories; however, in practice, because generated trajectories satisfy all constraints (including input constraints) this may not be too restrictive. In fact if one’s objective is to generate trajectories rapidly that are dynamically feasible, BLAZE may be the right approach. One other difficulty with BLAZE stems from its requirement to balance between the constraint penalty, k_{cons} , and the dynamic feasibility penalty, k , which can require some tuning.

CHAPTER 4

Risk-Aware, Real-Time, Reachability-Based Motion Planning

4.1 Introduction

For mobile robots to safely operate in unstructured environments, they must be able to sense their environments and develop plans to dynamically react to changes while avoiding obstacles. Unfortunately, it is challenging to accurately and precisely estimate and predict an obstacle’s movement. For mobile robots to operate robustly, the uncertainty within these estimations and predictions must be accounted for while generating motion plans. Various approaches to account for this uncertainty have been proposed in the literature, but these methods either (1) have difficulty being utilized in real-time, (2) make strong assumptions on the class of probability distributions used to model the uncertainty, or (3) generate motion plans that are overly conservative and thereby restrict motion. This chapter presents two algorithms for **risk-aware, real-time motion planning** while addressing each of the aforementioned challenges. The first is **RADIUS**¹²: **R**isk-Aware, **r**ead-time **t**rajectory **D**esign **I**n **U**ncertain **S**cenarios (introduced in Figure 4.1) and the second algorithm is **RADIUS 2.0: DIAMETER**, which builds on RADIUS, but proposes an alternate way to compute the risk over-approximation.

4.1.1 Related Work

We first summarize related algorithms for safe motion planning under uncertainty. Approaches to address this problem can be categorized as deterministic or stochastic. The deterministic approaches often assume some bounded level of uncertainty and solve for motion plans that are robust to this bounded uncertainty. Reachability-based methods [60, 70, 29]

¹Published in Robotics: Science and Systems, Daegu, Republic of Korea, July 2023

²Code found at <https://roahmlab.github.io/RADIUS/>

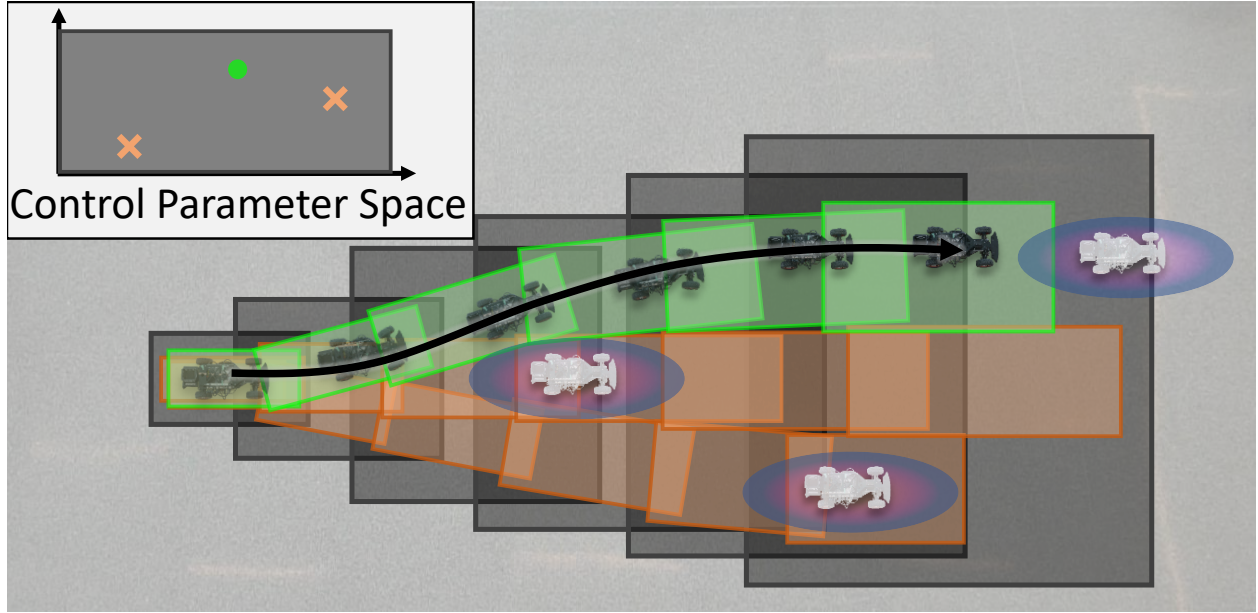


Figure 4.1: An illustration of the motion planning framework RADIUS that is developed in this chapter. RADIUS first performs offline reachability analysis using a closed-loop full-order vehicle dynamics to construct a series of control-parameterized, zonotope reachable sets (shown as dark gray boxes) that over-approximate all possible behaviors of the ego vehicle over the planning horizon. During online planning, given some user-defined risk of collision (ϵ) for the motion plan, RADIUS constructs a trajectory by solving an optimization problem that selects subsets of pre-computed zonotope reachable sets that are certified to have a no greater than ϵ risk of colliding with any obstacles. In this figure, the moving obstacles are shown in white and the $3\text{-}\sigma$ regions of the corresponding probability distributions for the obstacle locations are shown as the purple and blue ellipses where the probability density from low to high is illustrated from blue to purple. The subsets of the dark gray zonotope reachable sets corresponding to the trajectory parameter shown in green ensure a collision-free path that is guaranteed to have a no greater than ϵ risk of colliding with all obstacles, while the two trajectory parameters and their corresponding reachable sets shown orange may have a greater than ϵ risk of collision with the moving obstacles.

for instance, over-approximate the uncertain regions using polynomial level sets or polytopes and plan paths that do not intersect with these regions. As we illustrate in this chapter, such methods may be overly conservative as they must over-approximate the uncertain region to include events that may have an exceedingly small probability of occurring to ensure safety.

Stochastic methods incorporate probabilistic information about the environment to reduce the level of conservativeness of their motion plans. Many of these methods leverage chance constraints that allow for some user-specified amount of constraint violation. For instance, sampling-based algorithms for chance-constrained motion planning apply numerical integration techniques to compute the risk of collision between the agent and obstacles in the environment [52, 8]. In this context, random samples are drawn from the uncertain

region, and the number of samples that cause a collision is used to approximate the risk of collision. These methods are simple to implement, but can be time-consuming as they require large numbers of samples to converge to a good estimate of the risk of collision. To address these limitations, moment-based methods upper bound the risk of collision using moments of the probability distribution [115, 114, 15]. Recent works leverage MPC and SOS programming in conjunction with these moment-based upper bounds to perform motion planning. Unfortunately, these methods can be conservative as the bound must be valid for any distribution with the given moments. Coherent risk measures like conditional value-at-risk (CVaR) and Entropic value-at-risk (EVaR) have been used to regulate safety by limiting the expectation of the distance to the safe region using a worst-case quantile representation of a distribution [42, 5, 22]. However, CVaR constraints are difficult to construct directly due to the computationally expensive multi-dimensional integration that is required. As a result, they are often approximated using sampling methods, which as mentioned earlier can be computationally expensive. Branch-and-bound methods like the Chance-Constrained Parallel Bernstein Algorithm (CCPBA) use reachability-based methods in conjunction with a branch-and-bound style algorithm to compute tight bounds for the risk of collision [111]. To compute these probabilities they assume that they have access to a cumulative distribution function, *a priori*, that can be evaluated efficiently during online optimization. However, such an assumption may not hold for arbitrary distributions.

4.1.2 Chapter Contributions

This chapter makes the following contributions. First, it provides a novel, parallelizable, closed-form over-approximation of the risk of collision given an arbitrary probability distribution with a twice-differentiable density function (Section 4.6.1). Second, it describes the analytical derivative of the probability of collision’s over-approximation with respect to a control parameter (Section 4.6.3) which is used to speed up the optimization problem solved during planning. Third, this chapter presents a general, real-time risk-aware motion planning framework that guarantees robot safety up to a user-specified risk threshold over time intervals rather than just at discrete time instances (Section 4.6.4). Fourth, this chapter demonstrates RADIUS in simulation and on hardware and compares its performance to CCPBA, and Cantelli MPC on a variety of scenarios (Section 4.7). Lastly, this chapter presents an alternate parallelizable, closed-form over-approximation of the risk of collision called DIAMETER (Section 4.8) that in certain conditions yields a tighter risk over-approximation to that presented in Section 4.6.1. The rest of this chapter is organized as follows: Section 4.2 gives necessary notations for the chapter. Section 4.3 presents the

parameterized vehicle dynamics and environment representation. Section 4.4 describes obstacle uncertainty, risk-aware vehicle safety and online planning. Section 4.5 discusses offline reachability analysis of the vehicle dynamics. Section 4.9 concludes the chapter.

4.2 Notation

This section formalizes notations used throughout the chapter. Sets and subspaces are typeset using calligraphic font. Subscripts are primarily used as an index or to describe a particular coordinate of a vector. Let $\mathbb{R}$ and $\mathbb{N}$ denote the spaces of real numbers and natural numbers respectively. The Minkowski sum between two sets $\mathcal{A}$ and $\mathcal{A}'$ is $\mathcal{A} \oplus \mathcal{A}' = \{a + a' \mid a \in \mathcal{A}, a' \in \mathcal{A}'\}$. Given a set $\mathcal{A}$, denote its power set as $P(\mathcal{A})$. Given vectors $\alpha, \beta \in \mathbb{R}^n$, let $[\alpha]_i$ denote the i -th element of α , let $\mathbf{diag}(\alpha)$ denote the diagonal matrix with α on the diagonal, and let $\mathbf{int}(\alpha, \beta)$ denote the n -dimensional box $\{\gamma \in \mathbb{R}^n \mid [\alpha]_i \leq [\gamma]_i \leq [\beta]_i, \forall i = 1, \dots, n\}$. Given arbitrary matrix $A \in \mathbb{R}^{n \times n}$, and let $|A|$ be the determinant of A . Let $\mathbf{Prob}(\mathbf{E})$ denote the probability of occurrence of some event $\mathbf{E}$, and let $\mathbf{rotate}(a)$ denote the 2-dimensional rotation matrix $\begin{bmatrix} \cos(a) & -\sin(a) \\ \sin(a) & \cos(a) \end{bmatrix}$ for arbitrary $a \in \mathbb{R}$.

Next, we introduce a subclass of polytopes, called zonotopes, that are used throughout this chapter:

Definition 21. A zonotope $\mathcal{Z}$ is a subset of $\mathbb{R}^n$ defined as

$$\mathcal{Z} = \left\{ x \in \mathbb{R}^n \mid x = c + \sum_{k=1}^{\ell} \beta_k g_k, \quad \beta_k \in [-1, 1] \right\} \quad (4.1)$$

with center $c \in \mathbb{R}^n$ and ℓ generators $g_1, \dots, g_\ell \in \mathbb{R}^n$. For convenience, we denote $\mathcal{Z}$ by $\langle c, G \rangle$ where $G = [g_1, g_2, \dots, g_\ell] \in \mathbb{R}^{n \times \ell}$.

Note that an n -dimensional box is a zonotope because

$$\mathbf{int}(\alpha, \beta) = \langle \frac{1}{2}(\alpha + \beta), \frac{1}{2}\mathbf{diag}(\beta - \alpha) \rangle. \quad (4.2)$$

By definition the Minkowski sum of two arbitrary zonotopes $\mathcal{Z}_1 = \langle c_1, G_1 \rangle$ and $\mathcal{Z}_2 = \langle c_2, G_2 \rangle$ is still a zonotope as $\mathcal{Z}_1 \oplus \mathcal{Z}_2 = \langle c_1 + c_2, [G_1, G_2] \rangle$. Note that one can define the multiplication of a matrix A of appropriate size with a zonotope $\mathcal{Z} = \langle c, G \rangle$ as

$$A\mathcal{Z} = \left\{ x \in \mathbb{R}^n \mid x = Ac + \sum_{k=1}^{\ell} \beta_k Ag_k, \quad \beta_k \in [-1, 1] \right\}. \quad (4.3)$$

Note that AZ is equal to the zonotope $\langle Ac, AG \rangle$.

4.3 Preliminaries

The goal of this work is to plan trajectories for a mobile robot to navigate through environments with uncertainty in the locations of obstacles while having guarantees on the risk of collision. We illustrate the proposed method on an autonomous vehicle in this work. This section discusses the vehicle model, parameterized trajectories, and lastly the environment.

4.3.1 Open Loop Vehicle Dynamics

4.3.1.1 High-Speed Model

This work adopts the Front-Wheel-Drive vehicle model from [70], and can be generalized to All-Wheel-Drive and Rear-Wheel-Drive vehicle models. Let the ego vehicle's states at a time t be given by $z^{\text{hi}}(t) = [x(t), y(t), h(t), u(t), v(t), r(t)]^\top \in \mathbb{R}^6$, where $x(t)$ and $y(t)$ are the position of the ego vehicle's center of mass in the world frame, $h(t)$ is the heading of the ego vehicle in the world frame, $u(t)$ and $v(t)$ are the longitudinal and lateral speeds of the ego vehicle in its body frame, and $r(t)$ is the yaw rate of the vehicle center of mass. To simplify exposition, we assume vehicle weight is uniformly distributed and ignore the aerodynamic effect while modeling the flat ground motion of the vehicles by the following dynamics:

$$\dot{z}^{\text{hi}}(t) = \begin{bmatrix} u(t) \cos h(t) - v(t) \sin h(t) \\ u(t) \sin h(t) + v(t) \cos h(t) \\ r(t) \\ \frac{1}{m}(F_{\text{xf}}(t) + F_{\text{xr}}(t)) + v(t)r(t) + \Delta_u(t) \\ \frac{1}{m}(F_{\text{yf}}(t) + F_{\text{yr}}(t)) - u(t)r(t) + \Delta_v(t) \\ \frac{1}{I_{\text{zz}}}(l_{\text{f}}F_{\text{yf}}(t) - l_{\text{r}}F_{\text{yr}}(t)) + \Delta_r(t) \end{bmatrix}, \quad (4.4)$$

where l_{f} and l_{r} are the distances from the center of mass to the front and back of the vehicle, I_{zz} is the vehicle's moment of inertia, and m is the vehicle's mass. Note: l_{f} , l_{r} , I_{zz} and m are all assumed to be known constants. The tire forces along the longitudinal and lateral directions of the vehicle at time t are $F_{\text{xi}}(t)$ and $F_{\text{yi}}(t)$ respectively, where the 'i' subscript can be replaced by 'f' for the front wheels or 'r' for the rear wheels. $\Delta_u, \Delta_v, \Delta_r$ are modeling error signals that are unknown and account for imperfect state estimation and tire models. To ensure the system is well-posed (i.e., its solution exists and is unique), we make the following assumption:

Assumption 22. $\Delta_u, \Delta_v, \Delta_r$ are all square-integrable functions and are bounded (i.e., there exist real numbers $M_u, M_v, M_r \in [0, +\infty)$ such that $\|\Delta_u(t)\|_\infty \leq M_u$, $\|\Delta_v(t)\|_\infty \leq M_v$, $\|\Delta_r(t)\|_\infty \leq M_r$ for all t).

Note that $\Delta_u, \Delta_v, \Delta_r$ can be computed using real-world data [70, Section IX-C]. In this work, we assume linear tire models and assume that we can directly control the front tire forces. Such assumptions are validated in [70, Section V.B and VIII.B]. Additionally, we treat rear tire forces as observed signals.

4.3.1.2 Low-Speed Model

When the vehicle speed lowers below some critical value $u^{\text{cri}} > 0$, applying the model described in (4.4) becomes intractable as explained in [70, Section III-B]. As a result, in this work when $u(t) \leq u^{\text{cri}}$, the dynamics of a vehicle are modeled using a steady-state cornering model [40, Chapter 6], [21, Chapter 10]. Note that the critical velocity u^{cri} can be found according to [57, (5) and (18)].

The steady-state cornering model or low-speed vehicle model is described using four states, $z^{\text{lo}}(t) = [x(t), y(t), h(t), u(t)]^\top \in \mathbb{R}^4$ at time t . This model ignores transients on lateral velocity and yaw rate. Note that the dynamics of x , y , h and u in the low-speed model are the same as in the high-speed model (4.4); however, the steady-state cornering model describes the yaw rate and lateral speed as

$$r^{\text{lo}}(t) = \frac{\delta(t)u(t)}{l + C_{\text{us}}u(t)^2}, \quad v^{\text{lo}}(t) = l_{\text{r}}r^{\text{lo}}(t) - \frac{ml_{\text{f}}}{\bar{c}_{\text{or}}l}u(t)^2r^{\text{lo}}(t) \quad (4.5)$$

with understeer coefficient

$$C_{\text{us}} = \frac{m}{l} \left(\frac{l_{\text{r}}}{\bar{c}_{\text{af}}} - \frac{l_{\text{f}}}{\bar{c}_{\text{or}}} \right), \quad (4.6)$$

where $\delta(t)$ is the front tire steering angle at time t , $\bar{c}_{\text{af}}$ and $\bar{c}_{\text{or}}$ are cornering stiffness of the front and rear tires respectively.

As we describe in Section 4.3.3, the high-speed and low-speed models can be combined together as a hybrid system to describe the vehicle behavior across all longitudinal speeds. In short, when u transitions past the critical speed u^{cri} from above at time t , the low speed model's states are initialized as:

$$z^{\text{lo}}(t) = \pi_{1:4}(z^{\text{hi}}(t)) \quad (4.7)$$

where $\pi_{1:4} : \mathbb{R}^6 \rightarrow \mathbb{R}^4$ is the projection operator that projects $z^{\text{hi}}(t)$ onto its first four dimensions via the identity relation. If u transitions past the critical speed from below at

time t , the high-speed model's states are initialized as

$$z^{\text{hi}}(t) = [z^{\text{lo}}(t)^\top, v^{\text{lo}}(t), r^{\text{lo}}(t)]^\top. \quad (4.8)$$

4.3.2 Trajectory Parameterization

In this work, each trajectory plan is specified over a compact time interval of a fixed duration t_f . Because RADIUS performs receding-horizon planning, we make the following assumption about the time available to construct a new plan:

Assumption 23. *During each planning iteration starting from time t_0 , the ego vehicle has t_{plan} seconds to find a control input that is applied during the time interval $[t_0 + t_{\text{plan}}, t_0 + t_{\text{plan}} + t_f]$, where $t_f \geq 0$ is some user-specified constant. In addition, the vehicle state at time $t_0 + t_{\text{plan}}$ is known at time t_0 .*

This assumption requires RADIUS to generate plans in real-time. This means that the ego vehicle must create a new plan before it finishes executing its previously planned trajectory.

In each planning iteration, RADIUS chooses a *desired trajectory* to be followed by the ego vehicle. The desired trajectory is chosen from a pre-specified continuum of trajectories, with each uniquely determined by an n_p -dimensional *trajectory parameter* $p \in \mathcal{P} \subset \mathbb{R}^{n_p}$. $\mathcal{P}$ is a compact n_p -dimensional box given by $\text{int}(\underline{p}, \bar{p})$, where $\underline{p}$ is the smallest trajectory parameter in $\mathcal{P}$ and $\bar{p}$ is the largest trajectory parameter in $\mathcal{P}$. We adapt the definition of trajectory parametrization from [70, Definition 7], and note two important details about the parametrization: First, all desired trajectories share a time instant $t_m \in [t_0 + t_{\text{plan}}, t_0 + t_{\text{plan}} + t_f)$ such that every desired trajectory consists of a *driving maneuver* during $[t_0 + t_{\text{plan}}, t_0 + t_{\text{plan}} + t_m)$ and a *contingency braking maneuver* during $[t_0 + t_{\text{plan}} + t_m, t_0 + t_{\text{plan}} + t_f]$. Second, the contingency braking maneuver slows down the ego vehicle's longitudinal speed to 0 by t_f . Note this latter property is used to ensure safety as we describe in Section 4.4.2. There are many choices of trajectory parametrizations, and the trajectory parametrization utilized in this work is provided in Appendix C.2.1.

4.3.3 Closed Loop Vehicle Dynamics

A partial feedback linearization controller that tracks a parameterized desired trajectory robustly and accounts for the modeling error described by Δ_u , Δ_v and Δ_r is provided in [70, Section V]. Using this controller, the closed loop dynamics of the high and low-speed

systems can be written as:

$$\dot{z}^{\text{hi}}(t) = f^{\text{hi}}(t, z^{\text{hi}}(t), p), \quad (4.9)$$

$$\dot{z}^{\text{lo}}(t) = f^{\text{lo}}(t, z^{\text{lo}}(t), p), \quad (4.10)$$

respectively. Moreover, because the vehicle dynamics change depending on u , we model the ego vehicle as a hybrid system HS [76, Section 1.2] as is done in [70, Section V-C]. The hybrid system HS contains a high-speed mode and a low-speed mode with state $z := z^{\text{hi}}$, whose dynamics can be written as

$$z(t) = \begin{cases} f^{\text{hi}}(t, z^{\text{hi}}(t), p), & \text{if } u(t) > u^{\text{cri}}, \\ \begin{bmatrix} f^{\text{lo}}(t, z^{\text{lo}}(t), p) \\ 0_{2 \times 1} \end{bmatrix}, & \text{if } u(t) \leq u^{\text{cri}}, \end{cases} \quad (4.11)$$

Instantaneous transition between the two modes within HS is described using the notion of a *guard* and *reset map*. The guard triggers a transition and is defined as $\{z(t) \in \mathbb{R}^6 \mid u(t) = u^{\text{cri}}\}$. Once a transition happens, the reset map resets the first $z(t)$ via (4.7) if $u(t)$ approaches u^{cri} from above and via (4.8) if $u(t)$ approaches u^{cri} from below.

4.3.4 Ego Vehicle, Environment, and Sensing

To provide guarantees about vehicle behavior in a receding horizon planning framework, we define the ego vehicle's footprint similarly to [70, Definition 10]:

Definition 24. *Given $\mathcal{W} \subset \mathbb{R}^2$ as the world space, the ego vehicle is a rigid body that lies in a rectangle $\mathcal{O}^{\text{ego}} := \text{int}([-0.5L, -0.5W]^T, [0.5L, 0.5W]^T) \subset \mathcal{W}$ with width $W > 0$ and length $L > 0$ at time $t = 0$. $\mathcal{O}^{\text{ego}}$ is called the footprint of the ego vehicle.*

For arbitrary time t , given state $z(t)$ of the ego vehicle that starts from initial condition $z_0 \in \mathcal{Z}_0 \subset \mathbb{R}^6$ and applies a control input parameterized by $p \in \mathcal{P}$, the ego vehicle's *forward occupancy* at time t can be represented as

$$\mathcal{E}(t, z_0, p) := \text{rotate}(h(t)) \cdot \mathcal{O}^{\text{ego}} + [x(t), y(t)]^\top, \quad (4.12)$$

which is a zonotope by (4.3). We define the obstacles as follows:

Definition 25. *An obstacle is a set $\mathcal{O}_i^{\text{obs}}(t) \subset \mathcal{W}$ that the ego vehicle should not collide with at time t , where $i \in \mathcal{I}$ is the index of the obstacle and $\mathcal{I}$ contains finitely many elements.*

The dependency on t in the definition of an obstacle allows the obstacle to move as t varies. However, if the i -th obstacle is static, then $\mathcal{O}_i^{\text{obs}}(t)$ is a constant. Note that in this work, we assume that we do not have perfect knowledge of obstacle locations and motion as is normally the case in real-life scenarios. We assume that this uncertainty in the locations of obstacles is represented by some arbitrary probability distribution. This is described in the next section. Assuming that the ego vehicle has a maximum speed ν^{ego} and all obstacles have a maximum speed ν^{obs} for all time, we make the following assumption on planning and sensing horizon.

Assumption 26. *The ego vehicle senses all obstacles within a sensor radius greater than $(t_f + t_{\text{plan}}) \cdot (\nu^{\text{ego}} + \nu^{\text{obs}}) + 0.5\sqrt{L^2 + W^2}$ around its center of mass.*

Assumption 26 ensures that any obstacle which may cause a collision between times $t \in [t_0 + t_{\text{plan}}, t_0 + t_{\text{plan}} + t_f]$ can be detected by the vehicle [112, Theorem 15]. Note one could treat sensor occlusions as obstacles that travel at the maximum obstacle speed [121, 122]. To simplify notation, we reset time to 0 whenever a feasible control policy is about to be applied, *i.e.*, $t_0 + t_{\text{plan}} = 0$. Finally to aid in the descriptions of obstacle uncertainty and system trajectory over-approximation in Sections 4.4 and 4.5, we partition the planning horizon $[0, t_f]$ into t_f/Δ_t time intervals with some positive number Δ_t that divides t_f , and denote $\mathcal{T}_j$ the j -th time interval $[(j-1)\Delta_t, j\Delta_t]$ for any $j \in \mathcal{J} := \{1, 2, \dots, t_f/\Delta_t\}$.

4.4 Online Planning Under Uncertainty

This section constructs a chance-constrained optimization problem to generate risk-aware motion plans. First, we describe the representations of the obstacle uncertainty used in this work. Then, we define risk-aware vehicle safety and conclude by illustrating how to formulate online planning as a chance-constrained optimization that limits the risk of collision.

4.4.1 Obstacle Uncertainty

To incorporate uncertainty in both obstacle sensing and obstacle motion prediction into the motion planning framework, let $w_{i,j}^{\text{obs}}$ be a random variable that takes values in $\mathcal{W}$. $w_{i,j}^{\text{obs}}$ describes the possible locations of the center of the i -th obstacle, $\mathcal{O}_i^{\text{obs}}(t)$, for any time $t \in \mathcal{T}_j$. We then make the following assumption about how $w_{i,j}^{\text{obs}}$ is distributed:

Assumption 27. *For any $i \in \mathcal{I}$ and $j \in \mathcal{J}$, $w_{i,j}^{\text{obs}}$'s Probability Density Function (PDF), $q_{i,j} : \mathcal{W} \rightarrow [0, +\infty)$, exists and is twice-differentiable. In addition, there exists some $w_{i,j}^{\text{obs}}$ sampled according to $q_{i,j}$ such that $\mathcal{O}_i^{\text{obs}}(t) \subseteq \langle w_{i,j}^{\text{obs}}, G^{\text{obs}} \rangle$ at some $t \in \mathcal{T}_j$, where G^{obs} is a 2-row constant matrix.*

According to Assumption 27, $\langle w_{i,j}^{\text{obs}}, G^{\text{obs}} \rangle = w_{i,j}^{\text{obs}} + \langle 0, G^{\text{obs}} \rangle$ has an uncertain center $w_{i,j}^{\text{obs}}$ with probability density $q_{i,j}(w_{i,j}^{\text{obs}})$, and has invariant shape and size with respect to i and j due to the constant generator matrix G^{obs} that accounts for the footprint of any obstacle. Such probability density functions $q_{i,j}$ can be generated by, for example, performing variants of Kalman Filter [127, 6] on the i -th obstacle given its dynamics or detecting the i -th obstacle with a Bayesian confidence framework [35] during $\mathcal{T}_j$. The generator matrix G^{obs} can be generated as the union of footprints of all obstacles.

4.4.2 Risk-Aware Vehicle Safety

In dynamic environments, it may be difficult to avoid collisions in all scenarios (*e.g.*, a parked ego vehicle can be run into). As a result, we instead develop a trajectory synthesis technique that ensures that the ego vehicle is not-at-fault [112, Definition 11]. We define not-at-fault safety in this work from a probabilistic perspective by bounding the probability of the ego vehicle running into any obstacles.

Definition 28. *Let the ego vehicle start from initial condition $z_0 \in \mathcal{Z}_0$ with control parameter $p \in \mathcal{P}$. Given a user-specified allowable risk threshold $\epsilon \in [0, 1]$, the ego vehicle is not-at-fault with a risk of collision of at most ϵ , if it is stopped, or if*

$$\sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} \int_{\cup_{t \in \mathcal{T}_j} (\mathcal{E}(t, z_0, p) \oplus \langle 0, G^{\text{obs}} \rangle)} q_{i,j}(w) dw \leq \epsilon, \quad (4.13)$$

while it is moving during $[0, t_f]$.

Note that the domain of integration in (4.13) is the ego vehicle's forward occupancy buffered by the obstacle's footprint. Thus to satisfy (4.13), the probability that the buffered ego vehicle's forward occupancy intersects with an obstacle's center must be bounded by epsilon over all time intervals and all possible obstacles. In particular, this ensures that the probability of the ego vehicle (including its footprint) intersecting with any obstacle (including its footprint) over the planning horizon is bounded by ϵ via [41, Lem. 5.1].

4.4.3 Online Optimization

To construct a motion plan that ensures the ego vehicle is not-at-fault with a risk of collision at most ϵ during online planning, one could solve the following chance-constrained

optimization:

$$\begin{aligned}
& \min_{p \in \mathcal{P}} \text{cost}(z_0, p) & (\text{Opt-R}) \\
& \text{s.t. } \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} \int_{\cup_{t \in \mathcal{T}_j} (\mathcal{E}(t, z_0, p) \oplus \langle 0, G^{\text{obs}} \rangle)} q_{i,j}(w) dw \leq \epsilon
\end{aligned}$$

where $\text{cost} : \mathcal{Z}_0 \times \mathcal{P} \rightarrow \mathbb{R}$ is a user-specified cost function, and the constraint is a chance constraint that ensures the vehicle is not-at-fault with a maximum risk of collision ϵ during the planning horizon as stated in Definition 28.

To achieve real-time motion planning, (Opt-R) must be solved within t_{plan} seconds. However, efficiently evaluating the chance constraint in (Opt-R) in a closed form can be challenging in real applications for two reasons. First, the exact information of the ego vehicle's location $\mathcal{E}(t, z_0, p)$ at any time is usually inaccessible due to the nonlinear and hybrid nature of the vehicle dynamics. Second, the probability distribution that describes the uncertain observation of an obstacle's location as described in Assumption 27 can be arbitrary. As illustrated in Appendix C.3.2, even approximating this chance constraint accurately using Monte-Carlo is challenging for real-time motion planning. Therefore to achieve real-time performance, RADIUS seeks a closed-form approximation of the risk of collision for evaluation efficiency. This chapter focuses on constructing an over-approximation to ensure that RADIUS does not under-estimate the true risk of collision or generate plans that violate the not-at-fault condition from Definition 28. Moreover, it is preferable that this approximation is differentiable, as providing the gradient of the constraint can speed up the solving procedure of online optimization. We describe how we generate an approximation that satisfies these requirements in the next two sections.

4.5 Offline Reachability Analysis

Due to the nonlinear and hybrid nature of the vehicle dynamics, it is challenging to compute the trajectory of vehicle state exactly when evaluating the chance constraint in (Opt-R). To resolve this challenge, RADIUS over-approximates the ego vehicle's trajectory using zonotopes as stated below:

Assumption 29. *Let z be a solution to (4.11) starting from initial condition $z_0 \in \mathcal{Z}_0$ with control parameter $p \in \mathcal{P}$. For each $j \in \mathcal{J}$, there exists a map $\xi_j : \mathcal{Z}_0 \times \mathcal{P} \rightarrow P(\mathcal{W})$ such that*

1. $\xi_j(z_0, p)$ contains the ego vehicle's footprint during $\mathcal{T}_j$, i.e., $\cup_{t \in \mathcal{T}_j} \mathcal{E}(t, z_0, p) \subseteq \xi_j(z_0, p)$,
and

2. $\xi_j(z_0, p)$ is a zonotope of the form $\langle c_j(z_0) + A_j \cdot p, G_j \rangle$ with some linear function $c_j : \mathcal{Z}_0 \rightarrow \mathbb{R}^2$, some matrix $A_j \in \mathbb{R}^{2 \times n_p}$ and some 2-row matrix G_j .

The collection of maps $\{\xi_j\}_{j \in \mathcal{J}}$ can be constructed by applying existing techniques for offline reachability analysis [70, Section VI]. In particular, $\{\xi_j\}_{j \in \mathcal{J}}$ can be generated by first applying the open-source toolbox CORA [7] to over-approximate the trajectory of the ego vehicle's state with initial condition z_0 and control parameter p using a collection of zonotopes, where each zonotope over-approximates a segment of the trajectory during one small time interval among $\{\mathcal{T}_j\}_{j \in \mathcal{J}}$, and then accounts for the ego vehicle's footprint. The proof of Lemma 26 in [70] provides explicit formulas for ξ_j , c_j , A_j and G_j . Note formulas provided in [70, Lemma 26] assume the computation in the body frame of the ego vehicle, i.e., assuming $x(0) = y(0) = h(0) = 0$. In the case when the initial position and heading of the ego vehicle are not zeros, one can represent $\xi_j(z_0, p)$ in the world frame via frame transformation based on z_0 . In the remainder of this chapter, we assume $\xi_j(z_0, p)$ is represented in the world frame. We refer to $\xi_j(z_0, p)$ as the zonotope reachable set.

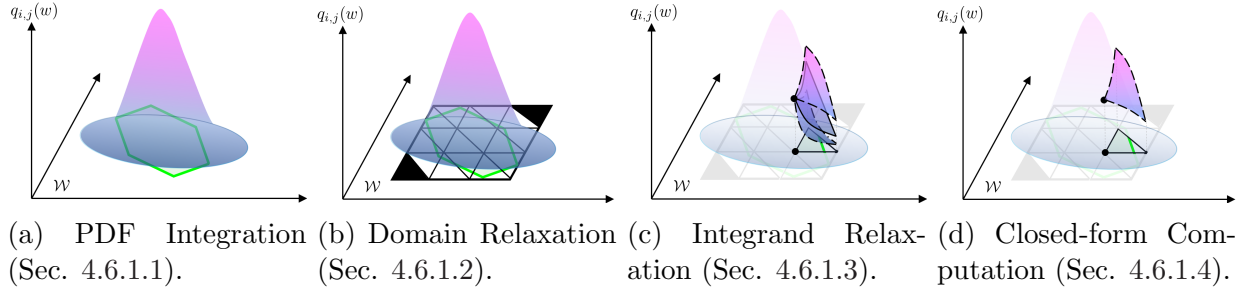


Figure 4.2: An illustration of chance constraint relaxation. Given arbitrary $(i, j) \in \mathcal{I} \times \mathcal{J}$, in (a) the risk of collision between the ego vehicle and the i -th obstacle during time interval $\mathcal{T}_j$ is relaxed as the integration of probability density function $q_{i,j}$ (shown in purple and blue) over zonotope $\xi_j(z_0, p) \oplus \langle 0, G^{\text{obs}} \rangle \subset \mathcal{W}$ (shown in green). In (b), $\xi_j(z_0, p) \oplus \langle 0, G^{\text{obs}} \rangle$ is over-approximated by a collection of right-angled triangles colored in white and black depending on if a triangle has a nontrivial intersection with $\xi_j(z_0, p) \oplus \langle 0, G^{\text{obs}} \rangle$ or not. In (c), Interval Arithmetic is used to generate an over- and under-approximation of $q_{i,j}$ over each right-angled triangle. And in (d), the integration of the over-approximation of $q_{i,j}$ over each right-angled triangle is computed in closed form.

4.6 An Implementable Alternative to (Opt-R)

This section describes an implementable alternative to (Opt-R) that can be solved rapidly. In particular, we discuss how to relax the chance constraint in (Opt-R) in a conservative fashion.

4.6.1 Chance Constraint Relaxation

The chance constraint in (Opt-R) is relaxed conservatively as illustrated in Figure 4.2. First, the risk of collision during $[0, t_f]$ is over-approximated using Assumption 29. Then we relax the domain of integration into a collection of right-angled triangles and relax the PDF as a quadratic polynomial. Finally, we describe a closed-form equation for the relaxed integral.

4.6.1.1 PDF Integration

Recall the ego vehicle's performance during the planning horizon is over-approximated by a collection of maps $\{\xi_j\}_{j \in \mathcal{J}}$, then the chance constraint in (Opt-R) is relaxed according to the following lemma which follows directly from the first property in Assumption 29:

Lemma 30. *Suppose the ego vehicle starts with initial condition $z_0 \in \mathcal{Z}_0$ and control parameter $p \in \mathcal{P}$. Let probability density functions $\{q_{i,j}\}_{(i,j) \in \mathcal{I} \times \mathcal{J}}$ and matrix G^{obs} be as assumed in Assumption 27. Let maps $\{\xi_j\}_{j \in \mathcal{J}}$ be as assumed in Assumption 29. Then for arbitrary $\epsilon \in [0, 1]$, (4.13) holds if*

$$\sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} \int_{\xi_j(z_0, p) \oplus \langle 0, G^{obs} \rangle} q_{i,j}(w) dw \leq \epsilon. \quad (4.14)$$

4.6.1.2 Domain Relaxation

Recall by Assumption 29 that $\xi_j(z_0, p)$ can be rewritten as $\langle c_j(z_0) + A_j \cdot p, G_j \rangle = \langle c_j(z_0), G_j \rangle + A_j \cdot p$. Then to relax the domain of integration $\xi_j(z_0, p) \oplus \langle 0, G^{obs} \rangle$, we start by constructing a k -by- k grid that covers zonotope $\xi_j(z_0, 0) \oplus \langle 0, G^{obs} \rangle = \langle c_j(z_0), [G_j, G^{obs}] \rangle$ where k is some user-specified positive integer. Each cell in the grid shares the same size and is indexed by its row and column index in the grid. Each cell in the grid is further divided into two simplexes as right-angled triangles that are indexed by 1 or -1 corresponding to the lower or upper triangles of the cell, respectively. For notational ease, denote $\mathcal{S}_{j,k_1,k_2,k_3}(z_0) \subset \mathcal{W}$ as the simplex indexed by $k_3 \in \{-1, 1\}$ in the cell on the k_1 -th row and k_2 -th column of the grid that covers $\langle c_j(z_0), [G_j, G^{obs}] \rangle$. Define

$$\begin{aligned} \bar{\mathcal{S}}_j(z_0) &:= \{\mathcal{S}_{j,k_1,k_2,k_3}(z_0) \mid k_1, k_2 \in \{1, 2, \dots, k\}, \\ &\quad |k_3| = 1, \mathcal{S}_{j,k_1,k_2,k_3}(z_0) \cap \langle c_j(z_0), [G_j, G^{obs}] \rangle \neq \emptyset\}. \end{aligned} \quad (4.15)$$

as the collection of every possible $\mathcal{S}_{j,k_1,k_2,k_3}(z_0)$ that intersects with $\langle c_j(z_0), [G_j, G^{obs}] \rangle$, thus

$$\langle c_j(z_0), [G_j, G^{obs}] \rangle \subset (\cup_{\mathcal{S} \in \bar{\mathcal{S}}_j(z_0)} \mathcal{S}). \quad (4.16)$$

Notice $\xi_j(z_0, p) = \xi_j(z_0, 0) + A_j \cdot p$, therefore $\xi_j(z_0, p) \oplus \langle 0, G^{\text{obs}} \rangle \subset \left((\cup_{\mathcal{S} \in \bar{\mathcal{S}}_j(z_0)} \mathcal{S}) + A_j \cdot p \right)$ and

$$\int_{\xi_j(z_0, p) \oplus \langle 0, G^{\text{obs}} \rangle} q_{i,j}(w) dw \leq \sum_{\mathcal{S} \in \bar{\mathcal{S}}_j(z_0)} \int_{\mathcal{S} + A_j p} q_{i,j}(w) dw. \quad (4.17)$$

Remark 31. Note that for any $\mathcal{S} \in \bar{\mathcal{S}}_j(z_0)$, $\mathcal{S}$ depends on z_0 and j . However, to reduce notational complexity, we drop its dependency on z_0 and j in the remainder of this chapter.

4.6.1.3 Integrand Relaxation

Next, we present a lemma, whose proof is provided in Appendix C.1, to conservatively approximate the integral in the chance constraint by relaxing the integrand.

Lemma 32. Suppose the ego vehicle starts with initial condition $z_0 \in \mathcal{Z}_0$ and control parameter $p \in \mathcal{P}$. Let $\mathcal{S} \in \bar{\mathcal{S}}_j(z_0)$ and let $w_{\mathcal{S}}^{\text{ctr}}$ denote the vertex of the right angle in $\mathcal{S}$. Define the function $q_{i,j,\mathcal{S}} : \mathcal{W} \times \mathcal{P} \rightarrow [0, \infty)$ as

$$\begin{aligned} q_{i,j,\mathcal{S}}(w, p) &:= q_{i,j}(w_{\mathcal{S}}^{\text{ctr}} + A_j p) + \frac{\partial q_{i,j}}{\partial w}(w_{\mathcal{S}}^{\text{ctr}} + A_j p) \cdot \\ &\quad \cdot (w - w_{\mathcal{S}}^{\text{ctr}} - A_j p) + \frac{1}{2}(w - w_{\mathcal{S}}^{\text{ctr}} - A_j p)^\top \cdot \\ &\quad \cdot H_{\mathcal{S}} \cdot (w - w_{\mathcal{S}}^{\text{ctr}} - A_j p), \end{aligned} \quad (4.18)$$

where $H_{\mathcal{S}} \in \mathbb{R}^{2 \times 2}$ is generated by taking element-wise supremum of the Hessian of $q_{i,j}$ over $\mathcal{S} \oplus A_j \mathcal{P}$ using Interval Arithmetic [44]. Then for all $w \in \mathcal{S} + A_j p$:

$$q_{i,j}(w) \leq q_{i,j,\mathcal{S}}(w, p). \quad (4.19)$$

Note the inequality in (4.19) flips if $H_{\mathcal{S}}$ is generated by taking an element-wise infimum of the Hessian of $q_{i,j}$. This would create an under-approximation to $q_{i,j}$.

4.6.1.4 Closed-form Computation

As a result of (4.17) and (4.19), the following inequality holds:

$$\int_{\xi_j(z_0, p) \oplus \langle 0, G^{\text{obs}} \rangle} q_{i,j}(w) dw \leq \sum_{\mathcal{S} \in \bar{\mathcal{S}}_j(z_0)} \int_{\mathcal{S} + A_j p} q_{i,j,\mathcal{S}}(w, p) dw. \quad (4.20)$$

Notice that $q_{i,j,\mathcal{S}}(w, p)$ defined in (4.19) is indeed a quadratic polynomial of w , and $\mathcal{S} + A_j p$ is a simplex in $\mathbb{R}^2$. One can then compute $\int_{\mathcal{S} + A_j p} q_{i,j,\mathcal{S}}(w, p) dw$ in closed-form as follows:

Theorem 33. For any $i \in \mathcal{I}$, $j \in \mathcal{J}$, $z_0 \in \mathcal{Z}_0$, $p \in \mathcal{P}$ and $\mathcal{S} \in \overline{\mathcal{S}}_j(z_0)$, let $A_j \in \mathbb{R}^{2 \times n_p}$ be defined as in Assumption 29, let $q_{i,j,\mathcal{S}}$ be defined as in (4.19), and let $\mathbf{i} \mathbf{d} \mathbf{x}_{k_3}(\mathcal{S})$ denote the last index of its argument (i.e., $\mathbf{i} \mathbf{d} \mathbf{x}_{k_3}(\mathcal{S}_{j,k_1,k_2,k_3}(z_0)) = k_3$). Assume that positive numbers l_1 and l_2 give the lengths of horizontal and vertical right angle sides of $\mathcal{S}$ respectively, then

$$\begin{aligned} \int_{\mathcal{S}+A_j p} q_{i,j,\mathcal{S}}(w,p) dw &= \frac{1}{2|(|A_{\mathcal{S}})|} \left(q_{i,j}(w_{\mathcal{S}}^{\text{ctr}} + A_j p) + \right. \\ &\quad + \begin{bmatrix} \frac{1}{4\sqrt{3}} & \frac{1}{4\sqrt{3}} \end{bmatrix} \left(\hat{H}_{\mathcal{S}} \odot \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \right) \begin{bmatrix} \frac{1}{2\sqrt{3}} \\ \frac{1}{2\sqrt{3}} \end{bmatrix} + \\ &\quad \left. + \frac{\partial q_{i,j}}{\partial w}(w_{\mathcal{S}}^{\text{ctr}} + A_j p) \cdot A_{\mathcal{S}}^{-1} \cdot \begin{bmatrix} \frac{1}{3} \\ \frac{1}{3} \end{bmatrix} \right) \end{aligned} \quad (4.21)$$

where $\odot$ denotes the element-wise multiplication and

$$A_{\mathcal{S}} = \begin{bmatrix} \mathbf{i} \mathbf{d} \mathbf{x}_{k_3}(\mathcal{S})/l_1 & 0 \\ 0 & \mathbf{i} \mathbf{d} \mathbf{x}_{k_3}(\mathcal{S})/l_2 \end{bmatrix}, \quad (4.22)$$

$$\hat{H}_{\mathcal{S}} = A_{\mathcal{S}}^{-\top} H_{\mathcal{S}} A_{\mathcal{S}}^{-1}. \quad (4.23)$$

Proof. The claim follows from [65, Theorem 1.1] and the fact that $A_{\mathcal{S}} \cdot ((\mathcal{S} + A_j p) - (w_{\mathcal{S}}^{\text{ctr}} + A_j p))$ equals the canonical simplex $\Delta := \{(a,b) \in \mathbb{R}^2 \mid a+b \leq 1, a \geq 0, b \geq 0\}$. $\square$

4.6.2 Tractable Online Optimization

The computation in Section 4.6.1 provides a tractable way to enforce not-at-fault behavior compared to the original chance constraint in (Opt-R). As a result, RADIUS solves the following optimization during online planning:

$$\begin{aligned} \min_{p \in \mathcal{P}} \quad & \text{cost}(z_0, p) && (\text{Opt-E}) \\ \text{s.t.} \quad & \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} \sum_{\mathcal{S} \in \overline{\mathcal{S}}_j(z_0)} \int_{\mathcal{S}+A_j p} q_{i,j,\mathcal{S}}(w,p) dw \leq \epsilon. \end{aligned}$$

(Opt-E) is a strengthened version of (Opt-R) because satisfaction of the chance constraint in (Opt-E) implies satisfaction of the chance constraint in (Opt-R) by Lemma 30, (4.16), and (4.19). Thus, the following lemma holds based on Definition 28.

Lemma 34. If the ego vehicle applies any feasible solution, $p^* \in \mathcal{P}$, of (Opt-E) beginning from $z_0 \in \mathcal{Z}_0$ at $t = 0$, then it is not-at-fault with a risk of collision at most ϵ during $[0, t_f]$.

Algorithm 4 Chance Constraint Parallelization

Require: $z_0 \in \mathcal{Z}_0$, $p \in \mathcal{P}$, $\{q_{i,j}\}_{(i,j) \in \mathcal{I} \times \mathcal{J}}$, $\{\xi_j\}_{j \in \mathcal{J}}$
1: **Generate** $\{\bar{\mathcal{S}}_j(z_0)\}_{j \in \mathcal{J}}$ as in (4.15) using $\{\xi_j\}_{j \in \mathcal{J}}$
2: **Parfor** $(i, j) \in \mathcal{I} \times \mathcal{J}$ **do**
3: **Parfor** $\mathcal{S} \in \bar{\mathcal{S}}_j(z_0)$ **do**
4: **Compute** $\int_{\mathcal{S}+A_j p} q_{i,j,\mathcal{S}}(w, p) dw$ as in (4.21)
5: **Compute** $\frac{\partial}{\partial p} \int_{\mathcal{S}+A_j p} q_{i,j,\mathcal{S}}(w, p) dw$ as in (4.24)
6: **End Parfor**
7: **End Parfor**

4.6.3 Constraint Gradient and Parallelization

To improve the solving procedure of (Opt-E), we provide the derivative of its chance constraint. To compute the gradient of the chance constraint in (Opt-E), it suffices to compute the derivative of $\int_{\mathcal{S}+A_j p} q_{i,j,\mathcal{S}}(w, p) dw$ in (4.21) with respect to p . Notice that $A_{\mathcal{S}}$ and $\hat{H}_{\mathcal{S}}$ are invariant over $\mathcal{P}$, then

$$\begin{aligned} \frac{\partial}{\partial p} \int_{\mathcal{S}+A_j p} q_{i,j,\mathcal{S}}(w, p) dw &= \\ &= \frac{1}{2|(|A_{\mathcal{S}})|)} \cdot \left(\frac{\partial q_{i,j}}{\partial w}(w_{\mathcal{S}}^{\text{ctr}} + A_j p) \cdot A_j + \right. \\ &\quad \left. + \begin{bmatrix} \frac{1}{3} & \frac{1}{3} \end{bmatrix} A_{\mathcal{S}}^{-\top} \text{Hess}_{q_{i,j}}(w_{\mathcal{S}}^{\text{ctr}} + A_j p) \cdot A_j \right), \end{aligned} \quad (4.24)$$

where $\text{Hess}_{q_{i,j}}(w_{\mathcal{S}}^{\text{ctr}} + A_j p)$ is the evaluation of the Hessian of $q_{i,j}$ at $w_{\mathcal{S}}^{\text{ctr}} + A_j p$.

Notice the computations of $\int_{\mathcal{S}+A_j p} q_{i,j,\mathcal{S}}(w, p) dw$ and its gradient can be parallelized for all $(i, j) \in \mathcal{I} \times \mathcal{J}$ and all $\mathcal{S} \in \bar{\mathcal{S}}_j(z_0)$ as indicated in Algorithm 4. $\bar{\mathcal{S}}_j(z_0)$ is generated in Line 1 for all $j \in \mathcal{J}$. The outer parallel for loop that starts at Line 2 iterates every element in $\mathcal{I} \times \mathcal{J}$, and the inner parallel for loop that starts at Line 3 iterates over each simplex in $\bar{\mathcal{S}}_j(z_0)$. The integral and its gradient over each simplex are computed from Lines 4 to 5. Then the chance constraint and its gradient in (Opt-E) can be computed as the summation of all computed $\int_{\mathcal{S}+A_j p} q_{i,j,\mathcal{S}}(w, p) dw$ and $\frac{\partial}{\partial p} \int_{\mathcal{S}+A_j p} q_{i,j,\mathcal{S}}(w, p) dw$ respectively.

4.6.4 Online Operation

Algorithm 5 summarizes the online operation of RADIUS. It begins by sensing and constructing predictions for obstacle locations as in Assumption 27 in Line 1. **OnlineOpt** then solves (Opt-E) to search for a not-at-fault plan with a risk of collision at most ϵ in Line 2. If (Opt-E) is infeasible, then RADIUS terminates planning in Line 3, otherwise it enters the

Algorithm 5 RADIUS Online Planning

Require: $z_0 \in \mathcal{Z}_0$ and $\epsilon \in [0, 1]$

- 1: **Initialize:** $\{q_{i,j}\}_{(i,j) \in \mathcal{I} \times \mathcal{J}} \leftarrow \text{SenseObstacles}()$
- 2: **Try** $p^* \leftarrow \text{OnlineOpt}(z_0, \{q_{i,j}\}_{(i,j) \in \mathcal{I} \times \mathcal{J}}, \epsilon)$
- 3: **Catch** terminate planning
- 4: **Loop:** // Line 6 executes simultaneously with Lines 7-11
- 5: **Reset** t to 0
- 6: **Execute** p^* during $[0, t_m)$
- 7: $\{q_{i,j}\}_{(i,j) \in \mathcal{I} \times \mathcal{J}} \leftarrow \text{SenseObstacles}()$
- 8: $z_0 \leftarrow \text{StatePrediction}(z_0, p^*, t_m)$
- 9: **If** $z_0 \notin \mathcal{Z}_0$, **then** break
- 10: **Try** $p^* \leftarrow \text{OnlineOpt}(z_0, \{q_{i,j}\}_{(i,j) \in \mathcal{I} \times \mathcal{J}}, \epsilon)$
- 11: **Catch** break
- 12: **End**
- 13: **Execute** p^* during $[t_m, t_f]$, **then** terminate planning

planning loop in Line 4. Within the loop, RADIUS first resets time to 0 in Line 5 and executes the entire driving maneuver corresponding to p^* in Line 6. Meanwhile, **SenseObstacles** updates the predictions for obstacle locations in Line 7 and **StatePrediction** predicts the ego vehicle state at $t = t_m$ as in Assumption 23 in Line 8. In the case when the predicted vehicle state $z_0 \notin \mathcal{Z}_0$, then RADIUS breaks the planning loop in Line 9. Otherwise (**Opt-E**) is solved again using the updated obstacle information and z_0 in Line 10, and RADIUS breaks the planning loop if (**Opt-E**) is infeasible or takes longer than t_{plan} to find a solution in Line 11. Finally, once RADIUS leaves the planning loop, the contingency braking maneuver corresponding to p^* is executed. This braking maneuver by construction brings the vehicle to a stop. Note that the risk of collision of this maneuver was already verified to be less than ϵ during the previous planning iteration. Subsequently, RADIUS terminates planning in Line 13. Note, we are able to obtain the following theorem by iteratively applying Lemma 34:

Theorem 35. *Suppose the ego vehicle can sense and predict the surrounding obstacles as in Assumption 27, and starts from rest with an initial condition $z_0 \in \mathcal{Z}_0$ at $t = 0$. Then by performing planning and execution as in Algorithm 5, the ego vehicle is not-at-fault with a risk of collision at most ϵ for all time.*

4.7 Experiments and Results

All experiments are conducted in MATLAB R2023a on a Ubuntu 22.04 machine with an AMD Ryzen 9 5950X CPU, two NVIDIA RTX A6000 48GB GPUs, and 64GB RAM. Parallelization is achieved using CUDA 12.1. RADIUS invokes C++ for online planning using

IPOPT. Additional details on experimental setup can be found in Appendix C.2. An additional experiment detailing the single planning iteration performance of RADIUS, CCPBA, Cantelli MPC and REFINE can be found in Appendix C.3.1. An ablation study illustrating the importance of the analytical gradient and closed-form over-approximation of the risk of collision for real-time performance can be found in Appendix C.3.2. Readers can find our implementation³ and the video⁴ of simulations and hardware demos utilising RADIUS online.

4.7.1 Tightness and Generality of Risk Approximation

For effective motion planning, we desire a tight over-approximation of the risk of collision. Additionally, we want RADIUS to be able to generalize to arbitrary probability distributions as well to accommodate other uncertainty representations. Thus, to evaluate the tightness of our over-approximation of $\int_{\xi_j(z_0, p) \oplus <0, G^{\text{obs}}>} q_{i,j}(w) dw$, we compare the proposed method’s approximation of the risk of collision against Monte-Carlo integration [14, Chapter 4], the Cantelli inequality [115, Section IV B], and the Chance-Constrained Parallel Bernstein Algorithm (CCPBA) [111, Chapter 6] on 9000 randomly generated test cases.

In each test case, (i, j, z_0, p) is randomly chosen from $\mathcal{I} \times \mathcal{J} \times \mathcal{Z}_0 \times \mathcal{P}$. Additionally, $q_{i,j}$ is set to be the probability density function of either a randomly generated 2-dimensional (2D) Gaussian distribution, 2D Beta distribution [86] or a 2D Multimodal distribution. Each type of distribution is evaluated in 3000 test cases. Note that because CCPBA is unable to handle a distribution that is not Gaussian, we only evaluate RADIUS and the Cantelli inequality on these non-Gaussian probability distributions. We perform the Monte-Carlo integration of $q_{i,j}$ over $\xi_j(z_0, p) \oplus <0, G^{\text{obs}}>$ with 10^6 samples and treat that as the ground truth.

The risk approximation error, which is the difference between the Monte-Carlo integration and each method, is illustrated in Table 4.1. The results from Table 4.1 show that RADIUS provides significantly tighter upper bounds to the ground truth than CCPBA and the Cantelli inequality for the tested probability distributions. The associated average and maximum times to compute the over-approximation for each method are shown in Table C.1 of Appendix C.2.2.

4.7.2 Simulations

This section compares RADIUS to two state-of-the-art chance-constrained motion planning algorithms: CCPBA [111] and Cantelli MPC [115], and one state-of-the-art determinis-

³<https://github.com/roahmlab/RADIUS>

⁴<https://youtu.be/8eU9fiA39sE>

Method	Gaussian Error (Mean, Max.)	Beta Error (Mean, Max.)	Multimodal Error (Mean, Max.)
RADIUS	(0.0073, 0.0523)	(0.0065, 0.0489)	(0.0079, 0.0262)
Cantelli	(0.1774, 0.3420)	(0.2221, 0.5062)	(0.4734, 0.5851)
CCPBA	(0.1881, 0.2149)	-	-

Table 4.1: Results for the risk of collision estimation error of RADIUS and the Cantelli Inequality when the obstacle location is represented by three types of probability distributions. Note that CCPBA can only handle Gaussian distributions so does not have results for the other types of distributions.

tic motion planning algorithm REFINE [70] in dense highway scenarios. Additionally, we evaluate how varying the allowable risk threshold (ϵ) for RADIUS affects the ego vehicle behavior in various unprotected left turn scenarios. For all simulation experiments, t_{plan} is set as 3[sec] and t_f is chosen according to [70, Lemma 14]. A description of parameterized desired trajectories that are used in this work is provided in Appendix C.2.1.

4.7.2.1 3-Lane Highway

This experiment evaluates the performance of RADIUS in a 3-lane highway environment where it must execute multiple planning iterations in succession and compares its performance to CCPBA, Cantelli MPC and REFINE. We compare the methods over 1000 randomly generated 3-lane highway scenarios in simulation with $\epsilon = 0.05$. In each simulation scenario, the ego vehicle is expected to navigate through dynamic traffic for 1000[m] from a given initial position. Each scenario contains 3 static obstacles and a number of moving vehicles as dynamic obstacles, where the number of moving vehicles in each scenario is randomly selected between 5 and 25. To ensure that we can compare to CCPBA, we choose the $q_{i,j}$ describing each obstacle’s location during $\mathcal{T}_j$ as a Gaussian with a standard deviation of $\sigma_{i,j}$ for any $(i, j) \in \mathcal{I} \times \mathcal{J}$. To compare to REFINE, which is a deterministic motion planning algorithm that assumes perfect knowledge of the locations of obstacles, we require that REFINE avoids a box that over-approximates the $5\text{-}\sigma_{i,j}$ region of a Gaussian distribution. Details on the reasoning behind the choice of $5\text{-}\sigma_{i,j}$ are explained in the last paragraph of Appendix C.2.3.

Each scenario is simulated for 10 trials resulting in a total of 10,000 simulation cases. In each trial, the starting locations of the ego vehicle and obstacles are the same, but the trajectories the obstacles follow are varied to capture the uncertain nature of each scenario. Additional details on how these scenarios and obstacle trajectories are generated can be found in Appendix C.2.3.

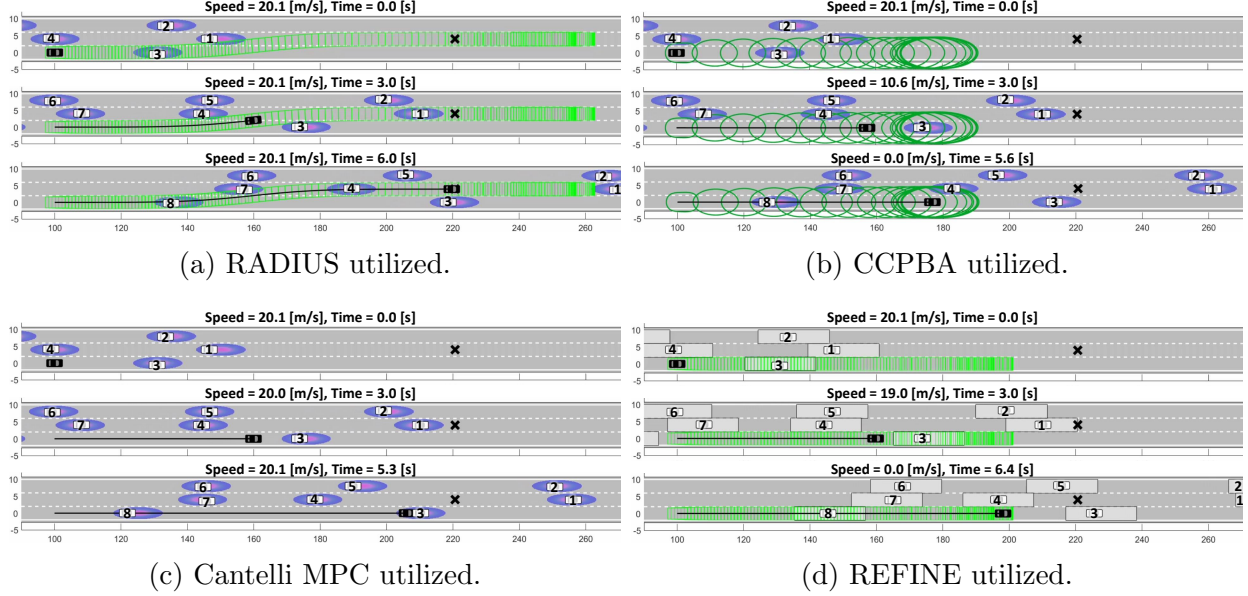


Figure 4.3: Example of a single planning iteration of a simulated scenario in which RADIUS is able to navigate the ego vehicle (black) to the provided waypoint (black cross) through a lane change maneuver solved by one planning iteration, while CCPBA and REFINE execute contingency braking maneuvers as they are unable to find feasible solutions. Cantelli MPC results in a crash, as it has no fail-safe maneuver, so it instead maintains its velocity and continues searching for a feasible solution to execute the lane change. Forward reachable sets are shown in green. Obstacles are shown in white and are marked by their indices to make them trackable among different time instances. Probability distributions are illustrated as purple and blue ellipses where the probability density from low to high is illustrated from blue to purple. Note that for visualization purposes we have buffered the probability distributions shown in this figure by the footprints of the obstacles. Lastly, the convex hull that REFINE uses to over-approximate the $5\text{-}\sigma_{i,j}$ regions of the obstacle probability distributions are shown as gray boxes around the obstacles for all $(i,j) \in \mathcal{I} \times \mathcal{J}$. Note that the distances on the x-axis and y-axis are in [m].

Figure 4.3 depicts an example of a single planning iteration of a scenario where RADIUS is able to successfully find a solution to execute a lane change to get to the given waypoint, while CCPBA, Cantelli MPC and REFINE are unable to execute the lane change. Table 4.2 presents RADIUS, CCPBA, Cantelli MPC, and REFINE’s results for all 10,000 cases of the 3-lane highway multiple planning horizon experiment. As seen in Table 4.2, RADIUS is able to achieve a higher success rate than CCPBA and Cantelli MPC. This is in part due to the fact that RADIUS is able to more closely approximate the ground truth risk of collision as shown in Table 4.1. REFINE also has a lower success rate than RADIUS due to its conservative treatment of the uncertain region for the obstacle location. Cantelli MPC has more crashes than CCPBA, and RADIUS because (1) Cantelli MPC, unlike CCPBA

Method	Success [%]	Crash [%]	Safely Stop [%]	Solve Time [s] (Mean, Max.)
RADIUS	80.3	0.8	18.9	(0.412, 0.683)
CCPBA	44.5	0.4	55.1	(0.291, 0.417)
Cantelli MPC	25.6	74.4	0.0	(0.743, 3.921)
REFINE	61.0	0.0	39.0	(0.341, 0.562)

Table 4.2: Simulation results comparing RADIUS to CCPBA, Cantelli MPC and REFINE on a 1000[m] stretch of dense highway. A successful trial means the ego vehicle was able to successfully travel 1000[m] without crashing or coming to a stop.

and RADIUS can only enforce risk-aware safety at discrete instances so crashes could occur in between these instances, (2) Cantelli MPC does not have a fail-safe stopping maneuver in case it cannot find a solution. Instead, it maintains speed and searches for a new solution and sometimes crashes into the vehicles in front of it.

Notice that despite the fact that $\epsilon = 0.05$, RADIUS has a crash rate that is smaller than 5%. This occurs because even though RADIUS has a significantly tighter over-approximation than the comparison methods, it still over-approximates the true risk of collision. This difference in the over-approximation and the true risk of collision can be attributed to: (1) the multiple relaxations made to compute the closed-form over-approximation highlighted in Section 4.6.1, and (2) the over-approximation of the actual ego vehicle location within the j -th time interval with the collection of maps introduced in Assumption 29. Future work will explore the contribution of each of these relaxations to the extent of over-approximation. **Note that despite this over-approximation, RADIUS is still significantly less conservative than deterministic algorithms like REFINE as seen in Table 4.2.**

4.7.2.2 Left Turning

The aim of this experiment is to evaluate how different allowable risk thresholds (ϵ) affect the behavior and performance of RADIUS over a single planning iteration, and unprotected left turns are ideal scenarios for this task. In these scenarios, being less conservative has a large effect on the ability of the ego vehicle to complete the left turn faster by navigating through tight windows in the traffic of oncoming vehicles. In this experiment, the risk threshold is set to 0.01, 0.05, 0.10, 0.20 and 0.50, and the ego vehicle is tasked with navigating through 100 randomly generated unprotected left turning scenarios. Each scenario is simulated for 10 trials, where in each trial the obstacle trajectories are varied in the same way as mentioned in Section 4.7.2.1. In each scenario, the ego vehicle is tasked with executing an unprotected left turn across two oncoming lanes at a 4-way intersection as depicted in Figure 4.4. Each

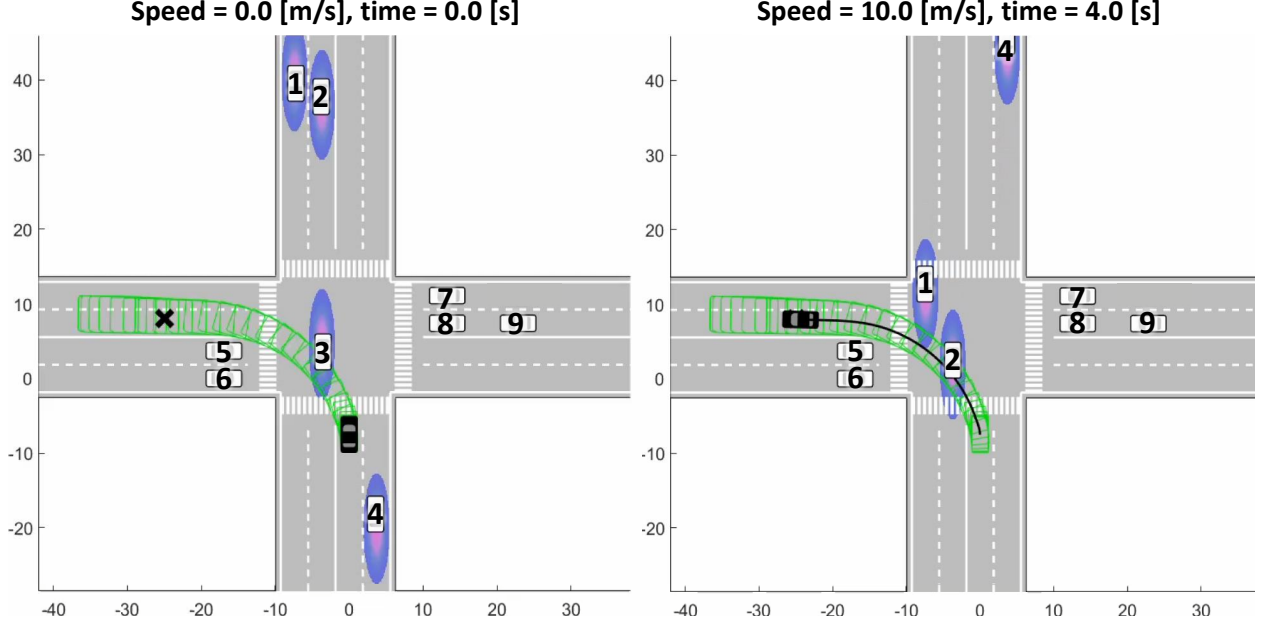


Figure 4.4: Example of a simulated unguarded left turning trial where the ego vehicle (black) navigates through a gap in the oncoming traffic to successfully execute the unguarded left turn. Forward reachable sets are shown in green. Obstacles are shown in white and are marked by their indices to make them trackable among different time instances. Probability distributions are illustrated as purple and blue ellipses where the probability density from low to high is illustrated from blue to purple.

scenario contains between 4 to 6 static obstacles occupying the horizontal lanes of the intersection and up to 4 dynamic obstacles that drive through the intersection in the vertical lanes, where the number of static and dynamic obstacles are randomly selected for each scenario. The starting lanes and initial positions of the dynamic obstacles are also randomly selected for each scenario. The initial speeds of the dynamic obstacles are randomly sampled from between 12[m/s] and 17[m/s]. Similar to Section 4.7.2.1, $q_{i,j}$ is chosen to describe an obstacle's location during $\mathcal{T}_j$ as a Gaussian for all $(i,j) \in \mathcal{I} \times \mathcal{J}$.

Table 4.3 summarizes the results of the left turning experiment. As the allowable risk threshold ϵ increases, we see a reduction in the Average Time To Goal (ATTG) and Maximum Time To Goal (MTTG) as the ego vehicle is less conservative with higher ϵ and thus executes the unprotected left turns more quickly. Additionally, as ϵ increases, the rate of collision also increases.

4.7.3 Hardware

To illustrate the capabilities of RADIUS, we also tested it on a $\frac{1}{10}$ th-scale All-Wheel-Drive car-like robot, Rover, based on a Traxxas RC platform. The first experiment shows RADIUS

ϵ	Success [%]	Crash[%]	ATTG [s]	MTTG [s]
0.01	100.0	0.0	8.868	13.500
0.05	99.9	0.1	8.752	13.500
0.10	99.5	0.5	8.187	13.400
0.20	98.7	1.3	6.085	10.200
0.50	95.6	4.4	5.891	10.000

Table 4.3: Simulation results over a single planning iteration for RADIUS executing a left turn at different risk thresholds. Average Time To Goal (ATTG) and Maximum Time To Goal (MTTG) are only taken over successful trials.

executing a lane change to overtake an obstacle Rover amidst uncertainty in the obstacle’s location with varying risk thresholds. At lower risk thresholds RADIUS is more conservative and is not able to execute the lane change, whereas at higher thresholds RADIUS allows the ego Rover to aggressively overtake the obstacle Rover. The second experiment illustrates a scenario where the obstacle Rover is driven laterally into the path of the ego Rover and it must generate a trajectory with a less than ϵ risk of collision into the obstacle Rover. At lower risk thresholds, RADIUS chooses to be conservative and triggers its contingency braking maneuver before it intersects with the obstacle Rover, unable to reach the set waypoint. However, at higher risk thresholds RADIUS is more aggressive and executes an aggressive lane change in order to get to the waypoint.

4.8 RADIUS 2.0: DIAMETER – Generating a tighter risk over-approximation

While RADIUS provides a tight and efficient risk over-approximation in many scenarios, it can be overly conservative in certain highway driving situations, particularly when the ego vehicle travels closely alongside other vehicles. In these cases, the obstacle vehicle’s probability distribution often places only a small amount of mass on the fringes of the ego vehicle’s zonotope reachable set. RADIUS still partitions and integrates over the entire zonotope, so many of the simplices it evaluates contain low probability mass. This creates relatively large simplices in reference to the amount of probability mass inside the simplices leading RADIUS to decently overestimate collision risk, often causing more cautious planning decisions. This conservatism can hinder maneuvers, such as overtaking, where the perceived risk of driving parallel to another vehicle is inflated, meaning at lower risk thresholds RADIUS tends to be a bit more conservative. To address these limitations, we propose DIAMETER, an alternative algorithm for generating risk over-approximations. DIAMETER retains computational efficiency while producing tighter bounds in such edge cases.

4.8.1 RADIUS’s Conservatism

This section discusses in more detail the challenges with RADIUS’s risk over-approximation highlighted above. We begin by summarizing the set of relaxations employed in RADIUS introduced in Chapter 4.6. First, the chance constraint in (Opt-R) is relaxed conservatively by first over-approximating the domain of integration with a zonotope using Assumption 29. Then this zonotope domain of integration is over-approximated by a collection of right-angled simplices, and the integrand is over-approximated by a quadratic polynomial. Finally, a closed-form equation is leveraged to compute the resulting integral of a quadratic polynomial over a simplex. This is computed in parallel for each simplex to obtain the integral over each zonotope, and thus the integral over the whole zonotope reachable set.

Though this set of relaxations yields a fairly tight over-approximation that outperforms the prior state-of-the-art methods, we observe that the tightness of the over-approximation is greatly affected by the ratio of the simplex size and the probability mass in each simplex. This is a direct result of generating H_S by taking the element-wise supremum of $\text{Hess}_{q_{i,j}}$ over $\mathcal{S} \oplus A_j\mathcal{P}$ using Interval Arithmetic in Lemma 32. Meaning that the larger the simplex $\mathcal{S}$, the wider the range over which Interval Arithmetic must bound the Hessian, leading to a looser (larger) H_S and thus a more inflated quadratic over-approximation of the PDF. When the true probability mass in $\mathcal{S}$ is small, this overshoot causes a disproportionately

large overestimation of the collision risk. This is the case in the scenarios highlighted above where driving alongside other vehicles creates situations where the probability distribution of the obstacle vehicle’s location often places only a small amount of mass on the fringes of the zonotope reachable set of the ego vehicle. Meaning, each simplex only has a small amount of probability mass in it, causing a large over-approximation of the integral once the element-wise supremum of $\text{Hess}_{q_{i,j}}$ over $\mathcal{S} \oplus A_j\mathcal{P}$ is computed. The next subsection introduces DIAMETER and discusses how this method addresses this issue.

4.8.2 DIAMETER’s Risk Over-Approximation Overview

This section describes an implementable alternative to (Opt-R) employed by DIAMETER that can be solved rapidly. In particular, we discuss how DIAMETER relaxes the chance constraint in (Opt-R) to generate a tight over-approximation that avoids undue conservatism. DIAMETER’s approach produces a closed-form over-approximation that is demonstrably tighter than previous methods. This is achieved using an algorithm that dynamically constructs a fine grid of simplices online, based on the amount of overlap between each obstacle’s probability support and the zonotope reachable set, and integrates the probability distribution over this grid to over-approximate the risk of collision. Unlike RADIUS [69], which applies its fixed grid to the entire zonotope, DIAMETER’s dynamic construction confines mesh generation to the overlap region. This allows us to use the same grid resolution, but apply it only to regions with non-trivial probability mass, ensuring that each simplex contains a substantial portion of the relevant probability mass, thus reducing the impact of Hessian over-approximation in these scenarios. The result is a dramatic tightening of the closed-form approximation of the chance constraint while maintaining a similar computational cost. The following sections detail the steps involved in constructing this new over-approximation and explain how the dynamic grid-building algorithm is integrated with the zonotope framework.

4.8.3 DIAMETER’s Chance Constraint Relaxation

4.8.3.1 PDF Integration

Recall that the trajectory of the ego vehicle over the planning horizon is over-approximated by a collection of maps $\{\xi_j\}_{j \in \mathcal{J}}$ and that (Opt-R) is relaxed as stated in Lemma 30.

To further relax the constraint in (4.14) and facilitate the computation of a tighter over-approximation, we extend Assumption 27 as follows:

Assumption 36. For any $\epsilon_0 \in [0, 1]$, $i \in \mathcal{I}$ and $j \in \mathcal{J}$, there exists a zonotope $\mathcal{Q}_{i,j}(\epsilon_0) \subset \mathcal{W}$ such that

$$\int_{\mathcal{Q}_{i,j}(\epsilon_0)} q_{i,j}(w) dw \in [1 - \epsilon_0, 1]. \quad (4.25)$$

For ϵ_0 near 0, Assumption 36 assumes the existence of a zonotope that captures the majority of the probability mass of a PDF $q_{i,j}$. Such a zonotope $\mathcal{Q}_{i,j}(\epsilon_0)$ can often be constructed from basic distributional information, even though no universal method yields exact approximations for arbitrary distributions. For example, multivariate Gaussians admit exact ellipsoidal confidence regions containing a prescribed mass, which can be overapproximated by zonotopes. Even minimal information, such as mean and covariance, is sufficient to apply Chebyshev-type bounds to obtain conservative high-probability regions [78]. If the CDF is available, one can construct a box containing at least $1 - \epsilon_0$ of the mass; because boxes are special cases of zonotopes, this gives a direct construction. For notational convenience, we suppress the dependency on ϵ_0 in $\mathcal{Q}_{i,j}$ in the remainder of this chapter. With this extension, we now present the following lemma, which conservatively over-approximates the integral in (4.14).

Lemma 37. Given a zonotope $\mathcal{Q}_{i,j}$ that satisfies Assumption 36, the integral in (4.14) can be conservatively bounded as:

$$\int_{\xi_j(z_0, p) \oplus <0, G^{\text{obs}}>} q_{i,j}(w) dw \leq \epsilon_0 + \int_{(\xi_j(z_0, p) \oplus <0, G^{\text{obs}}>) \cap \mathcal{Q}_{i,j}} q_{i,j}(w) dw. \quad (4.26)$$

Proof. Denote $\mathcal{Q}_{i,j}^C$ as the complement of $\mathcal{Q}_{i,j}$ in $\mathcal{W}$ for arbitrary $i \in \mathcal{I}$ and $j \in \mathcal{J}$. Note:

$$\xi_j(z_0, p) \oplus <0, G^{\text{obs}}> = ((\xi_j(z_0, p) \oplus <0, G^{\text{obs}}>) \cap \mathcal{Q}_{i,j}^C) \cup ((\xi_j(z_0, p) \oplus <0, G^{\text{obs}}>) \cap \mathcal{Q}_{i,j}) \quad (4.27)$$

Thus we can decompose the integral as

$$\begin{aligned} \int_{\xi_j(z_0, p) \oplus <0, G^{\text{obs}}>} q_{i,j}(w) dw &= \int_{(\xi_j(z_0, p) \oplus <0, G^{\text{obs}}>) \cap \mathcal{Q}_{i,j}^C} q_{i,j}(w) dw \\ &\quad + \int_{(\xi_j(z_0, p) \oplus <0, G^{\text{obs}}>) \cap \mathcal{Q}_{i,j}} q_{i,j}(w) dw \end{aligned} \quad (4.28)$$

By Assumption 36, the total probability mass outside $\mathcal{Q}_{i,j}$ is at most ϵ_0 , i.e.

$$\int_{\mathcal{Q}_{i,j}^C} q_{i,j}(w) dw \leq \epsilon_0 \quad (4.29)$$

and because

$$((\xi_j(z_0, p) \oplus <0, G^{\text{obs}}>) \cap \mathcal{Q}_{i,j}^{\text{C}}) \subset \mathcal{Q}_{i,j}^{\text{C}}, \quad (4.30)$$

it follows that

$$\int_{(\xi_j(z_0, p) \oplus <0, G^{\text{obs}}>) \cap \mathcal{Q}_{i,j}^{\text{C}}} q_{i,j}(w) dw \leq \epsilon_0 \quad (4.31)$$

then Assumption 36 implies

$$\int_{\xi_j(z_0, p) \oplus <0, G^{\text{obs}}>} q_{i,j}(w) dw \leq \epsilon_0 + \int_{(\xi_j(z_0, p) \oplus <0, G^{\text{obs}}>) \cap \mathcal{Q}_{i,j}} q_{i,j}(w) dw.$$

□

This proposed relaxation in (4.26) allows us to isolate a portion of $\xi_j(z_0, p) \oplus <0, G^{\text{obs}}>$ and only integrate over the region $(\xi_j(z_0, p) \oplus <0, G^{\text{obs}}>) \cap \mathcal{Q}_{i,j}$, which has a non-trivial probability mass, while conservatively accounting the leftover probability mass ϵ_0 . This is more beneficial than the approach used in RADIUS, which integrates over the entire zonotope $\xi_j(z_0, p) \oplus <0, G^{\text{obs}}>$. Specifically, because $(\xi_j(z_0, p) \oplus <0, G^{\text{obs}}>) \cap \mathcal{Q}_{i,j}$ is a subset of $\xi_j(z_0, p) \oplus <0, G^{\text{obs}}>$, integrating over the first set is likely to result in simplices of smaller size than the second set if each of these two sets is covered by a k -by- k grid with some fixed positive integer k .

4.8.3.2 Domain Relaxation

Our goal is to efficiently over-approximate

$$\int_{(\xi_j(z_0, p) \oplus <0, G^{\text{obs}}>) \cap \mathcal{Q}_{i,j}} q_{i,j}(w) dw$$

by covering the integration domain with a finite collection of simplices, similar in spirit to [69]. However, this intersection area

$$(\xi_j(z_0, p) \oplus <0, G^{\text{obs}}>) \cap \mathcal{Q}_{i,j}$$

varies in both size and position with p , meaning that one would need to reconstruct this grid of simplices for each candidate p every optimization step, which would introduce a large computational overhead. To avoid this, we exploit the affine-in- p structure of the zonotopes guaranteed by Assumption 29:

$$\xi_j(z_0, p) = <c_j(z_0) + A_j \cdot p, G_j> = <c_j(z_0), G_j> + A_j \cdot p$$

This means that the reachable set can be represented as some p -independent set of zonotopes that can be linearly transformed as a function of p . We leverage this property to avoid having to rebuild the grid each step of the optimization by generating a collection of simplices to cover the domain of integration $(\xi_j(z_0, p) \oplus \langle 0, G^{\text{obs}} \rangle) \cap \mathcal{Q}_{i,j}$. Specifically, we first cover a p -independent region, and then account for the effect of p efficiently by applying the linear map given by the matrix A_j .

We start by introducing two lemmas that are helpful to illustrate how p can be isolated in the domain of integration of this intersection region.

Lemma 38. *For arbitrary sets $\mathcal{A}$, $\mathcal{B}$ and vector ν in $\mathcal{W}$,*

$$(\mathcal{A} + \nu) \cap \mathcal{B} = (\mathcal{A} \cap (\mathcal{B} - \nu)) + \nu. \quad (4.32)$$

Proof. For any $\alpha \in \mathcal{A}$ and $\beta \in \mathcal{B}$ such that $\alpha + \nu = \beta \in (\mathcal{A} + \nu) \cap \mathcal{B}$, observe that $\alpha = \beta - \nu \in \mathcal{A} \cap (\mathcal{B} - \nu)$ and $\beta = (\beta - \nu) + \nu$. $\square$

Lemma 39. *For any $j \in \mathcal{J}$, $p \in \mathcal{P} = \text{int}(\underline{p}, \bar{p})$, and A_j as assumed in Assumption 29, let $p^{\text{ctr}} := \frac{1}{2}(\underline{p} + \bar{p})$, then there exists some linear function $\lambda_j : \mathcal{P} \rightarrow [-1, 1]^{n_p}$ and some matrix $G_j^p \in \mathbb{R}^{2 \times n_p}$ such that $\lambda_j(p^{\text{ctr}}) = 0$ and*

$$A_j \cdot p = G_j^p \cdot \lambda_j(p) + A_j \cdot p^{\text{ctr}}. \quad (4.33)$$

Proof. One can easily check that $\lambda_j(p) = 2(\text{diag}(\bar{p} - \underline{p}))^{-1} \cdot (p - p^{\text{ctr}})$ and $G_j^p = \frac{1}{2}A_j \cdot \text{diag}(\bar{p} - \underline{p})$ satisfy (4.33). $\square$

We now describe how to isolate p from the set intersection $(\xi_j(z_0, p) \oplus \langle 0, G^{\text{obs}} \rangle) \cap \mathcal{Q}_{i,j}$ in the following theorem:

Theorem 40. *For any $i \in \mathcal{I}$, $j \in \mathcal{J}$, $z_0 \in \mathcal{Z}_0$ and $p \in \mathcal{P}$, let ξ_j , c_j , A_j and G_j satisfy Assumption 29, let G^{obs} satisfy Assumption 27, let $\mathcal{Q}_{i,j}$ satisfy Assumption 36, and let p^{ctr} and G_j^p be as defined in Lemma 39, and let $\xi_{i,j}^{\mathcal{Q}}(z_0)$ be defined as the intersection between $\langle c_j(z_0) + A_j p^{\text{ctr}}, [G_j, G^{\text{obs}}] \rangle$ and $\mathcal{Q}_{i,j} \oplus \langle 0, G_j^p \rangle$, then*

$$(\xi_j(z_0, p) \oplus \langle 0, G^{\text{obs}} \rangle) \cap \mathcal{Q}_{i,j} \subseteq \xi_{i,j}^{\mathcal{Q}}(z_0) + A_j(p - p^{\text{ctr}}) \quad (4.34)$$

Proof. Notice that

$$\begin{aligned}
& (\xi_j(z_0, p) \oplus \langle 0, G^{\text{obs}} \rangle) \cap \mathcal{Q}_{i,j} \\
&= \langle c_j(z_0) + A_j p, [G_j, G^{\text{obs}}] \rangle \cap \mathcal{Q}_{i,j} \\
&= (\langle c_j(z_0) + A_j p^{\text{ctr}}, [G_j, G^{\text{obs}}] \rangle + A_j(p - p^{\text{ctr}})) \cap \mathcal{Q}_{i,j} \\
&= \langle c_j(z_0) + A_j p^{\text{ctr}}, [G_j, G^{\text{obs}}] \rangle \cap (\mathcal{Q}_{i,j} - A_j(p - p^{\text{ctr}})) + \\
&\quad + A_j(p - p^{\text{ctr}}) \\
&\subseteq \langle c_j(z_0) + A_j p^{\text{ctr}}, [G_j, G^{\text{obs}}] \rangle \cap (\mathcal{Q}_{i,j} \oplus \langle 0, G_j^{\text{p}} \rangle) + \\
&\quad + A_j(p - p^{\text{ctr}}) \\
&= \xi_{i,j}^{\mathcal{Q}}(z_0) + A_j(p - p^{\text{ctr}})
\end{aligned} \tag{4.35}$$

where the first equality follows from Assumption 29, the second equality follows from the definition of zonotopes, the third equality follows from Lemma 38, the inclusion follows from (4.33) and $\lambda_j(p) \in [-1, 1]^{n_p}$ in Lemma 39, and the last equality follows from the definition of $\xi_{i,j}^{\mathcal{Q}}(z_0)$. $\square$

Theorem 40 shows that we can conservatively over-approximate $(\xi_j(z_0, p) \oplus \langle 0, G^{\text{obs}} \rangle) \cap \mathcal{Q}_{i,j}$ by over-approximating the area p -independent area $\xi_{i,j}^{\mathcal{Q}}(z_0)$ in $\mathcal{W}$ and accounting for the value of p efficiently through matrix A_j as a linear map.

Next, similar to RADIUS, we construct a k -by- k grid that covers zonotope $\xi_{i,j}^{\mathcal{Q}}(z_0)$ where k is some user-specified positive integer. Once again, each cell in the grid shares the same size and is indexed by its row and column index in the grid and each cell is further divided into two simplexes as right-angled triangles that are indexed by 1 or -1 corresponding to the lower or upper triangles of the cell, respectively. Let $\mathcal{S}_{i,j,k_1,k_2,k_3}(z_0) \subset \mathcal{W}$ now be the simplex indexed by $k_3 \in \{-1, 1\}$ in the cell on the k_1 -th row and k_2 -th column of the grid that covers the p -independent area $\xi_{i,j}^{\mathcal{Q}}(z_0)$. Define

$$\begin{aligned}
\bar{\mathcal{S}}_{i,j}^{\mathcal{Q}}(z_0) &:= \{ \mathcal{S}_{i,j,k_1,k_2,k_3}(z_0) \mid k_1, k_2 \in \{1, 2, \dots, k\}, \\
&\quad |k_3| = 1, \mathcal{S}_{i,j,k_1,k_2,k_3}(z_0) \cap \xi_{i,j}^{\mathcal{Q}}(z_0) \neq \emptyset \}.
\end{aligned} \tag{4.36}$$

as the collection of every possible $\mathcal{S}_{i,j,k_1,k_2,k_3}(z_0)$ that intersects with $\xi_{i,j}^{\mathcal{Q}}(z_0)$, thus

$$\xi_{i,j}^{\mathcal{Q}}(z_0) \subset (\cup_{\mathcal{S} \in \bar{\mathcal{S}}_{i,j}^{\mathcal{Q}}(z_0)} \mathcal{S}). \tag{4.37}$$

Remark 41. Note that for any $\bar{\mathcal{S}}_{i,j}^{\mathcal{Q}}(z_0)$, $\mathcal{S}$ depends on z_0 , i and j . However, to reduce notational complexity, we drop its dependency on z_0 , i and j in the remainder of this chapter.

From Theorem 40 and (4.37), notice that with a collection $\bar{\mathcal{S}}_{i,j}^{\mathcal{Q}}$ of simplices that covers the p -independent area $\xi_{i,j}^{\mathcal{Q}}(z_0)$ in $\mathcal{W}$, the following inclusion holds:

$$(\xi_j(z_0, p) \oplus \langle 0, G^{\text{obs}} \rangle) \cap \mathcal{Q}_{i,j} \subseteq \cup_{\mathcal{S} \in \bar{\mathcal{S}}_{i,j}^{\mathcal{Q}}} \mathcal{S} + A_j(p - p^{\text{ctr}}). \quad (4.38)$$

and

$$\int_{\xi_j(z_0, p) \oplus \langle 0, G^{\text{obs}} \rangle \cap \mathcal{Q}_{i,j}} q_{i,j}(w) dw \leq \sum_{\mathcal{S} \in \bar{\mathcal{S}}_{i,j}^{\mathcal{Q}}(z_0)} \int_{\mathcal{S} + A_j(p - p^{\text{ctr}})} q_{i,j}(w) dw. \quad (4.39)$$

The next steps in computing this over-approximation leverage the results introduced earlier in Chapter 4.6.1.3, which we restate in a modified form in the next two sections for convenience.

4.8.3.3 Integrand Relaxation

Next, we present a lemma to conservatively approximate the integral in the chance constraint by relaxing the integrand. The proof of this lemma follows directly from the proof of Lemma 32 in Appendix C.1.

Lemma 42. *Suppose the ego vehicle starts with initial condition $z_0 \in \mathcal{Z}_0$ and control parameter $p \in \mathcal{P}$. Let $\mathcal{S} \in \bar{\mathcal{S}}_{i,j}^{\mathcal{Q}}(z_0)$ and let $w_{\mathcal{S}}^{\text{ctr}}$ denote the vertex of the right angle in $\mathcal{S}$. Define the function $q_{i,j,\mathcal{S}} : \mathcal{W} \times \mathcal{P} \rightarrow [0, \infty)$ as*

$$\begin{aligned} q_{i,j,\mathcal{S}}(w, p) &:= q_{i,j}(w_{\mathcal{S}}^{\text{ctr}} + A_j(p - p^{\text{ctr}})) + \\ &\quad + \frac{\partial q_{i,j}}{\partial w}(w_{\mathcal{S}}^{\text{ctr}} + A_j(p - p^{\text{ctr}})) \cdot \\ &\quad \cdot (w - w_{\mathcal{S}}^{\text{ctr}} - A_j(p - p^{\text{ctr}})) + \\ &\quad + \frac{1}{2}(w - w_{\mathcal{S}}^{\text{ctr}} - A_j(p - p^{\text{ctr}}))^{\top} \cdot \\ &\quad \cdot H_{\mathcal{S}} \cdot (w - w_{\mathcal{S}}^{\text{ctr}} - A_j(p - p^{\text{ctr}})), \end{aligned} \quad (4.40)$$

where $H_{\mathcal{S}} \in \mathbb{R}^{2 \times 2}$ is generated by taking element-wise supremum of the Hessian of $q_{i,j}$ over $\mathcal{S} \oplus A_j\mathcal{P}$ using Interval Arithmetic [44]. Then for all $w \in \mathcal{S} + A_jp$:

$$q_{i,j}(w) \leq q_{i,j,\mathcal{S}}(w, p). \quad (4.41)$$

4.8.3.4 Closed-form Computation

As a result of (4.39) and (4.41), the following inequality holds:

$$\int_{\xi_j(z_0, p) \oplus <0, G^{\text{obs}}> \cap \mathcal{Q}_{i,j}} q_{i,j}(w) dw \leq \sum_{\mathcal{S} \in \bar{\mathcal{S}}_{i,j}^{\mathcal{Q}}(z_0)} \int_{\mathcal{S} + A_j(p - p^{\text{ctr}})} q_{i,j,\mathcal{S}}(w, p) dw. \quad (4.42)$$

Notice that $q_{i,j,\mathcal{S}}(w, p)$ defined in (4.41) is indeed a quadratic polynomial of w , and $\mathcal{S} + A_j(p - p^{\text{ctr}})$ is a simplex in $\mathbb{R}^2$. One can then compute $\int_{\mathcal{S} + A_j(p - p^{\text{ctr}})} q_{i,j,\mathcal{S}}(w, p) dw$ in closed-form as follows:

Theorem 43. *For any $i \in \mathcal{I}$, $j \in \mathcal{J}$, $z_0 \in \mathcal{Z}_0$, $p \in \mathcal{P}$ and $\mathcal{S} \in \bar{\mathcal{S}}_{i,j}^{\mathcal{Q}}(z_0)$, let $A_j \in \mathbb{R}^{2 \times n_p}$ be defined as in Assumption 29, let $q_{i,j,\mathcal{S}}$ be defined as in (4.41), and let $\mathbf{i} \mathbf{d} \mathbf{x}_{k_3}(\mathcal{S})$ denote the last index of its argument (i.e., $\mathbf{i} \mathbf{d} \mathbf{x}_{k_3}(\mathcal{S}_{j,k_1,k_2,k_3}(z_0)) = k_3$). Assume that positive numbers l_1 and l_2 give the lengths of horizontal and vertical right angle sides of $\mathcal{S}$ respectively, then*

$$\begin{aligned} & \int_{\mathcal{S} + A_j(p - p^{\text{ctr}})} q_{i,j,\mathcal{S}}(w, p) dw = \\ &= \frac{1}{2 \det(A_{\mathcal{S}})} \left(q_{i,j}(w_{\mathcal{S}}^{\text{ctr}} + A_j(p - p^{\text{ctr}})) + \right. \\ &+ \left[\frac{1}{4\sqrt{3}} \quad \frac{1}{4\sqrt{3}} \right] \left(\hat{H}_{\mathcal{S}} \odot \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \right) \begin{bmatrix} \frac{1}{2\sqrt{3}} \\ \frac{1}{2\sqrt{3}} \end{bmatrix} + \\ &+ \left. \frac{\partial q_{i,j}}{\partial w}(w_{\mathcal{S}}^{\text{ctr}} + A_j(p - p^{\text{ctr}})) \cdot A_{\mathcal{S}}^{-1} \cdot \begin{bmatrix} \frac{1}{3} \\ \frac{1}{3} \end{bmatrix} \right) \end{aligned} \quad (4.43)$$

where $\odot$ denotes the element-wise multiplication and

$$A_{\mathcal{S}} = \begin{bmatrix} \mathbf{i} \mathbf{d} \mathbf{x}_{k_3}(\mathcal{S})/l_1 & 0 \\ 0 & \mathbf{i} \mathbf{d} \mathbf{x}_{k_3}(\mathcal{S})/l_2 \end{bmatrix}, \quad (4.44)$$

$$\hat{H}_{\mathcal{S}} = A_{\mathcal{S}}^{-\top} H_{\mathcal{S}} A_{\mathcal{S}}^{-1}. \quad (4.45)$$

Proof. The claim follows from [65, Theorem 1.1] and the fact that $A_{\mathcal{S}} \cdot ((\mathcal{S} + A_j(p - p^{\text{ctr}})) - (w_{\mathcal{S}}^{\text{ctr}} + A_j(p - p^{\text{ctr}})))$ equals the canonical simplex $\Delta := \{(a, b) \in \mathbb{R}^2 \mid a + b \leq 1, a \geq 0, b \geq 0\}$. $\square$

4.8.4 DIAMETER's Tractable Online Optimization

The computation in Section 4.8.3 provides a tractable way to enforce not-at-fault behavior compared to the original chance constraint in (0pt-R). As a result, DIAMETER solves the

following optimization during online planning:

$$\begin{aligned}
& \min_{p \in \mathcal{P}} \text{cost}(z_0, p) && (\text{Opt-D}) \\
& \text{s.t.} \quad \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} \sum_{\mathcal{S} \in \bar{\mathcal{S}}_{i,j}^{\mathcal{Q}}(z_0)} \int_{\mathcal{S} + A_j(p - p^{\text{ctr}})} q_{i,j,\mathcal{S}}(w, p) \, dw \leq \epsilon.
\end{aligned}$$

(Opt-D) is a strengthened version of (Opt-R) because satisfaction of the chance constraint in (Opt-D) implies satisfaction of the chance constraint in (Opt-R) by Lemma 30, (4.37), and (4.41). Thus, the following lemma holds based on Definition 28.

Lemma 44. *If the ego vehicle applies any feasible solution, $p^* \in \mathcal{P}$, of (Opt-D) beginning from $z_0 \in \mathcal{Z}_0$ at $t = 0$, then it is not-at-fault with a risk of collision at most ϵ during $[0, t_f]$.*

4.8.5 DIAMETER's Constraint Gradient and Parallelization

To improve the solving procedure of (Opt-D), we provide the derivative of its chance constraint. To compute the gradient of the chance constraint in (Opt-D), it suffices to compute the derivative of $\int_{\mathcal{S} + A_j(p - p^{\text{ctr}})} q_{i,j,\mathcal{S}}(w, p) \, dw$ in (4.43) with respect to p . Notice that $A_{\mathcal{S}}$ and $\hat{H}_{\mathcal{S}}$ are invariant over $\mathcal{P}$, then

$$\begin{aligned}
\frac{\partial}{\partial p} \int_{\mathcal{S} + A_j(p - p^{\text{ctr}})} q_{i,j,\mathcal{S}}(w, p) \, dw &= \frac{1}{2 \det(A_{\mathcal{S}})} \cdot \left(\frac{\partial q_{i,j}}{\partial w}(w_{\mathcal{S}}^{\text{ctr}} + A_j(p - p^{\text{ctr}})) \cdot A_j + \right. \\
&\quad \left. + \begin{bmatrix} \frac{1}{3} & \frac{1}{3} \end{bmatrix} A_{\mathcal{S}}^{-\top} \text{Hess}_{q_{i,j}}(w_{\mathcal{S}}^{\text{ctr}} + A_j(p - p^{\text{ctr}})) \cdot A_j \right), \tag{4.46}
\end{aligned}$$

where $\text{Hess}_{q_{i,j}}(w_{\mathcal{S}}^{\text{ctr}} + A_j(p - p^{\text{ctr}}))$ is the evaluation of the Hessian of $q_{i,j}$ at $w_{\mathcal{S}}^{\text{ctr}} + A_j(p - p^{\text{ctr}})$.

Notice the computations of $\int_{\mathcal{S} + A_j(p - p^{\text{ctr}})} q_{i,j,\mathcal{S}}(w, p) \, dw$ and its gradient can be parallelized for all $(i, j) \in \mathcal{I} \times \mathcal{J}$ and all $\mathcal{S} \in \bar{\mathcal{S}}_{i,j}^{\mathcal{Q}}(z_0)$ as indicated in Algorithm 6. $\bar{\mathcal{S}}_{i,j}^{\mathcal{Q}}(z_0)$ is generated in Line 1 for all $(i, j) \in \mathcal{I} \times \mathcal{J}$. The outer parallel for loop that starts at Line 2 iterates every element in $\mathcal{I} \times \mathcal{J}$, and the inner parallel for loop that starts at Line 3 iterates over each simplex in $\bar{\mathcal{S}}_{i,j}^{\mathcal{Q}}(z_0)$. The integral and its gradient over each simplex are computed from Lines 4 to 5. Then the chance constraint and its gradient in (Opt-D) can be computed as the summation of all computed $\int_{\mathcal{S} + A_j(p - p^{\text{ctr}})} q_{i,j,\mathcal{S}}(w, p) \, dw$ and $\frac{\partial}{\partial p} \int_{\mathcal{S} + A_j(p - p^{\text{ctr}})} q_{i,j,\mathcal{S}}(w, p) \, dw$ respectively.

4.8.6 Evaluating DIAMETER's Risk-Overapproximation

In this section we evaluate DIAMETER's risk over-approximation by performing a modified version of the experiment performed in Chapter 4.7.1. In this experiment, we compare the

Algorithm 6 Chance Constraint Parallelization

Require: $z_0 \in \mathcal{Z}_0$, $p \in \mathcal{P}$, $\{q_{i,j}\}_{(i,j) \in \mathcal{I} \times \mathcal{J}}$, $\{\xi_j\}_{j \in \mathcal{J}}$

- 1: **Generate** $\{\bar{\mathcal{S}}_{i,j}^Q(z_0)\}_{(i,j) \in \mathcal{I} \times \mathcal{J}}$
- 2: **Parfor** $(i, j) \in \mathcal{I} \times \mathcal{J}$ **do**
- 3: **Parfor** $\mathcal{S} \in \bar{\mathcal{S}}_{i,j}^Q(z_0)$ **do**
- 4: **Compute** $\int_{\mathcal{S}+A_j(p-p^{\text{ctr}})} q_{i,j,\mathcal{S}}(w, p) dw$ as in (4.43)
- 5: **Compute** $\frac{\partial}{\partial p} \int_{\mathcal{S}+A_j(p-p^{\text{ctr}})} q_{i,j,\mathcal{S}}(w, p) dw$ as in (4.46)
- 6: **End Parfor**
- 7: **End Parfor**

DIAMETER’s approximation of the risk of collision against, Monte-Carlo integration [14, Chapter 4], which is used as the ground truth risk of collision, RADIUS [69], the Cantelli inequality [115, Section IV B], and the Chance-Constrained Parallel Bernstein Algorithm (CCPBA) [111, Chapter 6] on 12,000 new randomly generated test cases. In each of the test cases, (i, j, z_0, p) is randomly chosen from $\mathcal{I} \times \mathcal{J} \times \mathcal{Z}_0 \times \mathcal{P}$. In the first 3000 we deliberately confine only a small fraction of a Gaussian PDF’s mass to lie in $\xi_j(z_0, p) \oplus <0, G^{\text{obs}}>$, to simulate the scenarios where the vehicle drives alongside obstacle vehicles and has a small amount of probability mass inside the zonotope reachable set. We refer to these cases as the **Gaussian-S** cases. The other 9,000 test cases are split evenly across three distribution types: a randomly generated 2-dimensional (2D) Gaussian, a 2D Beta distribution [86], and a 2D bimodal distribution (3,000 cases per type). For these cases we permit a large fraction of each distribution’s mass to lie within $\xi_j(z_0, p) \oplus <0, G^{\text{obs}}>$. Once again, note that because CCPBA is unable to handle a distribution that is not Gaussian, we only evaluate DIAMETER, RADIUS and the Cantelli inequality on these non-Gaussian probability distributions. Once again, we perform the Monte-Carlo integration of $q_{i,j}$ over $\xi_j(z_0, p) \oplus <0, G^{\text{obs}}>$ with 10^6 samples to represent the ground truth. The risk approximation error, which is the difference between the Monte-Carlo integration and each method, is illustrated in Table 4.4. The results from Table 4.4 show that DIAMETER provides significantly tighter upper bounds to the ground truth than, RADIUS, CCPBA and the Cantelli inequality for the tested probability distributions.

4.9 Conclusion

This chapter proposes RADIUS (and its variant DIAMETER) as real-time, risk-aware trajectory planners for autonomous vehicles operating in uncertain and dynamic environments. It also proposes two methods for computing a tight, differentiable, closed-form over-approximation of the risk of collision with an obstacle (the original RADIUS approxima-

Method	Stat	Gaussian-S	Gaussian	Beta	Bimodal
DIAMETER	Mean	0.0004	0.0178	0.0015	0.0195
	Max	0.0020	0.5501	0.0340	0.6767
RADIUS	Mean	0.0181	0.0200	0.0015	0.0199
	Max	0.1718	0.7453	0.0340	0.8831
Cantelli	Mean	0.7766	0.5501	0.3981	0.5625
	Max	0.9988	0.9991	0.9612	0.9990
CCPBA	Mean	0.4987	0.2165	-	-
	Max	0.9973	0.9563	-	-

Table 4.4: Mean and maximum errors in estimated collision risk (relative to Monte-Carlo ground truth) for each method, evaluated over four different test cases with different obstacle distributions: Gaussian-S (only a small fraction of Gaussian probability mass lies inside each zonotope), Gaussian, Beta, and Bimodal distributions, where each are allowed to have more substantial probability mass in each zonotope. CCPBA is only applicable to Gaussian cases.

tion and the alternative DIAMETER approximation). Given some allowable risk threshold (ϵ), the proposed planners use their respective over-approximations, in conjunction with reachability-based methods to generate motion plans that have a risk of collision no greater than ϵ with any obstacles for the duration of the planned trajectory. The chapters shows that RADIUS is able to outperform the existing state-of-the-art chance-constrained and deterministic approaches in a variety of driving scenarios while achieving real-time performance, and that DIAMETER is able to generate tighter risk-overapproximations than RADIUS and the existing state-of-the-art chance-constrained approaches.

CHAPTER 5

Let’s Make a Splan: Risk-Aware Trajectory Optimization in a Normalized Gaussian Splat

5.1 Introduction

For a robot to safely navigate its environment, it must understand the scene geometry it operates within. This understanding must include a detailed model of the scene and a method to reason about collisions with the environment. Radiance field representations, such as Neural Radiance Fields (NeRFs [85]) and Gaussian Splatting [56], have recently emerged as powerful methods for building detailed models of the scene. A radiance field is a five-dimensional function that maps a 3D point and viewing direction to an RGB color and volume rendering opacity. This function is then integrated along camera rays to approximate the image formation process. NeRFs use neural networks to learn the parameters of a radiance field, while Gaussian Splatting models use a set of 3D Gaussian functions. Over the past several years, radiance field representations have marked a paradigm shift in computer vision, with wide-ranging impacts on scene reconstruction [125, 91], novel view synthesis [85], 3D tracking [123, 118], and more.

The robotics research community has begun trying to integrate these models for robotic tasks such as localization [68], mapping [103, 50], and navigation [1]. A key strength of radiance models, such as 3D Gaussian Splats (3DGS), is that they represent the 3D scene using a continuous basis set. This differs from conventional discrete representations in robotics, such as point clouds and occupancy grids. In particular, point clouds do not inherently encode surface or object connectivity because they are a sampled representation of the environment. As a result, it’s unclear how to derive a probabilistic interpretation for the occupancy of a continuous portion of the scene from a point cloud. Though voxel grids do have a direct volumetric and probabilistic interpretation [47], the resolution of the map must be selected a-priori, which presents a strong tradeoff between fidelity and computational demands.



Figure 5.1: SPLANNING constructs risk-aware trajectories in a Gaussian Splatting map in real-time. The top image shows a real-world scene that a 7DOF serial manipulator must plan through, starting at the **blue** configuration (left) and ending at the **green** configuration (right). The scene is represented as a normalized 3D Gaussian Splat. Then, in real-time, SPLANNING solves an optimization problem that constrains the probability that the robot’s forward occupancy (bottom center, **purple**) collides with the scene.

In addition, to ensure safe planning, voxel-based methods typically employ obstacle buffering to account for the robot’s geometry. This is straightforward to perform if the robot’s footprint can be modeled as a point or sphere; however, this buffering operation is challenging to perform for articulated robots without unnecessarily restricting free space [74]. On the other hand, just as with point clouds, one could process the voxel grid to create a surface model such as a signed distance field. However, this introduces additional computational overhead and potential inaccuracies in collision checking [26]. Despite representing the scene

using a continuous basis set, reasoning rigorously about collisions in 3DGS representations is challenging. While existing planners offer practical solutions to this problem, such as discretizing the robot body [1], discretizing the map before planning [18], or treating the confidence ellipsoids of the Gaussians in a Gaussian Splat as obstacles [19], work remains to fully exploit the continuous nature of radiance field models.

This chapter proposes a novel re-formulation of 3DGS enabling collision checking that is computationally simple and fully differentiable, allowing it to be incorporated into a trajectory optimizer and gradient-based learning methods. Notably, we demonstrate that our collision avoidance representation is more accurate than existing methods. Further, we describe a planning framework that works with serial robot manipulators, unlike prior risk-aware planners for radiance field models that operate on quadrotors [1, 18, 19]. The proposed trajectory optimization-based risk-aware planner that leverages this re-formulated 3DGS method is called SPLANNING¹².

5.1.1 Related Work

SPLANNING is an algorithm combining trajectory optimization, reachability analysis for robot safety, and radiance fields for 3D scene representation. We discuss the relevant literature here.

5.1.1.1 Trajectory Optimization

To generate safe motion plans, state-of-the-art trajectory optimizers such as CHOMP [128], TrajOpt [94], MPOT [66], and cuRobo [104] model the robot or the environment with simple geometric primitives such as spheres [24, 36], ellipsoids [12], capsules [23, 25], or convex polygons and perform collision-checking along a given trajectory at discrete time instances. CHOMP represents the robot as a collection of discrete spheres and avoids collisions using a signed distance field to maintain a safety margin with the environment. TrajOpt uses the support mapping of convex shapes to represent the environment obstacles and the robot. Then, the signed distance (positive distance and penetration depth) between two convex shapes is computed by the Gilbert-Johnson-Keerthi [39] and Expanding Polytope Algorithms [110]. MPOT represents the robot geometry and environment obstacles as a collection of spheres and implements a collision-avoidance cost using an occupancy map [66]. Using a gradient-free approach, MPOT optimizes a batch of smooth trajectories and selects the one with the lowest cost. Similarly, a recent method called cuRobo [104] represents

¹Published in IEEE Transactions on Robotics, 41:4380–4397, 2025

²Code found at <https://roahmlab.github.io/splanning/>

the robot as a collection of spheres and solves multiple trajectory optimization problems in parallel on the GPU to identify a collision-free path. In each of these approaches, collision avoidance is enforced using a soft penalty in the cost function. The major drawback of these approaches is that the resulting trajectories are not guaranteed to be safe. Furthermore, these approaches require explicit representations of scene geometry, such as zonotopes, point clouds, or 3D occupancy grids. Each of these methods suffers from a fundamental limitation: zonotopes are challenging to construct from sensor data, point clouds lack a clear volumetric and probabilistic interpretation, and 3D occupancy grids’ computational requirements grow cubically with the resolution and size of the map. In contrast, SPLANNING enforces collision avoidance as a constraint and operates directly in a learned Gaussian Splatting model. In addition to being constructible directly from sensor data, Gaussian Splats allow for photorealistic rendering of the scene.

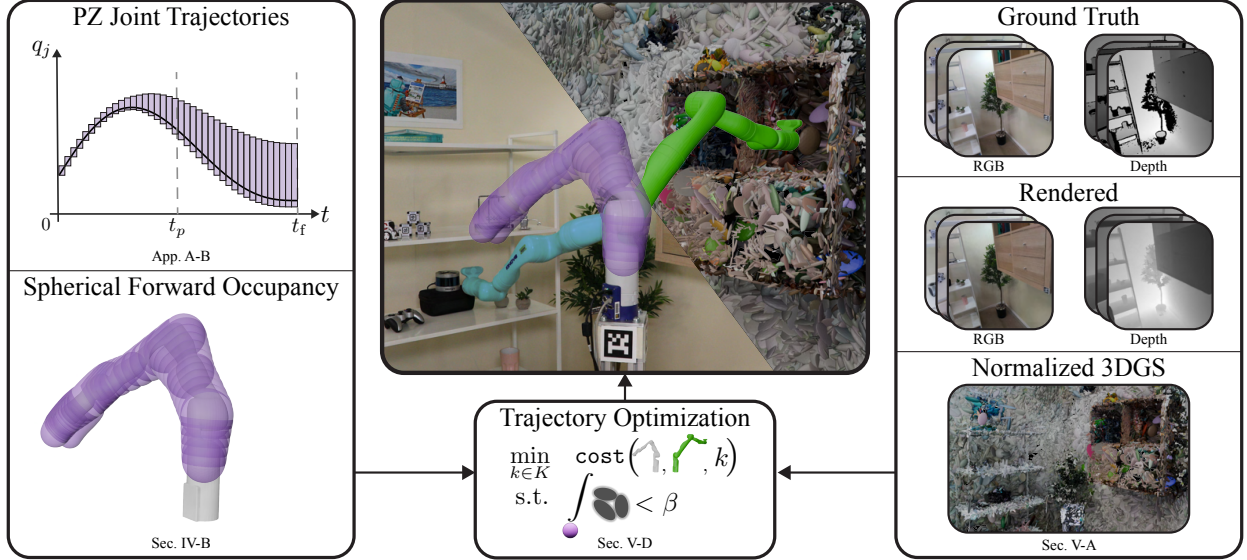


Figure 5.2: SPLANNING optimizes trajectories in a Normalized 3D Gaussian Splat given a start configuration (blue) and goal configuration (green). Offline, a Normalized 3D Gaussian Splat is constructed to represent the scene geometry (Sec. 5.4.1, bottom right). Online, a family of parameterized trajectories (App. D.1) is partitioned into a finite set of intervals (App. D.2, top left). Then, for each time interval, the Spherical Forward Occupancy ($\mathcal{SFO}$) (purple) is computed as an overapproximation of the robot’s swept volume (Sec. 5.3.2, bottom left). Finally, during online trajectory optimization (Sec. 5.4.4, bottom middle), a novel constraint (Sec. 5.4.2–5.4.3) bounds the probability that the $\mathcal{SFO}$ intersects with the scene, as represented by a Normalized 3D Gaussian Splat.

5.1.1.2 Reachability Analysis

Reachability-based Trajectory Design (RTD) [61] is a recent approach to real-time motion planning that generates provably-safe trajectories in a receding-horizon fashion. At run-time, RTD constructs reachable sets that overapproximate all possible robot configurations corresponding to a pre-specified continuum of parameterized trajectories. RTD then solves a nonlinear optimization problem to select a feasible trajectory such that the robot’s motion is guaranteed to be collision-free. If a feasible trajectory is not found, RTD brings the robot safely to a stop by executing a fail-safe braking maneuver. Unlike traditional trajectory optimization methods, RTD constructs reachable sets such that obstacle-avoidance constraints are satisfied in *continuous-time*. Recent extensions of RTD have demonstrated real-time, certifiably-safe motion planning for robotic arms [45, 81, 83, 11, 84] and mobile robots [70, 80]. Probabilistic extensions of RTD have also been proposed [69], which provide safety guarantees in uncertain environments. However, these reachability-based methods assume ground-truth knowledge of obstacle geometry or that probability density functions on the locations of obstacles are provided. SPLANNING extends the literature on reachability analysis by presenting a reachability-based planner that operates in a radiance field model.

5.1.1.3 Radiance Fields

Neural Radiance Fields (NeRFs) were first introduced to address the problem of novel view synthesis [85]. NeRFs and their variants use neural networks to estimate the radiance emitted by a scene point when viewed from a given direction. Since their introduction, NeRFs have found many use cases in robotics [116] including navigation [1, 54], pose estimation [120, 68], manipulation [49], and SLAM [103, 50]. NeRF-Nav [1], an early effort toward safe planning in radiance fields, approximates the robot as a set of points and then avoids collisions by integrating the NeRF density along the path traced by each point. A later work, CATNIPS [18], instead presents a framework for relating NeRFs to Poisson Point Processes. CATNIPS then introduces a method to convert a NeRF to an occupancy grid to plan robot motions.

While NeRFs use a neural network to model image formation, Gaussian Splatting methods learn a similar representation using Gaussian basis functions [129]. Recently, Kerbl et al. introduced 3D Gaussian Splatting (3DGS) [56], which learns the parameters of un-normalized 3D Gaussians via gradient descent. 3DGS offers high-speed rendering and a training method compatible with modern Graphics Processing Units (GPUs). However, similar to NeRFs, only limited efforts have been made towards using 3DGS for real-time motion planning. Splat-Nav is a real-time robot navigation pipeline designed specifically for Gaussian Splatting

representations. It works by constructing safe polytope corridors through the environment by deriving the distance between a robot described by an ellipsoid while traveling along a straight line and another ellipsoid that corresponds to a Splat within the scene. It then optimizes Bezier curve trajectories within these corridors, ensuring they remain collision-free. This approach is validated on quadrotors.

Unfortunately, extending this approach to robots such as manipulators presents significant challenges. First, motion planning within polytopic constraints is well-understood for quadrotors due to their simpler dynamics and direct workspace constraints, but it becomes considerably more complex for articulated manipulators. Their high-dimensional, nonlinear configuration spaces create intricate, often non-convex, mappings from workspace polytopes to configuration space constraints. Second, Splat-Nav’s distance computation approach is designed specifically for straight-line movements and would not be immediately applicable when performing trajectory optimization over arbitrary trajectories (e.g., non-straight lines in the workspace). Manipulator planning typically requires considering paths through configuration space rather than linear segments in workspace, necessitating a different mathematical formulation for collision avoidance. Finally, the Splat-Nav constraint is implemented using a bisection search, complicating its integration into differentiable trajectory optimizers.

5.1.2 Chapter Contributions

This chapter extends the literature on motion planning in radiance fields by proposing a real-time, receding-horizon trajectory optimization algorithm called SPLANNING and a variant for mobile manipulation called Go-Fetch. This chapter’s key contributions are: 1) A rigorous definition and derivation of rigid body collision within a radiance field model, starting directly from the rendering equation; 2) a computationally efficient approach to upper-bound the probability of collision within a Gaussian Splatting model that can be incorporated into a real-time risk-aware trajectory planner; 3) a re-formulation of Gaussian Splatting that normalizes the 3D Gaussians to ensure the correctness of the collision probabilities; 4) A real-time, risk-aware motion planner that leverages this novel splatting representation to give safety guarantees during planning. Simulation and hardware experiments on a Kinova Gen3 manipulator illustrate that the risk-aware planner solves challenging tasks in real-time; 5) A novel spherical reachable set representation for mobile manipulators that enables safe mobile manipulation in a Gaussian Splat.

5.2 Proposed Method Overview

As illustrated in Fig. 5.2, SPLANNING [82] computes risk-aware trajectories using a visual scene representation. The key insight behind SPLANNING is the combination of reachability analysis with a novel Normalized 3D Gaussian Splat. This combination allows SPLANNING to constrain the probability of collision between the robot’s reachable set and the scene.

5.2.1 Notation

We briefly outline the notation used throughout the rest of the document.

5.2.1.1 Sets and Functions

$n_A \in \mathbb{N}$ represents an arbitrary constant, while $N_A = \{1, \dots, n_A\} \subset \mathbb{N}$ is a set. Capital Roman letters X denote sets, and $\text{co}(X)$ denote the convex hull of X . $f(x)$ denotes a function of x , while $f[g]$ denotes a functional of the function g .

5.2.1.2 Probabilities

A probability space is denoted $(\Omega, \mathcal{F}, \mathcal{P})$ with sample space Ω , event space $\mathcal{F}$, and probability measure $\mathcal{P}$. A random variable is denoted $\mathbf{X} : \Omega \rightarrow S$ for some measurable space S . For a random event E , the probability that E occurs is denoted $\text{P}(E)$.

5.2.1.3 Robot Descriptions

$q(t; k)$ and $\dot{q}(t, k)$ represent a robot’s time-dependent configuration and velocity, respectively. $k \in K \subset \mathbb{R}^{n_k}$ parameterizes a continuum of trajectories, and $t \in T \subset \mathbb{R}$ specifies the time horizon. The forward occupancy of a robot while following a trajectory $q(t; k)$ is denoted $\text{FO}(q(t; k))$.

5.2.2 Overview of Optimization

At runtime, SPLANNING computes probabilistically safe trajectories by solving an optimization problem that limits the probability of the robot’s forward occupancy intersecting the environment. SPLANNING tries to solve the following optimization problem in a

receding-horizon manner:

$$\min_{k \in K} \quad \text{cost}(k) \quad (5.1)$$

$$q(t; k) \in [q_{\text{lim}}^-, q_{\text{lim}}^+] \quad \forall t \in T \quad (5.2)$$

$$\dot{q}(t; k) \in [\dot{q}_{\text{lim}}^-, \dot{q}_{\text{lim}}^+] \quad \forall t \in T \quad (5.3)$$

$$\text{P}(\text{FO}(q(t; k)) \cap \mathcal{E} \neq \emptyset) < \beta \quad \forall t \in T \quad (5.4)$$

The cost function (5.1) is a user-defined objective, such as bringing the robot close to a desired goal. Input constraints enforce limits on joint positions (5.2) and velocities (5.3). Finally, (5.4) is a safety constraint ensuring that the probability of the robot's forward occupancy $\text{FO}(q(t; k))$ intersecting the environment $\mathcal{E}$ stays below the risk threshold β .

The rest of this chapter is organized as follows. Section 5.3 details the representation of the robot and outlines how the forward occupancy of the robot is overapproximated. Section 5.4 provides a detailed description of SPLANNING, including the Normalized 3D Gaussian Splatting used to represent the scene and a method for overapproximating the risk constraint in (5.4). Section 5.5 evaluates SPLANNING both in simulation and on hardware, and Section 5.7 concludes and discusses avenues for future work.

5.3 Robot Representation

This section describes the construction of a sphere-based safety representation for a serial robotic manipulator. Section 5.3.1 summarizes the kinematics of the robotic arm. Section 5.3.2 then introduces the arm occupancy, which is the volume occupied by the arm in the environment. Lastly, Thm. 47 establishes the existence of the robot's Spherical Forward Occupancy, a sphere-based reachable set representation used to construct SPLANNING's novel chance constraint in Section 5.4.3.

5.3.1 Arm Kinematics

Given a compact time interval $T \subset \mathbb{R}$, we define a trajectory for the robot's configuration as $q : T \rightarrow Q \subset \mathbb{R}^{n_q}$ and a trajectory for the velocity as $\dot{q} : T \rightarrow \mathbb{R}^{n_q}$. We restate an assumption [84, Ass. 4] about the robot model:

Assumption 45. *The robot operates in a three-dimensional workspace, denoted $W_s \subset \mathbb{R}^3$, such that $W_s \subset W$ where W denotes the world frame. There exists a reference frame called the base frame, denoted the 0^{th} frame, that indicates the origin of the robot's kinematic chain.*

We assume the robot's base frame coincides with the origin of the world frame. The robot is fully actuated and composed of only revolute joints, where the j^{th} joint actuates the robot's j^{th} link. The robot's j^{th} joint has position and velocity limits given by $q_j(t) \in [q_{\text{lim}}^-, q_{\text{lim}}^+]$ and $\dot{q}_j(t) \in [\dot{q}_{\text{lim}}^-, \dot{q}_{\text{lim}}^+]$ for all $t \in T$, respectively. The robot's input is given by $u : T \rightarrow \mathbb{R}^{n_q}$.

We also assume that the robot's j^{th} reference frame $\{\hat{x}_j, \hat{y}_j, \hat{z}_j\}$ is attached to the robot's j^{th} revolute joint, and that $\hat{z}_j = [0, 0, 1]^\top$ points in direction of the j^{th} joint's axis of rotation. Then $\text{FK}_j : Q \rightarrow \mathbb{R}^{4 \times 4}$ maps the robot's time-dependent configuration to the pose of the j^{th} joint in the world frame such that $p_j(q(t))$ and $R_j(q(t))$ are the position and orientation of frame j with respect to world frame W , respectively.

For simplicity, Assum. 45 only specifies revolute joints to simplify the description of the forward kinematics; the methods described in this chapter readily extend to other joint types. We also ignore any uncertainty in the robot's dynamics, and instead assume that one could apply an inverse dynamics controller [99] to track any trajectory of the robot perfectly. As a result, this work focuses exclusively on modeling the kinematic behavior of the robotic arm.

5.3.2 Arm Occupancy

In this subsection, we define the *forward occupancy* as the volume occupied by the arm in the workspace W_s . Let $L_j \subset W_s \subset \mathbb{R}^3$ denote the volume occupied by the robot's j^{th} link with respect to the j^{th} reference frame. Then the forward occupancy of link j is the map $\text{FO}_j : Q \rightarrow \mathcal{P}(W_s)$ defined as

$$\text{FO}_j(q(t)) = p_j(q(t)) \oplus R_j(q(t))L_j, \quad (5.5)$$

where $p_j(q(t))$ and $R_j(q(t))$ specify the pose of the j^{th} joint, and $R_j(q(t))L_j$ is the rotated volume of link j . The volume occupied by the entire arm in the workspace is then defined by the map $\text{FO} : Q \rightarrow W_s$ such that

$$\text{FO}(q(t)) = \bigcup_{j=1}^{n_q} \text{FO}_j(q(t)) \subset W_s. \quad (5.6)$$

Because the geometry of any of the robot's links may be arbitrarily complex, we restate an assumption [84, Ass. 5] that simplifies the construction of an overapproximation to the forward occupancy:

Assumption 46. *Given a robot configuration $q(t)$ and any $j \in \{1, \dots, n_q\}$, there exists a ball with center $p_j(q(t))$ and radius r_j that overapproximates the volume occupied by the j^{th}*

joint in W_s . We further assume that link volume L_j is a subset of the tapered capsule formed by the convex hull of the balls overapproximating the j^{th} and $(j+1)^{\text{th}}$ joints.

Following Assum. 46, we now define the ball $S_j(q(t))$ overapproximating the volume occupied by the j^{th} joint as

$$S_j(q(t)) = B_2(p_j(q(t)), r_j) \quad (5.7)$$

and the tapered capsule $TC_j(q(t))$ overapproximating the j^{th} link as

$$TC_j(q(t)) = \text{co}\left(S_j(q(t)) \cup S_{j+1}(q(t))\right). \quad (5.8)$$

Then, the volumes occupied by the j^{th} link (5.5) and the entire arm (5.6) is overapproximated by

$$\text{FO}_j(q(t)) \subset TC_j(q(t)) \quad (5.9)$$

and

$$\text{FO}(q(t)) \subset \bigcup_{j=1}^{n_q} TC_j(q(t)) \subset W_s, \quad (5.10)$$

respectively.

For convenience, the notation $\text{FO}(q(T))$ denotes the forward occupancy over an entire time interval T . $\text{FO}(q(T))$ is also called the *reachable set* of the robot. Notably, the arm is *collision-free* over the time interval T if $\text{FO}(q(T))$ does not intersect with the environment.

To facilitate the exposition of our approach, we summarize the construction of the robot's safety representation in the following theorem:

Theorem 47. *Given a serial manipulator with $n_q \in \mathbb{N}$ revolute joints and a time partition T of a finite set of intervals T_i (i.e., $T = \cup_{i=1}^{n_t} T_i$), the swept volume corresponding to the robot's motion over T is overapproximated by a collection of L_2 balls in $\mathbb{R}^3$, which we call the Spherical Forward Occupancy ($\mathcal{SFO}$) [84] defined as*

$$\mathcal{SFO} = \bigcup_{j=1}^{n_q} \bigcup_{i=1}^{n_t} \bigcup_{m=1}^{n_s} S_{j,i,m}(q(T_i; k)), \quad (5.11)$$

where each $S_{j,i,m}(q(T_i; k))$ is an L_2 ball in $\mathbb{R}^3$, $n_s \in \mathbb{N}$ is a parameter that specifies the number of closed balls overapproximating each of the robot's links, and k is a trajectory parameter that characterizes the motion of the robot over T .

Note that one can explicitly construct an $\mathcal{SFO}$ that satisfies this assumption using the approach described in [84], which we summarize for convenience in Appendix D.2.

5.4 Detailed Description of SPLANNING

This section provides a detailed overview of SPLANNING, a novel approach for generating risk-aware motion plans in cluttered scenes represented as radiance fields using Gaussian basis functions. Section 5.4.1 provides a brief overview of radiance fields and Gaussian Splatting. Section 5.4.2 then describes how to bound the probability of collision between a ball in $\mathbb{R}^3$ and a learned radiance field represented by Gaussian Splats. Section 5.4.3 presents an easily-computed expression for an upper-bound on the aforementioned probability of collision, and Section 5.4.4 discusses how to leverage this result as a computationally-tractable chance constraint for online trajectory optimization.

5.4.1 3D Scene Representation

This section introduces the Normalized 3D Gaussian Splatting model used to represent the environment.

5.4.1.1 Volume Rendering

A radiance field is a 5-dimensional function $\mathcal{L} : (x, d) \mapsto (r, g, b, \sigma)$ that maps a point $x \in \mathbb{R}^3$ and viewing direction $d \in \mathbb{S}^2$ to r, g, b colors and a volume density σ [105]. This work neglects color as it does not impact the collision probability. Further, the density σ does not depend on the view direction. As a result, we simplify the description of the radiance model by estimating the density function $\sigma : \mathbb{R}^3 \rightarrow \mathbb{R}$.

We define a ray $\varphi : \mathbb{R} \rightarrow \mathbb{R}^3$ by $\varphi(t) = o_\varphi + tv_\varphi$, where $o_\varphi \in \mathbb{R}^3$ is the ray origin and $v_\varphi \in \mathbb{S}^2$ is the unit direction vector. From the density function σ , we may compute the probability that a particle travels along $\varphi(t)$ from $t = a$ to $t = b$ without collision using the transmittance function $\mathcal{T}_a^b$, as derived in [105]:

$$\mathcal{T}_a^b[\varphi] = \exp \left(- \int_a^b \sigma(\varphi(t)) dt \right). \quad (5.12)$$

Equivalently, we may define $C_a^b[\varphi]$ as the random event describing a particle colliding while traveling along φ from a to b , where $P(C_a^b[\varphi]) = 1 - \mathcal{T}_a^b[\varphi]$ denotes the probability that a collision occurs.

5.4.1.2 Normalized Gaussian Splatting

Traditional splatting approaches represent the density function σ using basis functions such as 3D Gaussian functions [129]. This allows the integral in (5.12) to be computed by trans-

forming each Gaussian to each ray’s coordinate system and then analytically marginalizing the depth dimension. The result is a set of 2D Gaussian functions on the image plane that are queried and blended to form the image. 3DGS [56] proposed a variation of the traditional splatting formulation that uses un-normalized 3D Gaussian functions to represent a scene. A consequence of using un-normalized Gaussians is that the 3D to 2D transformation performed during rasterization *projects* the 3D Gaussians onto the image plane rather than integrating the 3D Gaussians. Because the 2D Gaussians are not constructed from integrating 3D Gaussians along rays, the 3D Gaussians cannot be interpreted as a basis set for σ .

In contrast, the method we introduce for collision evaluation (Section 5.4.2) relies on using the 3D Gaussians as a basis for σ . Hence, one of our key contributions is to re-formulate 3DGS using $n_G \in \mathbb{N}$ normalized 3D Gaussians such that σ can be used to compute the probability of collision of a particle traveling along φ . In particular,

$$\begin{aligned} P(C_a^b[\varphi]) &= 1 - \mathcal{T}_a^b[\varphi] \\ &= 1 - \exp\left(-\int_a^b \sigma(\varphi(t))dt\right) \\ &= 1 - \exp\left[\sum_{n=1}^{n_G} \frac{w_n}{\sqrt{(2\pi)^3 |\Sigma_n|}} \int_a^b \exp\left(-\frac{1}{2} \|\varphi(t) - \mu_n\|_{\Sigma_n^{-1}}^2\right) dt\right], \end{aligned} \quad (5.13)$$

where $w_n \in \mathbb{R}^+$ is a weight parameter and $G_n : \mathbb{R}^3 \rightarrow \mathbb{R}$ gives the normalized Gaussian density with mean $\mu_n \in \mathbb{R}^3$ and covariance matrix $\Sigma_n \in \mathbb{R}^{3 \times 3}$. That is,

$$G_n(x) = \frac{1}{\sqrt{(2\pi)^3 |\Sigma_n|}} \exp\left(-\frac{1}{2} (x - \mu_n)^T \Sigma_n^{-1} (x - \mu_n)\right). \quad (5.14)$$

Additionally, existing 3D Gaussian Splatting implementations apply a low-pass filter to the 2D Gaussian functions on the image plane by convolving the projected 2D Gaussians with an isotropic 2D Gaussian with a covariance of 0.3 pixels. This low-pass filter reduces artifacts in the rendered images. We omit this step from the normalized 3DGS rendering procedure because it has an effect that cannot be reasoned about in 3D. This would undermine the validity of our collision avoidance constraint.

Instead, we apply an analogous filter in 3D Gaussians by convolving the 3D Gaussians in world coordinates with an isotropic 3D Gaussian with a small covariance of $1e-6$. Because this operation is applied to the 3D Gaussians, it does not impact the ability of the collision

constraint to use the 3D Gaussians to model σ . The implementation details describing the updated Gaussian Splat training process are provided in Appendix D.5.

5.4.2 Bounding the Probability of Collision

SPLANNING enforces safety by ensuring the probability of collision between the robot and the scene is below a given risk threshold. This subsection derives a method for bounding the probability of collision between a ball in $\mathbb{R}^3$ and a scene represented as a radiance field. We make the following assumption about the Normalized 3D Gaussian Splat’s representation of the scene:

Assumption 48. *The normalized 3D Gaussian Splat is assumed to have converged to an accurate representation of the scene, such that for all possible rays in the scene, the transmittance defined in (5.12) is accurately computed. Furthermore, we assume that the transmittance, which represents the probability of a particle colliding while traveling along a ray, is equivalent to the probability that an infinitesimal segment of a rigid body experiences a collision when traveling along the same ray.*

Without loss of generality, suppose $S = B_2(0, \rho)$ is a closed L_2 ball centered at the origin with radius ρ and whose boundary is denoted ∂S . This assumption is made without loss of generality because one can apply a frame transformation to any arbitrary ball in $\mathbb{R}^3$ to shift its center to the origin. Let $C(S)$ denote the random event that the ball S collides with the environment. Then, we seek to compute the probability that $C(S)$ occurs, which we denote $P(C(S))$. We model this as the probability that a ray randomly cast from the center of S experiences a collision before reaching ∂S .

Formally, let $(\Omega_{\mathbf{V}}, \mathcal{F}_{\mathbf{V}}, \mathcal{P}_{\mathbf{V}})$ be a probability space, and let $\mathbf{V} : \Omega_{\mathbf{V}} \rightarrow \mathbb{S}^2$ be a random variable representing the direction of a randomly-cast ray originating at the center of S . In particular, let $\mathbf{V}$ be uniformly distributed on $\mathbb{S}^2$ under $\mathcal{P}_{\mathbf{V}}$. Define the random ray Φ as:

$$\Phi(t, \omega_{\mathbf{V}}) = t\mathbf{V}(\omega_{\mathbf{V}}), \quad \omega_{\mathbf{V}} \in \Omega_{\mathbf{V}}. \quad (5.15)$$

The transmittance of $\Phi(t, \omega_{\mathbf{V}})$ from $t = 0$ to $t = \rho$ is given by

$$\mathcal{T}_0^\rho(\omega_{\mathbf{V}}) = \exp\left(-\int_0^\rho \sigma(\Phi(t, \omega_{\mathbf{V}}))dt\right), \quad \omega_{\mathbf{V}} \in \Omega_{\mathbf{V}}. \quad (5.16)$$

Note that $\mathcal{T}_0^\rho(\omega_{\mathbf{V}})$ depends on $\Phi(t, \omega_{\mathbf{V}})$, which in turn depends on $\mathbf{V}(\omega_{\mathbf{V}})$. Hence, $\mathcal{T}_0^\rho : \Omega_{\mathbf{V}} \rightarrow \mathbb{R}$ is also a random variable on $(\Omega_{\mathbf{V}}, \mathcal{F}_{\mathbf{V}}, \mathcal{P}_{\mathbf{V}})$. The probability the random ray collides on the interval from $t = 0$ to $t = \rho$ is computed by $1 - \mathcal{T}_0^\rho(\omega_{\mathbf{V}})$. Finally, the probability that

the collision risk for the randomly-cast ray exceeds a risk threshold $\alpha \in \mathbb{R}^+$ is given by

$$\mathbb{P}(C(S)) = \mathbb{P}(1 - \mathcal{T}_0^p(\omega_{\mathbf{V}}) \geq \alpha), \quad \omega_{\mathbf{V}} \in \Omega_{\mathbf{V}}. \quad (5.17)$$

The following provides an upper bound on $\mathbb{P}(C(S))$:

Theorem 49. *Consider, without loss of generality, a ball $S = B_2(0, \rho)$ of radius ρ that is centered at the origin. Let $\alpha \in \mathbb{R}^+$ denote the risk threshold defined in (5.17) for the probability of collision between a ray and the environment. Then, the probability that the ball S collides with the environment is bounded above by*

$$\mathbb{P}(C(S)) \leq \frac{1}{\alpha} \left[1 - \exp \left(-\frac{1}{4\pi} \int_S \sigma(x) dx \right) \right] \quad (5.18)$$

where the integral denotes a volume integral over the sphere $\mathcal{S}$.

A proof is provided in Appendix D.3.1.

5.4.3 Evaluating the Collision Bound

Computing the exact integral in (5.18) is non-trivial. While [53, Thm 3.3] provides a method for computing the integral of Gaussian functions over spheres, it depends on an infinite series that is difficult to evaluate in practice. Therefore, we derive a computationally attractive expression for computing an upper bound for the probability of collision in Theorem 49.

Theorem 50. *Let $S = B_2(0, \rho)$ be a closed ball of radius ρ that is centered at the origin. Suppose the density function $\sigma : \mathbb{R}^3 \rightarrow \mathbb{R}$ is represented by a set of $n_G \in \mathbb{N}$ normalized Gaussian basis functions $\{G_n\}_{n=1}^{n_G}$ with means $\{\mu_n\}_{n=1}^{n_G}$ and covariance matrices $\{\Sigma_n\}_{n=1}^{n_G}$, each with Eigendecomposition given by $\Sigma_n = R_n \Lambda_n R_n^T$. Finally, let the diagonal elements of Λ_n be denoted as $\lambda_{n,1}, \lambda_{n,2}, \lambda_{n,3}$ and represent the eigenvalues of Σ_n . Then*

$$\int_S \sigma(x) dx \leq \mathcal{H}(S) \quad (5.19)$$

where

$$\begin{aligned} \mathcal{H}(S) &= \sum_{n=1}^{n_G} \eta'_n w_n \cdot \\ &\prod_{\ell=1}^3 \left[\sqrt{\frac{\pi \lambda'_{n,\ell}}{2}} \left(\operatorname{erf} \left(\frac{\rho - \mu'_{n,\ell}}{\sqrt{2\lambda'_{n,\ell}}} \right) - \operatorname{erf} \left(\frac{\rho + \mu'_{n,\ell}}{\sqrt{2\lambda'_{n,\ell}}} \right) \right) \right]. \end{aligned} \quad (5.20)$$

Above, erf denotes the error function and μ'_n , λ'_n , and η'_n correspond to the mean, eigenvalues, and normalization constant of the normalized Gaussian G'_n obtained by rotating G_n by R_n^T . That is, G'_n has mean $\mu'_n = R_n^T \mu_n$ and covariance $\Sigma'_n = R_n^T \Sigma_n R_n$. Furthermore,

$$P(C(S)) \leq \frac{1}{\alpha} \left[1 - \exp \left(-\frac{1}{4\pi} \int_S \sigma(x) dx \right) \right] \quad (5.21)$$

$$\leq \frac{1}{\alpha} \left[1 - \exp \left(-\frac{1}{4\pi} \mathcal{H}(S) \right) \right]. \quad (5.22)$$

Thus, we may constrain the risk of collision by enforcing

$$\left[1 - \exp \left(-\frac{1}{4\pi} \mathcal{H}(S) \right) \right] < \alpha \cdot \beta \quad (5.23)$$

for a given risk threshold $\beta \in (0, 1]$.

A proof is provided in Appendix D.3.2. Note that, in practice, we set $\alpha = \beta$ for simplicity due to their mutual dependence introduced in (5.23).

5.4.4 Risk-Aware Motion Planning

The purpose of SPLANNING is to generate risk-aware motion plans in cluttered environments in a receding-horizon fashion. Prior to planning, a normalized Gaussian Splatting representation of the scene is constructed. At every planning iteration, the robot is given $t_{\text{plan}} \leq 0.5$ seconds to find a feasible trajectory by solving

$$\min_{k \in K} \quad \text{cost}(k) \quad (\text{Splanning-Opt}) \quad (5.24)$$

$$\text{s.t.} \quad q_j(T_i; k) \subseteq [q_{j,\text{lim}}^-, q_{j,\text{lim}}^+] \quad \forall (i, j) \in N_t \times N_q \quad (5.25)$$

$$\dot{q}_j(T_i; k) \subseteq [\dot{q}_{j,\text{lim}}^-, \dot{q}_{j,\text{lim}}^+] \quad \forall (i, j) \in N_t \times N_q \quad (5.26)$$

$$\sum_{\substack{(j,m) \in \\ N_q \times N_s}} 1 - \exp \left(-\frac{\mathcal{H}(S_{j,i,m}(q_j(T_i; k)))}{4\pi} \right) < \alpha \cdot \beta \quad \forall i \in N_t \quad (5.27)$$

where $k \in K$ is the trajectory parameter (Appendix. D.2.1) and $\text{cost}(k)$ is a task-specific cost function. The robot's reachable set $\{S_{j,i,m}\}_{m=1}^{n_s}$ is a function of the robot's position (and hence trajectory parameter) and is computed repeatedly while numerically solving (Splanning-Opt). Safety is enforced using the novel collision-avoidance constraint (5.27). If a solution is not found, the robot executes a braking maneuver using the trajectory parameter found in the previous planning iteration. Since SPLANNING's collision-avoidance

Table 5.1: Reconstruction evaluation on Replica and TUM-RGBD. Standard 3DGS slightly outperforms the normalized variant. However, the performance is comparable, and only Normalized 3DGS is suitable for risk-aware motion planning.

Dataset	Scene	Unnormalized				Normalized			
		PSNR	SSIM	Train Time	Render FPS	PSNR	SSIM	Train Time	Render FPS
Replica	office0	44.052	0.986	1253.0	205.4	42.103	0.979	1144.1	174.0
	office1	42.510	0.974	1008.9	218.1	40.457	0.964	941.1	180.0
	office2	37.663	0.972	1647.6	174.5	36.055	0.963	1718.0	147.0
	office3	37.533	0.969	1545.5	186.6	36.617	0.961	1636.7	145.4
	office4	39.188	0.971	1488.4	194.2	38.504	0.968	1610.5	151.0
	room0	37.884	0.971	1923.2	147.1	36.301	0.961	2190.4	108.0
	room1	39.347	0.973	1695.3	165.8	37.513	0.967	1826.9	126.3
	room2	40.280	0.975	1669.8	168.0	38.995	0.970	1635.0	138.1
	Average	39.807	0.974	1529.0	182.5	38.318	0.967	1587.8	146.2
TUM-RGBD	fr1/floor	23.989	0.703	1206.2	190.1	23.626	0.697	1332.3	176.0
	fr1/plant	19.688	0.671	1203.0	194.3	19.189	0.661	1493.7	154.4
	fr1/desk	21.313	0.758	1069.2	259.3	21.018	0.751	1075.9	240.5
	fr1/desk2	20.727	0.747	1146.0	244.1	20.435	0.743	1185.0	225.6
	fr2/360_hemi	20.190	0.683	2479.5	99.9	19.008	0.664	3028.7	75.2
	fr2/dishes	23.680	0.819	1347.8	191.8	23.256	0.811	1574.3	148.9
	fr2/flower	22.179	0.763	1465.5	173.5	21.372	0.747	1796.5	127.2
	fr2/coke	21.291	0.773	1474.1	167.5	20.061	0.755	1741.7	126.5
	fr1/room	19.488	0.690	2368.6	101.3	19.216	0.691	2595.1	92.2
	fr1/teddy	19.403	0.652	1377.6	184.5	18.855	0.648	1449.2	169.4
	fr1/360	20.155	0.717	1462.3	130.6	18.585	0.668	5626.5	33.0
	Average	21.100	0.725	1509.1	176.1	20.420	0.712	2081.7	142.6

constraints are differentiable, analytical constraint gradients are provided to ensure real-time motion planning. For simplicity, expressions for kinematics and dynamics constraints were not included in (Splanning-Opt). However, [81] and [84] provide detailed explanations for including such constraints.

5.5 Experimental Results

This section assesses the effectiveness of SPLANNING by evaluating Normalized 3DGS reconstructions, the proposed constraint representation, and the proposed trajectory optimizer against baselines.

5.5.1 Normalized 3DGS Evaluation

We first evaluate the ability of the normalized 3DGS to reconstruct scenes compared to the original 3DGS formulation. We evaluate reconstruction quality on two common RGB-D datasets. First, the method was evaluated on Replica [100], which is a simulated dataset. Replica contains sequences of simulated images at 20Hz; these were decimated to 2Hz. Then, 1 in every 8 retained images was excluded from training for evaluation, consistent with the methodology from [56, Sec. 7.2].

Table 5.2: The RMSE depth error (m) is computed per evaluation image, then averaged over each image in the dataset. The standard 3DGS slightly outperforms the Normalized variant, but only Normalized 3DGS is suitable for risk-aware planning.

		Normalized	Unnormalized
Replica	office0	0.017	0.034
	office1	0.013	0.035
	office2	0.017	0.029
	office3	0.022	0.036
	office4	0.026	0.039
	room0	0.023	0.046
	room1	0.015	0.033
	room2	0.024	0.034
	Average	0.020	0.036
TUM-RGBD	fr1/floor	0.040	0.044
	fr1/plant	0.241	0.257
	fr1/desk	0.087	0.092
	fr1/desk2	0.116	0.122
	fr2/360_hemi	0.278	0.302
	fr2/dishes	0.135	0.154
	fr2/flower	0.311	0.338
	fr2/coke	0.286	0.307
	fr1/room	0.149	0.162
	fr1/teddy	0.239	0.241
	fr1/360	0.155	0.206
	Average	0.185	0.202

Next, Normalized 3DGS was evaluated on TUM-RGBD [101], which is a real-world dataset. TUM-RGBD is a relatively difficult dataset for accurate 3D reconstruction due to low-resolution images and noisy depth data. TUM-RGBD sequences were downsampled to achieve an approximate total of 250 images per sequence. Again, 1 in every 8 images was excluded from training for evaluation. The reported metrics were only computed on the images excluded from training.

Visual reconstruction results, measured by Structural Similarity Index Measure (SSIM) and Peak Signal-to-Noise Ratio (PSNR) [48], are shown in Table 5.1. SSIM and PSNR are widely used metrics for evaluating image reconstruction quality. PSNR measures the overall error between a reconstructed image and its reference by comparing the peak signal level to the noise level; it is expressed in decibels, and higher values indicate better quality. In contrast, SSIM assesses perceptual similarity by comparing local patterns of pixel intensities that have been normalized for luminance, contrast, and structure. SSIM values range from 0 to 1, with values closer to 1 indicating images that are more similar in structure.

Experiments indicate that Normalized 3DGS approaches the visual reconstruction quality of standard 3DGS, and with similar training and rendering speeds. This indicates that the Normalized 3DGS representation maintains the ability of 3DGS to support rapid and high-quality view synthesis.

Additionally, the geometric accuracy of each method is evaluated by measuring the difference between the rendered depth and ground-truth depth. For each image excluded from training, a depth image is rendered and compared to the ground truth. The Root-Mean-Square (RMS) error is calculated for each image and then averaged over all images to find the final metric. Similar to SSIM and PSNR results, Normalized 3DGS approaches the accuracy of standard 3DGS, as shown in Table 5.2.

5.5.2 Simulation Environment

The simulation environment, implemented with PyRender³, contains cubical obstacles with 20cm sides. The obstacles are randomly placed within the reachable space of the robot arm. Three sets of scenes are created with 10, 20, and 40 obstacles per scene, respectively. For each number of obstacles, 100 random configurations are generated resulting in a total of 300 random scenes with varying degrees of clutter. A simulated Kinova Gen3 7-DOF serial manipulator is used in the planning experiments. The simulation environment and a corresponding 3D Gaussian Splat model are shown in Figure 5.3.

³<https://github.com/mmatl/pyrender>

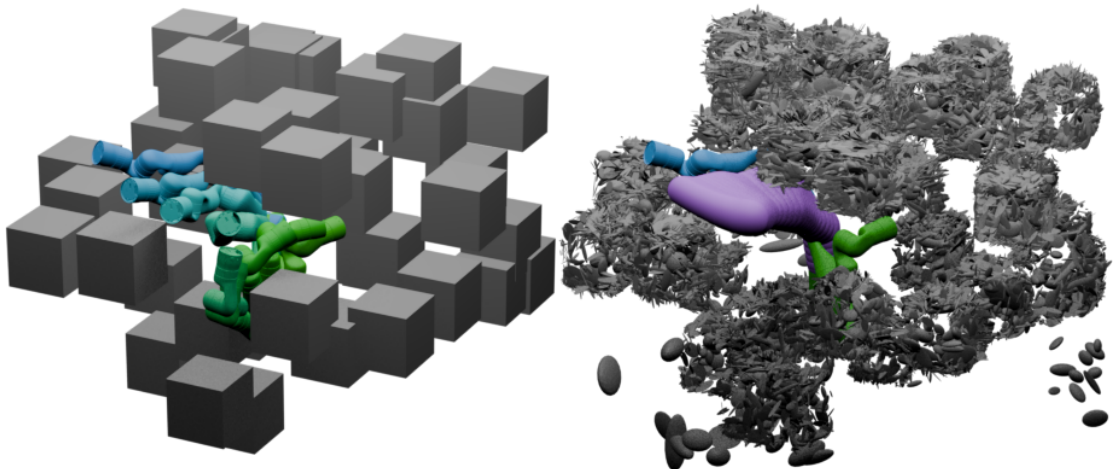


Figure 5.3: SPLANNING generates safe trajectories in densely cluttered environments in simulation. The left panel shows discrete time steps of sequential trajectories that brings the arm from the start configuration (blue) safely to the goal configuration (green). The right panel shows an intermediate planning step where the Spherical Forward Occupancy (purple) avoids the obstacles represented by a Normalized 3D Gaussian Splat.

5.5.3 Collision Probability Evaluation

We first evaluate the performance of our constraint representation and compare it against existing methods that plan in radiance fields. In particular, we compare against CATNIPS [18], which represents the world using a Neural Radiance Field, and Splat-Nav [19], which represents the world using an un-normalized 3D Gaussian Splat. Each method’s collision representation is shown in Fig. 5.4. The NeRF representation for CATNIPS was trained using Nerfstudio [106], while the Splat-Nav scene representations were built using the original Gaussian Splatting method [56]. We treat each collision detection method as a classifier to determine whether a configuration will result in a collision. The performance of each method is measured by evaluating the precision and recall of the collision classifier [90]. A True Positive indicates the correct identification of a collision, while a True Negative indicates the correct identification of free space.

We select 15 scenes, 5 for each of 10, 20, and 40 randomly-placed obstacles to sample within. In each scene, the arm configurations are classified into three categories: Unsafe (in collision), Nearly Safe ($< 40\text{cm}$ to the nearest obstacle), and Safe ($> 40\text{cm}$ to the nearest obstacle). We randomly sample 30 configurations of the arm, which includes 10 configurations from each category to ensure an informative collection of samples. We conduct our experiments under two conditions. First, we compare estimated collisions to ground truth collisions over the individual spheres that comprise the robot’s $\mathcal{SFO}$ for stationary configurations. Second, we compare each method’s estimate of whether an entire configuration is

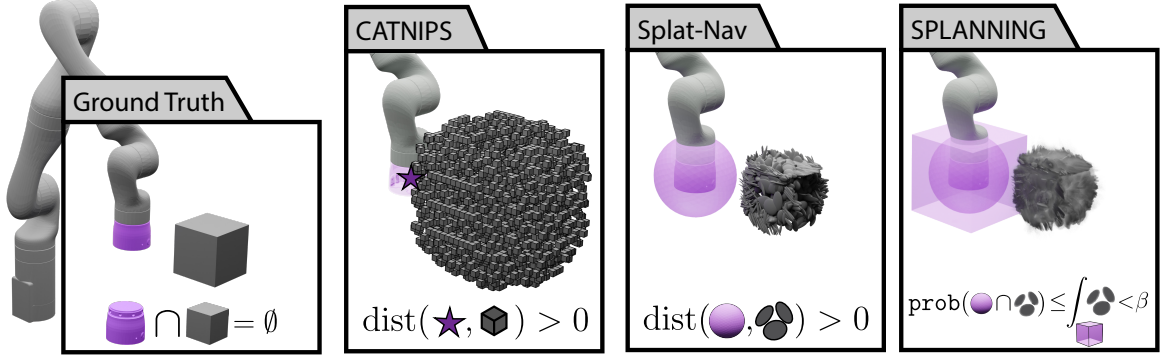


Figure 5.4: The collision constraint representation of CATNIPS, Splat-Nav, and SPLANNING are compared. CATNIPS transforms a NeRF into a 3D occupancy grid by relating it to a Poisson Point Process; after convolving this grid with a robot kernel, the center of the robot is checked for collision with the grid. Splat-Nav deterministically evaluates whether the confidence ellipsoids of each Gaussian intersect with a spherical robot. Finally, SPLANNING integrates a Normalized 3D Gaussian Splat over an overapproximation of the robot arm to form a risk constraint.

in collision against the ground truth. When evaluating collisions, we use the radius provided by the $\mathcal{SFO}$ for Splat-Nav and SPLANNING, and as CATNIPS utilizes a fixed-size robot kernel, we fix each sphere to have the radius of the largest sphere in the $\mathcal{SFO}$. Descriptions of how these baselines are implemented are detailed in Appendix D.4.

We calculate the Precision-Recall values at various thresholds corresponding to the allowable collision risk. SPLANNING, CATNIPS, and Splat-Nav each express this maximum allowable risk differently. For SPLANNING, we vary the parameters α and β , in the range of $(0,1]$, as discussed in Section 5.4.2. For CATNIPS, we vary the collision probability σ in the range of $(0,1]$ and leave all other parameters as their respective defaults from [18]. Finally, Splat-Nav deterministically checks whether the robot collides with the 1-sigma ellipsoids, meaning the method does not emit a risk parameter. Instead, we evaluate the method across several level sets of the Gaussian functions, ranging from the 10^{-5} -sigma ellipsoid to the 10-sigma ellipsoid.

We report Precision-Recall curves in Fig. 5.5. All three methods achieve high recall, indicating that all were able to classify unsafe configurations correctly. CATNIPS has the lowest precision, indicating the presence of false positives which would cause a motion planner to behave overly conservatively, and has very limited control over the behavior of their constraint when σ is varied. Splat-Nav achieves higher precision and recall, with some control over the constraint behavior. The SPLANNING constraint achieves the highest precision and recall. Further, Fig. 5.5 shows that the thresholds α and β in SPLANNING allow for significant control of the behavior of the constraint, ranging from highly aggressive near the

left to more conservative on the right.

5.5.4 Planner Performance Evaluation

In this section, we quantitatively evaluate the performance of SPLANNING on the Kinova 7-DOF arm within a PyRender simulation environment. In each of the 300 randomly generated scenes, a start and end pose are sampled from the free space of the scene; note that it is not guaranteed that a collision-free path exists between the start and the goal.

Table 5.3: Number of successes for PHLAME, SPARROWS, ARMTD, MPOT, CHOMP, and TrajOpt in the Kinova planning experiment with 10, 20, and 40 randomly-placed obstacles. For each set of experiments, 100 randomly-generated scenes were tested. The first number reported is the number of successes (higher is better). The second number, **orange**, indicates trials that failed to reach the goal state but did not result in crashes. **Red** indicates the number of failures due to collision (lower is better). The **most** and second most successes are annotated.

Methods	Scene Representation	# Successes		
		10 Obstacles	20 Obstacles	40 Obstacles
PHLAME, Risk = 0.01	Normalized 3DGS	29, 71 , 0	6, 94 , 0	0, 100 , 0
PHLAME, Risk = 0.025	Normalized 3DGS	60, 39 , 1	47, 52 , 1	25, 75 , 0
PHLAME, Risk = 0.0375	Normalized 3DGS	63, 37 , 0	50, 46 , 4	26, 71 , 3
PHLAME, Risk = 0.05	Normalized 3DGS	<u>69</u> , 30 , 1	<u>52</u> , 43 , 5	<u>28</u> , 61 , 11
PHLAME, Risk = 0.1	Normalized 3DGS	67, 25 , 8	35, 29 , 36	25, 37 , 38
SPARROWS	Ground-Truth	85 , 15 , 0	69 , 31 , 0	41 , 59 , 0
ARMTD	Ground-Truth	56, 44 , 0	17, 83 , 0	0, 100 , 0
CHOMP [128]	Ground-Truth	30, 70 , 0	9, 91 , 0	4, 96 , 0
TrajOpt [94]	Ground-Truth	33, 0 , 67	9, 0 , 91	6, 0 , 94
MPOT [66]	Ground-Truth	58, 0 , 42	23, 0 , 77	9, 0 , 91
cuRobo [104]	Ground-Truth	59, 0 , 41	45, 0 , 55	22, 0 , 78

We compare SPLANNING to state-of-the-art trajectory optimization methods, including SPARROWS [84], ARMTD [45], CHOMP [128], TrajOpt [94], MPOT [66], and cuRobo [104].

Table 5.4: The peak memory usage of SPLANNING was measured for each execution, then averaged across the 100 randomly-generated scenes for each number of obstacles.

Numb Obs.	Mean Peak Memory ± Std. Dev. (GB)
10	12.36 ± 0.98
20	17.55 ± 1.05
40	20.86 ± 0.98

Each baseline method uses privileged geometry information from the simulation to inform collision avoidance (i.e., the exact 3D geometry of the environment is known), while SPLANNING uses simulated perception to reconstruct a Normalized 3DGS for planning. To evaluate whether each method produces collision-free trajectories, the simulator checks whether the robot is in collision with the ground-truth scene. In these experiments, SPLANNING, SPARROWS, and ARMTD are limited to 0.5s to construct the plan, and each trial is run for a maximum of 150 planning horizons. A success indicates that the robot successfully reached the goal. Collisions are detected by checking for collisions in simulation between the robot mesh and the obstacle meshes. Failures are also reported if the robot fails to reach the goal in the 150 planning horizons or if the optimizer fails to find a feasible plan for two consecutive planning iterations.

For the following evaluations, the experiments involving SPLANNING, SPARROWS, and ARMTD were conducted on a system with an AMD Ryzen 5950X @ 3.4GHz and dual NVIDIA RTX A6000 GPUs. The experiments involving CUROBO and MPOT were conducted on a system equipped with an Intel Core i7-8700K CPU @ 3.70GHz and dual NVIDIA RTX A6000 GPUs. Baseline experiments for CHOMP and TrajOpt were performed on a machine with an Intel Core i9-12900H CPU @ 4.90GHz. During all simulation experiments, only one of the two GPUs was used for trajectory optimization. In the hardware experiments described below, one GPU ran the optimizer while the other computed high-level plans in parallel.

Table 5.3 presents the number of successes and collisions achieved by each planner in the experiments. SPARROWS, which has access to the ground-truth scene, had the highest number of successes and zero collisions. This is the expected result since SPARROWS solves a similar optimization problem to SPLANNING but is given ground-truth scene knowledge. In contrast, SPLANNING is the only method to incorporate perception. When SPLANNING’s α and β parameters are set between 0.025 and 0.05, SPLANNING achieves more successes than all other baselines. Closest behind SPLANNING are cuRobo and MPOT, but both come with a significantly higher number of collisions.

The time to formulate and solve each optimization problem was measured. For SPLANNING, the timings were averaged over all α, β parameters. Further, SPLANNING requires a warm-up for compiled PyTorch functions; as a result, the first planning cycle is excluded from the timing results.

Timing results are summarized in Table 5.5. All methods remain under the 0.5s limit. Notably, SPLANNING optimizes faster than ARMTD for the 20 and 40 obstacle cases, despite ARMTD having access to ground-truth obstacles and SPLANNING incorporating perception.

Table 5.5: Mean per-step planning time for SPLANNING, SPARROWS, and ARMTD in Kinova planning experiments with 10, 20, and 40 obstacles with a 0.5s time limit. For SPLANNING, the first planning horizon is excluded from each timing measurement as the system warms up.

Methods	Mean Planning Time [s]		
	10 Obstacles	20 Obstacles	40 Obstacles
SPLANNING	0.202	0.277	0.319
SPARROWS	0.112	0.128	0.153
ARMTD	0.106	0.213	0.332

Table 5.6: Mean runtime for constraint and constraint gradient evaluation for SPLANNING, SPARROWS, and ARMTD in Kinova planning experiments with 10, 20, and 40 obstacles. For SPLANNING, the first planning constraint evaluation is excluded from each timing measurement as the system warms up.

Methods	Mean Constraint Evaluation Time [ms]		
	10 Obstacles	20 Obstacles	40 Obstacles
SPLANNING	6.56	7.68	8.06
SPARROWS	2.27	2.59	3.25
ARMTD	2.67	3.30	3.92

The time to compute each constraint and its gradient is evaluated in Table 5.6. The mean constraint evaluation for SPLANNING is under 10ms in all scenarios. SPLANNING takes slightly longer for each constraint evaluation than SPARROWS and ARMTD, both of which use ground-truth representations of obstacles rather than the Normalized 3DGS used by SPLANNING. However, as shown in Table 5.5, SPLANNING comfortably remains under the 0.5s planning time limit.

5.5.5 Real-World Demonstrations

The planner was evaluated on a Kinova Gen3 7-DOF manipulator in a real-world setting. For the hardware implementation, a high-level planner was added to the system. In particular, we deploy an open-source implementation [102] of bidirectional RRT [62]. The RRT planner evaluated collisions at discrete poses by computing the spherical forward occupancy of the static arm, and evaluating the SPLANNING collision constraint. Additionally, to deal with the non-uniform lighting conditions arising from real-world sensor data, we augment the Normalized 3D Gaussian Splatting method to include the appearance embedding method proposed by [64].

The hardware evaluations run the motion planner in real-time. To enable this, four pro-

Table 5.7: Hardware experiments commanded a robot to sets of three high-level waypoints then back to the start. Each trial listed corresponds to one such set of motions. A trial is counted a success only if all four motions succeed; otherwise, the first failure is reported. Asterisks* indicate the number of trials where the robot gently grazed an obstacle, but motion was not impeded. RRT and Optimization Time (s) were averaged across all executions for each configuration.

Risk Level (RRT, Opt.)	Success	RRT Fail	Stuck	Crash	RRT Time	Opt. Time
0.01,0.05	2**	15	0	1	14.59	0.319
0.025, 0.025	15**	3	0	0	16.53	0.329
0.05, 0.01	14	1	2	1	13.17	0.338
0.1, 0.01	9*	0	8	1	12.12	0.351

cesses are run on a desktop computer with an AMD Ryzen 5950X and two NVIDIA A6000 GPUs. First, the high-level planner takes the start and goal configuration and computes a sequence of joint waypoints that are each collision-free, as measured by the SPLANNING constraint. Second, a SPLANNING optimizer iteratively solves the optimization problem defined in Section 5.4.4 to compute a control parameter k . Third, a low-level controller consumes the control parameter k and uses a combination of RNEA-based inverse dynamics [75] and a PD error tracking term. Fourth, an orchestration process manages the parallel execution of the above three components. By assuming the robot tracks the target trajectories perfectly, the SPLANNING process may compute the next plan as the current plan is being executed. Similarly, the high-level planner computes the next high-level plan as the current sequence of waypoints is being tracked.

The robot was mounted in an indoor setting with two sets of shelves, as shown in Fig. 5.2. For each set of shelves, three target configurations were placed in the environment. For each set of shelves, the robot was commanded to initialize at a safe configuration, cycle between the three target configurations, and then return to the home position. Retaining cycle order, each target configuration was treated as the starting target configuration for each set of shelves, resulting in three cycles per set of shelves and six overall cycles. Finally, each cycle was repeated for three trials, resulting in a total of 18 trials. This configuration was run with a range of risk thresholds for both the RRT high-level planner and the trajectory optimizer. Table 5.7 provides the results. When the RRT is run with a low risk threshold, it frequently fails to find a solution. These failures arise both due to (a) incorrectly classifying the high-level waypoints as in collision and (b) failure to connect the start and goal configurations. Conversely, a high RRT risk threshold combined with a low optimizer risk threshold leads to high-level plans that the trajectory optimizer is unable to track. This results in the optimizer

frequently becoming stuck, shown in the final row of Table 5.7. When risk thresholds are chosen according to the results in Sections 5.5.3 and 5.5.4, a high rate of success is achieved.

Table 5.7 also provides timing metrics for the hardware demonstrations. The RRT timing metrics exclude cases where the RRT classifies a high-level waypoint as in-collision, in which case the planner immediately reports a failure. Optimization times are averaged across all planning trials. Despite these multiple processes running on the computer, real-time trajectory optimization is maintained.

5.6 Extending SPLANNING to Mobile Manipulation

In this section we discuss how SPLANNING can be applied to a mobile manipulation. This section introduces a new Spherical Forward Occupancy formulation for mobile manipulators that is complementary with the SPLANNING framework. We introduce the method to apply SPLANNING to mobile manipulation as the algorithm Go-Fetch.

5.6.1 Mobile Manipulator Kinematics

Given a compact time interval $T \subset \mathbb{R}$, we define a trajectory for the robot’s configuration as $q : T \rightarrow Q \subset \mathbb{R}^{n_q}$ and a trajectory for the velocity as $\dot{q} : T \rightarrow \mathbb{R}^{n_q}$. where:

$$q(t) = \begin{bmatrix} q_b(t) \\ q_a(t) \end{bmatrix} \quad (5.28)$$

where $q_b(t) = [q_x(t), q_y(t), q_\theta(t)]^T \in \mathbb{R}^3$ are the time-dependent states of the mobile base and q_a are the time-dependent states of the manipulator. We assume that the mobile manipulator follows dubins paths hence the mobile base state $q_b(t)$ is given by:

$$q_b(t) = q_b(0) + \int_0^t \begin{bmatrix} v(\tau) \cos q_\theta(\tau) \\ v(\tau) \sin q_\theta(\tau) \\ \omega(\tau) \end{bmatrix} d\tau \quad (5.29)$$

where $v(t)$ is the time-dependent linear velocity of the mobile base and $w(t)$ is the time-dependent angular velocity of the mobile base.

Let $\text{FK}_b : Q_b \rightarrow \mathbb{R}^{4 \times 4}$ map the robot base’s time-dependent state to the pose of the robot base in the world frame W . Let $H_f^e \in \mathbb{R}^{4 \times 4}$ be the transformation from frame e to frame f . For the mobile manipulator’s arm, let F denote the mobile base body-fixed frame and A the arm base frame fixed at the point of attachment to the mobile base. We then define

the transformation between the mobile base body-fixed frame and the mounting point of the arm as $H_A^F \in SE(3)$. We formulate the arm's kinematics similarly to what is done in [82] where we assume that the arm's j^{th} reference frame $\{\hat{x}_j, \hat{y}_j, \hat{z}_j\}$ is attached to the arm's j^{th} revolute joint, and that $\hat{z}_j = [0, 0, 1]^\top$ points in direction of the j^{th} joint's axis of rotation. Then the forward kinematics $\text{FK}_j : Q \rightarrow \mathbb{R}^{4 \times 4}$ maps the arm's time-dependent configuration to the pose of the j^{th} joint in the world frame:

$$\begin{aligned} \text{FK}_j(q(t)) &= \text{FK}_b(q_b(t)) \cdot H_A^F \cdot \prod_{l=1}^j H_l^{l-1}(q_{l+3}(t)) \\ &= \begin{bmatrix} R_j(q(t)) & p_j(q(t)) \\ 0 & 1 \end{bmatrix}, \end{aligned} \quad (5.30)$$

$$R_j(q(t)) := R_b(q_b(t)) \cdot R_A^F \cdot \prod_{l=1}^j R_l^{l-1}(q_{l+3}(t)) \quad (5.31)$$

$$p_j(q(t)) := p_b(q_b(t)) + R_b(q_b(t)) \cdot \left(p_A^F + R_A^F \sum_{l=1}^j R_l^A(q(t)) p_l^{l-1} \right) \quad (5.32)$$

5.6.2 Mobile Manipulator Forward Occupancy

In this subsection, we define the *forward occupancy* as the volume occupied by the mobile manipulator in the workspace W_s . Let $L_b^F \subset W_s \subset \mathbb{R}^3$ denote the volume occupied by the mobile base of the mobile manipulator expressed in the mobile base body-fixed frame F . Then the forward occupancy of the mobile base is the map $\text{FO}_b : Q_b \rightarrow \mathcal{P}(W_s)$ defined as:

$$\text{FO}_b(q_b(t)) = \text{FK}_b(q_b(t)) L_b^F \quad (5.33)$$

where $\text{FK}_b(q_b(t)) L_b^F$ is the transformed volume of the mobile base.

Let $n_{q_a} = n_q - 3 \in \mathbb{N}$ be the number of states for just the manipulator. Let $L_j \subset W_s \subset \mathbb{R}^3$ denote the volume occupied by the arm's j^{th} link with respect to the j^{th} reference frame. Then the forward occupancy of link j of the arm is the map $\text{FO}_j : Q \rightarrow \mathcal{P}(W_s)$ defined as

$$\text{FO}_j(q) = p_j(q) \oplus R_j(q) L_j, \quad (5.34)$$

where $p_j(q)$ and $R_j(q)$ specify the pose of the arm's j^{th} joint, and $R_j(q) L_j$ is the rotated volume of the arm's j^{th} link. The volume occupied by the entire arm in the workspace is

then defined by the map $\text{FO} : Q \rightarrow W_s$ such that:

$$\text{FO}(q) = \text{FO}_b(q_b(t)) \cup \left(\bigcup_{j=1}^{n_{qa}} \text{FO}_j(q) \right) \subset W_s. \quad (5.35)$$

Because the geometry of any of the robot's links may be arbitrarily complex, we make the following assumptions:

Assumption 51. *There exists a finite collection of closed balls $\{B_2(c_i^B, r_i)\}_{i=1}^{n_{sb}}$ such that*

$$B_2(c, r) := \{x \in \mathbb{R}^3 \mid \|x - c\|_2 \leq r\}. \quad (5.36)$$

Moreover,

$$L_b^F \subseteq \bigcup_{i=1}^{n_{sb}} B_2(c_i^B, r_i). \quad (5.37)$$

where $n_{sb} \in \mathbb{N}$ is the number of spheres needed to overapproximate the volume of the base L_b^F . Consequently, for any base configuration $q_b(t)$, the base forward occupancy in the world frame $\text{FO}_b(q_b(t))$ is overapproximated by the transformed sphere union

$$\text{FO}_b(q_b(t)) \subseteq \bigcup_{i=1}^M B_2(p_b(q_b(t)) + R_b(q_b(t)) c_i^B, r_i) \quad (5.38)$$

We define collections of spheres as follows:

$$S_b(q_b(t)) = \bigcup_{i=1}^{n_{sb}} B_2(p_b(q_b(t)) + R_b(q_b(t)) c_i^B, r_i) \quad (5.39)$$

For the manipulator, we extend Assumption 46 that simplifies the construction of an overapproximation to the forward occupancy in the following way:

Assumption 52. *Given a mobile manipulator's configuration q and any arm joint index $j \in \{1, \dots, n_{qa}\}$, there exists a ball with center $p_j(q)$ and radius r_j that overapproximates the volume occupied by the j^{th} joint of the manipulator in W_s . We further assume that the manipulator link volume L_j is a subset of the tapered capsule formed by the convex hull of the balls overapproximating the j^{th} and $(j+1)^{\text{th}}$ joints of the manipulator.*

Following Assum. 52, we now follow the same exposition from Equations (5.7)–(5.8) to obtain that the volume occupied by the j^{th} link (5.34) is overapproximated by

$$\text{FO}_j(q) \subset TC_j(q(t)) \quad (5.40)$$

Thus the forward occupancy of the mobile manipulator is overapproximated by

$$\text{FO}(q) \subset S_b(q_b(t)) \cup \left(\bigcup_{j=1}^{n_{qa}} TC_j(q(t)) \right) \subset W_s, \quad (5.41)$$

respectively.

Now we abuse notation and redefine $\text{FO}(q(T))$ as the forward occupancy of the mobile manipulator over an entire time interval T . Notably, the mobile manipulator is *collision-free* over the time interval T if $\text{FO}(q(T))$ does not intersect with the environment.

Next, we summarize the construction of the mobile manipulator's safety representation using the following theorem:

Theorem 53. *Given a mobile manipulator comprised of a mobile base that follows dubins paths, and a manipulator with $n_{qa} \in \mathbb{N}$ revolute joints and a time partition T of a finite set of intervals T_i (i.e., $T = \bigcup_{i=1}^{n_t} T_i$), the swept volume corresponding to the robot's motion over T is overapproximated by a collection of L_2 balls in $\mathbb{R}^3$, which we call the Spherical Forward Occupancy of the mobile manipulator ($\mathcal{SFO}_{mm}$) defined as*

$$\mathcal{SFO}_{mm} = \mathcal{SFO}_b \cup \mathcal{SFO}_a \quad (5.42)$$

where $\mathcal{SFO}_b$ is the Spherical Forward Occupancy of the mobile base is given by:

$$\mathcal{SFO}_b = \bigcup_{i=1}^{n_t} \bigcup_{n=1}^{n_{sb}} S_{b,i,n}(q(T_i; k)) \quad (5.43)$$

and $\mathcal{SFO}_a$ is the Spherical Forward Occupancy of the manipulator is given by:

$$\mathcal{SFO}_a = \bigcup_{j=1}^{n_{qa}} \bigcup_{i=1}^{n_t} \bigcup_{m=1}^{n_s} S_{j,i,m}(q(T_i; k)) \quad (5.44)$$

where each $S_{j,i,m}(q(T_i; k))$ and each $S_{b,i,n}(q(T_i; k))$ is an L_2 ball in $\mathbb{R}^3$, $n_s \in \mathbb{N}$ and $n_{sb} \in \mathbb{N}$ are parameters that specify the number of closed balls overapproximating each of the manipulator's links and the number of closed balls overapproximating the mobile base, respectively and k is a trajectory parameter that characterizes the motion of the robot over T .

5.6.3 Risk-Aware Motion Planning for a Mobile Manipulator

At runtime, Go-Fetch computes probabilistically safe trajectories by solving an optimization problem that limits the probability of the robot's forward occupancy intersecting the environment. Go-Fetch tries to solve the following optimization problem in a receding-horizon

manner:

$$\min_{k \in K} \quad \text{cost}(k) \quad (5.45)$$

$$q(t; k) \in [q_{\text{lim}}^-, q_{\text{lim}}^+] \quad \forall t \in T \quad (5.46)$$

$$\dot{q}(t; k) \in [\dot{q}_{\text{lim}}^-, \dot{q}_{\text{lim}}^+] \quad \forall t \in T \quad (5.47)$$

$$\text{P}(\text{FO}(q(t; k)) \cap \mathcal{E} \neq \emptyset) < \beta \quad \forall t \in T \quad (5.48)$$

The cost function (5.45) is a user-defined objective, such as bringing the robot close to a desired goal. We enforce limits on joint positions (5.46) and velocities (5.47). Finally, (5.48) is a safety constraint ensuring that the probability of the robot's forward occupancy $\text{FO}(q(t; k))$ intersecting the environment $\mathcal{E}$ stays below the risk threshold β .

Leveraging Theorem 49, Go-Fetch can now generate risk-aware motion plans in a receding-horizon fashion. Prior to planning, a Normalized Gaussian Splatting representation of the scene is constructed. At every planning iteration, the robot is given $t_{\text{plan}} \leq 0.5$ seconds to find a feasible trajectory by solving

$$\min_{k \in K} \quad \text{cost}(k) \quad (\text{Go-Fetch-Opt}) \quad (5.49)$$

$$\text{s.t. } q_j(T_i; k) \subseteq [q_{j,\text{lim}}^-, q_{j,\text{lim}}^+] \quad \forall (i, j) \in N_t \times N_q \quad (5.50)$$

$$\dot{q}_j(T_i; k) \subseteq [\dot{q}_{j,\text{lim}}^-, \dot{q}_{j,\text{lim}}^+] \quad \forall (i, j) \in N_t \times N_q \quad (5.51)$$

$$\sum_{\substack{(j,m) \in \\ N_q \times n_s}} \mathcal{C}(S_{j,i,m}(q(T_i; k))) + \sum_{n \in N_{s_b}} \mathcal{C}(S_{b,i,n}(q_b(T_i; k))) < \alpha \cdot \beta, \quad (5.52)$$

$\forall i \in N_t$

where

$$\mathcal{C}(S) = \left[1 - \exp\left(-\frac{1}{4\pi} \mathcal{H}(S)\right) \right] \quad (5.53)$$

and $k \in K$ is the trajectory parameter (Appendix. D.2.1) and $\text{cost}(k)$ is a task-specific cost function. The spheres in the robots reachable set $\{S_{j,i,m}\}_{m=1}^{n_s}$ and $\{S_{b,i,n}\}_{n=1}^{n_{s_b}}$ are functions of the robot's position (and hence trajectory parameter) and are computed repeatedly while numerically solving (Go-Fetch-Opt). Safety is enforced using the collision-avoidance constraint (5.52). If a solution is not found, the robot executes a braking maneuver using the trajectory parameter found in the previous planning iteration.

5.7 Conclusion

This chapter introduced a framework for evaluating collisions in a radiance field model. The result is SPLANNING, a trajectory optimizer that synthesizes trajectories while constraining the probability of colliding with the environment. To enable the constraint formulation, we presented a normalized variant of 3D Gaussian Splatting suitable for risk-aware planning while achieving similar reconstruction quality to standard 3DGS. The risk-aware constraint was demonstrated to outperform other radiance field methods for collision avoidance while also being efficient to compute in real-time planning. Planner evaluations demonstrated that the trajectory optimizer achieves a high number of successes in cluttered environments. Hardware demonstrations indicated that the planner is viable for real-world use. This chapter also introduces a sphere reachable set-based safety representation for mobile manipulators and a method for safe mobile manipulation called Go-Fetch that leverages the normalized gaussian splat and a similar constraint to that of SPLANNING.

Building on these methods, there are several promising avenues for future work. First, adding support for dynamic objects would aid in planning in real-world scenarios. Second, the Normalized 3DGS may be improved to increase the reconstruction quality. A key difference between Normalized 3DGS and standard 3DGS is the difference in filtering techniques; standard 3DGS applies a screen-space low-pass filter, while Normalized 3DGS applies the filter in world-space. This adaptation allows for a probabilistic interpretation of the 3D Gaussians but is suboptimal for visual reconstruction. Hence, future work avenues will explore other low-pass filtering techniques that improve reconstruction quality while maintaining the correctness for planning. Furthermore, in instances where the Normalized 3DGS is imperfect, another future research direction will explore closed-loop, active exploration strategies for improving the reconstruction quality online. Next, in our present formulation, although α and β are conceptually distinct, they are mathematically interchangeable. Future work will investigate this issue by deriving alternate chance constraints leveraging different approximation inequalities other than Markov’s inequality. Finally, incorporating a real-time simultaneous localization and mapping (SLAM) module would enable navigation of scenes that have not been previously mapped.

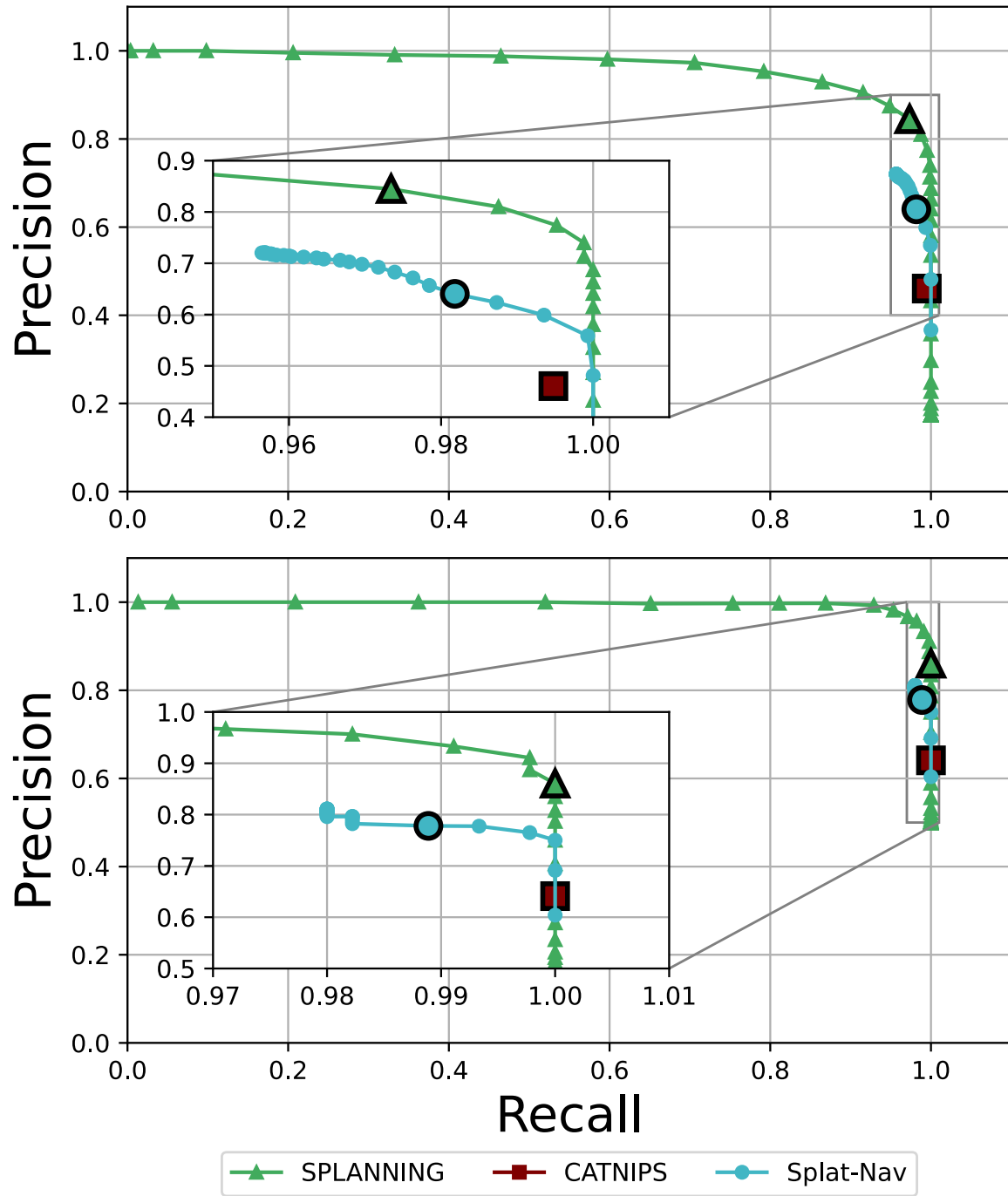


Figure 5.5: Two Precision-Recall curves are presented. Treating the collision constraints from SPLANNING, CATNIPS, Splat-Nav as classifiers, where a positive indicates collision and a negative indicates no collision, the top plot shows the Precision-Recall over individual spheres while the bottom plot show Precision-Recall over configurations. Highlighted markers indicate nominal parameters $\alpha = \beta = 0.025$, $\sigma = 0.99$, and $\sigma = 1$ for SPLANNING, CATNIPS, and Splat-Nav respectively.

CHAPTER 6

Conclusion: Towards End-to-End Safe Motion Planning for High-Dimensional Robots

This dissertation develops a set of computationally tractable models, algorithms, and representations aimed at advancing safe, real-time motion planning for high-dimensional robotic systems operating in visually rich and uncertain environments such as warehouses, crowded public spaces, and unstructured outdoor settings.

Robots deployed in these domains must plan in real time, provide formal safety guarantees without undue conservatism, reason about uncertain agents, and leverage visual inputs so they can react as scenes change. Existing methods struggle on three fronts: 1) generating dynamically feasible, high-quality trajectories for high-DoF systems is computationally expensive, and treating those trajectories as loose waypoints forces frequent replanning or emergency halts when the mid-level controller cannot track them; 2) handling environmental uncertainty often resorts to worst-case bounds or restrictive distributional assumptions, producing overly conservative behavior or slow sample-based estimates; and 3) standard perception pipelines convert images into occupancy grids or convex primitives that either discard fine geometric detail or incur heavy reconstruction costs, degrading downstream planning and manipulation.

To address these shortcomings, this work pursues three complementary capabilities: 1) producing high-quality, dynamics-aware high-level plans fast enough for receding-horizon controllers; 2) quantifying and enforcing collision risk under expressive uncertainty models in real time without excessive conservatism; and 3) closing the loop between dense visual perception and planning so planners can reason directly with appearance and geometric uncertainty. Across Chapters 2–5, I introduce PHLAME and BLAZE (fast AGHF-based trajectory generators), RADIUS and DIAMETER (real-time risk-aware motion planning), and SPLANNING and Go-Fetch (probabilistic planning in Normalized Gaussian splats). Collectively, these tools provide the algorithmic primitives needed for a end-to-end safe motion planning algorithm that goes directly from vision to safe motion plans: PHLAME/BLAZE would generate

dynamically feasible high-level plans based on the same normalized gaussian splatting representation used in SPLANNING; techniques from RADIUS/DIAMETER would be used to account for uncertainty in dynamic obstacles and techniques from SPLANNING would be used to account for static obstacles, while the normalized gaussian splat would enable direct reasoning from a high fidelity vision-based representation. Integrating these components is the natural next step toward a robust, end-to-end safe motion-planning solution.

6.1 Summary of Contributions

The first set of contributions (Chapters 2 and 3) introduce AGHF-based trajectory generators that recast trajectory optimization as a curve-shortening PDE solved by forward integration of a system of ODEs. By combining pseudospectral collocation, spatial vector algebra, and analytical expressions for the AGHF and its Jacobian, as well as a Phase-1/Phase-2 strategy for constrained optimization, these methods dramatically reduce the number of AGHF evaluations required to obtain high-quality trajectories. The result is a rapid trajectory generator that produces dynamically feasible motions for systems with forty or more degrees of freedom in seconds, providing reliable high-level plans suitable for downstream receding-horizon safety modules.

The second set of contributions (Chapter 4) develop RADIUS and DIAMETER, real-time risk-aware motion planners that integrate chance-constrained optimization with reachability analysis. RADIUS and DIAMETER supply parallelizable, closed-form over-approximations of collision probability for arbitrary twice-differentiable obstacle distributions and compute analytic gradients with respect to trajectory parameters that correspond to potential trajectories. These approximations enable online optimization that enforces user-specified risk thresholds over temporal intervals while avoiding the extreme conservatism of worst-case approaches; empirical results show improved agility and bounded safety on both simulated scenarios and hardware demonstrations.

The third set of contributions (Chapter 5) introduce a normalized Gaussian splatting formulation that brings probabilistic planning into radiance fields. By reformulating neural radiance field representations as normalized 3D Gaussians and deriving tight analytic upper bounds on collision probability, SPLANNING (and its mobile-manipulation variant Go-Fetch) enable receding-horizon, risk-aware trajectory optimization directly on a dense, probabilistic visual representation. This approach preserves geometric and appearance detail useful for downstream tasks while avoiding heavy reconstruction pipelines; experiments on cluttered scenes and on a Kinova Gen3 manipulator demonstrate measurable gains in planning speed and collision avoidance.

6.2 Future Work

While the contributions presented in this dissertation present significantly enhance trajectory optimization and safe motion planning for high-dimensional robot systems, they also exhibit certain limitations.

PHLAME and BLAZE’s current formulations assume differentiable dynamics and do not yet handle contact switching or complementarity constraints natively; extension to hybrid, contact-rich domains will require new formulations or hybridization strategies. RADIUS and DIAMETER’s closed-form over-approximations rely on the twice-differentiability of obstacle location probability densities and, in adversarial or strongly non-smooth regimes, may generate conservative risk bounds. SPLANNING and Go-Fetch depend on reasonably accurate and up-to-date splat reconstructions. Currently, these splats are also built offline so quickly adapting to extreme scene changes remains challenging. Finally, a full, certifiable end-to-end integration (perception to global dynamics-aware plan to online risk-aware receding control) has not yet been realized and validated at operational scale.

To address these challenges this dissertation suggests multiple concrete research directions:

- **End-to-end integration and certification.** Construct and empirically validate an integrated pipeline that combines the Gaussian splatting scene representation with the AGHF-based high-level planners to generate high-level plans that can then be passed to a mid-level planner that uses techniques from SPLANNING and RADIUS to deal with static and dynamic obstacles. In this combined planner, visual inputs will be used to build and update a Normalized Gaussian-splat that encodes dense geometry, appearance, and uncertainty; PHLAME/BLAZE will be used to produce dynamically feasible, time-parameterized high-level trajectories using the Normalized Gaussian Splatting Constraint for collision avoidance; techniques from RADIUS/DIAMETER will enable risk-aware motion planning for moving obstacles; and SPLANNING supplies tight, analytic collision bounds for static scene geometry, so the receding-horizon optimizer can reason about both static and dynamic risk without costly conversions. Because the high-level plans already respect dynamics and provide warm starts, the online planner can focus on local obstacle avoidance and fast replanning, yielding less conservative safety plans.
- **Hybrid dynamics and contact.** Extending AGHF/BLAZE to contact-rich domains requires confronting discontinuities and combinatorial mode structure introduced by contact switching and complementarity conditions. A potential approach could be to treat contacts as smoothly activated constraints rather than hard, discontinuous switches. Practically, we can leverage a two-phase approach similar to that proposed

by BLAZE, where in the first phase one could solve a relaxed (smoothed) problem to produce a feasible warm start and then incrementally tighten penalties or projects onto harder complementarity conditions to recover physically valid contact sequences. This hybrid formulation would then be combined with the existing pseudospectral collocation and spatial-vector algebra machinery so that gradients and PDE solvers remain efficient for high-DoF systems.

- **Normalized Gaussian Splatting SLAM.** Apply Gaussian Splatting SLAM [126] (Simultaneous Localization and Mapping) techniques to build a Normalized Gaussian splat representation online using visual SLAM. In generating these representations, we must also account for the uncertainty in the splat representation as it is being constructed, as it may not have fully converged to an accurate representation of the scene geometry. Uncertainty quantification techniques like that employed in [28] could be used to bake map uncertainty into the collision constraints and risk bounds so safe plans can be generated even before the map fully converges.

This thesis makes substantive progress toward practical, certifiable end-to-end safe planning by leveraging rapid trajectory optimization, analytic risk approximation, and a differentiable, high-fidelity scene representation that supports both obstacle avoidance and downstream reasoning.

Future research directions would focus on enabling real-time construction and updating of the normalized scene representation, extending AGHF-based planners to hybrid, contact-rich dynamics, and integrating these components into a unified, certifiable framework. These advances would materially expand the environments in which high-DoF robots can operate reliably and safely, enabling safe navigation through cluttered and unstructured areas like warehouses and public spaces.

APPENDIX A

PHLAME Appendices

A.1 Proof of Theorem 5

Proof. The Affine Geometric Heat Flow (AGHF) Equation is given by

$$\frac{\partial x}{\partial s}(t, s) = G^{-1}(x(t, s)) \left(\frac{d}{dt} \frac{\partial L}{\partial \dot{x}_s}(x_s(t), \dot{x}_s(t)) - \frac{\partial L}{\partial x_s}(x_s(t), \dot{x}_s(t)) \right) \quad (\text{A.1})$$

Let

$$L(x_s(t), \dot{x}_s(t)) = (\dot{x}_s(t) - F_d(x_s(t)))^T G(x_s(t)) (\dot{x}_s(t) - F_d(x_s(t))) \quad (\text{A.2})$$

where

$$\begin{aligned} G(x_s(t)) &= (\bar{F}(x_s(t))^{-1})^T K \bar{F}(x_s(t))^{-1} \\ G(x_s(t)) &= \underbrace{\begin{bmatrix} I_{N \times (2N-m)} & 0_{N \times m} \\ 0_{N \times (2N-m)} & B^{-1} H(x_{P1}(t, s)) \end{bmatrix}}_{(\bar{F}(x_s(t))^{-1})^T}^T \underbrace{\begin{bmatrix} kI_{N \times N} & 0_{N \times N} \\ 0_{N \times N} & I_{N \times N} \end{bmatrix}}_K \underbrace{\begin{bmatrix} I_{N \times (2N-m)} & 0_{N \times m} \\ 0_{N \times (2N-m)} & B^{-1} H(x_{P1}(t, s)) \end{bmatrix}}_{(\bar{F}(x_s(t))^{-1})} \end{aligned} \quad (\text{A.3})$$

We assume, without loss of generality (WLOG), that $B = I$. Under this assumption, multiplying the matrices in (A.3) yields

$$G(x_s(t)) = \begin{bmatrix} kI_{N \times N} & 0_{N \times N} \\ 0_{N \times N} & H^T H \end{bmatrix} \quad (\text{A.4})$$

From here for conciseness we drop the dependence on (t, s) for the following equations (i.e. $x(t, s)$ is denoted by x). We also drop the dependence on the x terms for the dynamics functions (i.e. $H(x_{P1})$ is denoted by just H and so on).

To compute (A.1), we need to compute the derivatives $\frac{d}{dt} \frac{\partial L}{\partial \dot{x}}$ and $\frac{\partial L}{\partial x}$. We begin by showing that $\frac{d}{dt} \frac{\partial L}{\partial \dot{x}} = \Omega_1$. First we must compute the derivative $\frac{\partial L}{\partial \dot{x}}$,

$$\frac{\partial L}{\partial \dot{x}} = 2G(\dot{x} - F_d) = 2 \underbrace{\begin{bmatrix} kI_{N \times N} & 0_{N \times N} \\ 0_{N \times N} & H^T H \end{bmatrix}}_G \left(\underbrace{\begin{bmatrix} \dot{x}_{P1} \\ \dot{x}_{P2} \end{bmatrix}}_{\dot{x}} - \underbrace{\begin{bmatrix} x_{P2} \\ -H^{-1}C \end{bmatrix}}_{F_d} \right) = 2 \begin{bmatrix} k(\dot{x}_{P1} - x_{P2}) \\ H^T H \dot{x}_{P2} + H^T C \end{bmatrix} \quad (\text{A.5})$$

taking the time derivative of this yields:

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{x}} = 2 \begin{bmatrix} k(\ddot{x}_{P1} - \dot{x}_{P2}) \\ (\dot{H}^T H + H^T \dot{H})\dot{x}_{P2} + H^T H \ddot{x}_{P2} + \dot{H}^T C + H^T \dot{C} \end{bmatrix} \quad (\text{A.6})$$

and multiplying by G^{-1} yields

$$\Omega_1 = G^{-1} \left(\frac{d}{dt} \frac{\partial L}{\partial \dot{x}} \right) = 2 \begin{bmatrix} \ddot{x}_{P1} - \dot{x}_{P2} \\ (H^T H)^{-1} ((\dot{H}^T H + H^T \dot{H})\dot{x}_{P2} + H^T H \ddot{x}_{P2} + \dot{H}^T C + H^T \dot{C}) \end{bmatrix} \quad (\text{A.7})$$

Next we show that $G^{-1} \frac{\partial L}{\partial x} = \Omega_2 - \Omega_3 + \Omega_4$

First, taking the derivative of (A.2) wrt. x , and taking transposes to ensure that the resultant derivatives are column vectors we obtain:

$$\frac{\partial L}{\partial x} = -\frac{\partial F_d^T}{\partial x} G(\dot{x} - F_d) + (\dot{x} - F_d)^T \frac{\partial G}{\partial x} (\dot{x} - F_d) + \left((\dot{x} - F_d)^T G \left(-\frac{\partial F_d}{\partial x} \right) \right)^T \quad (\text{A.8})$$

expanding the first and third terms yields

$$\frac{\partial L}{\partial x} = -\frac{\partial F_d^T}{\partial x} G \dot{x} + \frac{\partial F_d^T}{\partial x} G F_d + (\dot{x} - F_d)^T \frac{\partial G}{\partial x} (\dot{x} - F_d) + \left(-\dot{x}^T G \frac{\partial F_d}{\partial x} \right)^T + \left(F_d^T G \frac{\partial F_d}{\partial x} \right)^T \quad (\text{A.9})$$

Since G is symmetric (because kI and $H^T H$ are symmetric) the last two terms can be simplified yielding:

$$\frac{\partial L}{\partial x} = -\frac{\partial F_d^T}{\partial x} G \dot{x} + \frac{\partial F_d^T}{\partial x} G F_d + (\dot{x} - F_d)^T \frac{\partial G}{\partial x} (\dot{x} - F_d) - \frac{\partial F_d^T}{\partial x} G \dot{x} + \frac{\partial F_d^T}{\partial x} G F_d \quad (\text{A.10})$$

Grouping terms and simplifying yields

$$\frac{\partial L}{\partial x} = 2 \frac{\partial F_d^T}{\partial x} G F_d - 2 \frac{\partial F_d^T}{\partial x} G \dot{x} + (\dot{x} - F_d)^T \frac{\partial G}{\partial x} (\dot{x} - F_d) \quad (\text{A.11})$$

Let $FD_0 = -H^{-1}C$. Next we derive the constituent terms in (A.11). The first term is given by:

$$2 \frac{\partial F_d^T}{\partial x} G F_d = 2 \underbrace{\begin{bmatrix} 0_{N \times N} & \frac{\partial FD_0^T}{\partial x_{P1}} \\ I_{N \times N} & \frac{\partial FD_0^T}{\partial x_{P2}} \end{bmatrix}}_{\frac{\partial F_d^T}{\partial x}} \underbrace{\begin{bmatrix} kI_{N \times N} & 0_{N \times N} \\ 0_{N \times N} & H^T H \end{bmatrix}}_G \underbrace{\begin{bmatrix} x_{P2} \\ -H^{-1}C \end{bmatrix}}_{F_d} = 2 \begin{bmatrix} -\frac{\partial FD_0^T}{\partial x_{P1}} H^T C \\ kx_{P2} - \frac{\partial FD_0^T}{\partial x_{P2}} H^T C \end{bmatrix} \quad (\text{A.12})$$

and the second term:

$$2 \frac{\partial F_d^T}{\partial x} G \dot{x} = 2 \underbrace{\begin{bmatrix} 0_{N \times N} & \frac{\partial FD_0^T}{\partial x_{P1}} \\ I_{N \times N} & \frac{\partial FD_0^T}{\partial x_{P2}} \end{bmatrix}}_{\frac{\partial F_d^T}{\partial x}} \underbrace{\begin{bmatrix} kI_{N \times N} & 0_{N \times N} \\ 0_{N \times N} & H^T H \end{bmatrix}}_G \underbrace{\begin{bmatrix} \dot{x}_{P1} \\ \dot{x}_{P2} \end{bmatrix}}_{\dot{x}} = 2 \begin{bmatrix} \frac{\partial FD_0^T}{\partial x_{P1}} H^T H \dot{x}_{P2} \\ k\dot{x}_{P1} + \frac{\partial FD_0^T}{\partial x_{P2}} H^T H \dot{x}_{P2} \end{bmatrix} \quad (\text{A.13})$$

Collecting and rearranging the first 2 terms of (A.11) yields:

$$2 \frac{\partial F_d^T}{\partial x} G F_d - 2 \frac{\partial F_d^T}{\partial x} G \dot{x} = \begin{bmatrix} -\frac{\partial FD_0^T}{\partial x_{P1}} H^T (2C + 2H\dot{x}_{P2}) \\ -\frac{\partial FD_0^T}{\partial x_{P2}} H^T (2C + 2H\dot{x}_{P2}) \end{bmatrix} - \begin{bmatrix} \mathbf{0} \\ 2k(\dot{x}_{P1} - x_{P2}) \end{bmatrix} \quad (\text{A.14})$$

and multiplying them by G^{-1} gives us Ω_2 and Ω_3 :

$$G^{-1} \left(\frac{\partial F_d^T}{\partial x} G F_d + \left(F_d^T G \frac{\partial F_d}{\partial x} \right)^T - 2 \frac{\partial F_d^T}{\partial x} G \dot{x} \right) = \underbrace{\begin{bmatrix} -\frac{1}{k} I_{N \times N} \frac{\partial F D_0^T}{\partial x_{P1}} H^T (2C + 2H \dot{x}_{P2}) \\ -(H^T H)^{-1} \frac{\partial F D_0^T}{\partial x_{P2}} H^T (2C + 2H \dot{x}_{P2}) \end{bmatrix}}_{\Omega_2} - \underbrace{\begin{bmatrix} \mathbf{0} \\ 2k(H^T H)^{-1}(\dot{x}_{P1} - x_{P2}) \end{bmatrix}}_{\Omega_3} \quad (\text{A.15})$$

The third term of (A.11) is given by:

$$(\dot{x} - F_d)^T \frac{\partial G}{\partial x} (\dot{x} - F_d) = (\dot{x} - F_d)^T \begin{bmatrix} 0_{N \times N \times 2N} & \frac{\partial(H^T H)}{\partial x} \end{bmatrix} (\dot{x} - F_d), \quad (\text{A.16})$$

where $\frac{\partial(H^T H)}{\partial x_i}$ be the partial derivative of $H^T H$ wrt. the i th element of the $2N \times 1$ column vector x . Doing the vector-tensor-vector multiplication in (A.16) yields an $2N \times 1$ column vector of the following form:

$$(\dot{x} - F_d)^T \frac{\partial G}{\partial x} (\dot{x} - F_d) = \begin{bmatrix} (\dot{x} - F_d)^T \begin{bmatrix} 0_{N \times N} & \frac{\partial(H^T H)}{\partial x_1} \end{bmatrix} (\dot{x} - F_d) \\ \vdots \\ (\dot{x} - F_d)^T \begin{bmatrix} 0_{N \times N} & \frac{\partial(H^T H)}{\partial x_{2N}} \end{bmatrix} (\dot{x} - F_d) \end{bmatrix} \quad (\text{A.17})$$

and applying (2.3) to (A.17) yields:

$$(\dot{x} - F_d)^T \frac{\partial G}{\partial x} (\dot{x} - F_d) = \begin{bmatrix} (\dot{x}_{P2} - F D_0)^T \frac{\partial(H^T H)}{\partial x_1} (\dot{x}_{P2} - F D_0) \\ \vdots \\ (\dot{x}_{P2} - F D_0)^T \frac{\partial(H^T H)}{\partial x_{2N}} (\dot{x}_{P2} - F D_0) \end{bmatrix} \quad (\text{A.18})$$

Notice that since H is only a function of x_{P1} , then

$$\begin{bmatrix} \frac{\partial(H^T H)}{\partial x_{N+1}} \\ \vdots \\ \frac{\partial(H^T H)}{\partial x_{2N}} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \quad (\text{A.19})$$

and

$$\begin{aligned} (\dot{x} - F_d)^T \frac{\partial G}{\partial x} (\dot{x} - F_d) &= \begin{bmatrix} \begin{bmatrix} (\dot{x}_{P2} - F D_0)^T \frac{\partial(H^T H)}{\partial x_1} (\dot{x}_{P2} - F D_0) \\ \vdots \\ (\dot{x}_{P2} - F D_0)^T \frac{\partial(H^T H)}{\partial x_N} (\dot{x}_{P2} - F D_0) \end{bmatrix} \\ \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \end{bmatrix} \\ &= \begin{bmatrix} (\dot{x}_{P2} - F D_0)^T \frac{\partial(H^T H)}{\partial x_{P1}} (\dot{x}_{P2} - F D_0) \\ \mathbf{0} \end{bmatrix} \end{aligned} \quad (\text{A.20})$$

Next we show how to further simplify (A.20) to yield Ω_4 . First, let β_i be defined as the i th entry of $(\dot{x} - F_d)^T \frac{\partial G}{\partial x} (\dot{x} - F_d)$ given by:

$$\beta_i = (\dot{x}_{P2} - F D_0)^T \frac{\partial(H^T H)}{\partial x_i} (\dot{x}_{P2} - F D_0) \quad (\text{A.21})$$

evaluating $\frac{\partial(H^T H)}{\partial x_i}$ and expanding (A.21) gives:

$$\beta_i = (\dot{x}_{P2} - F D_0)^T \frac{\partial H^T}{\partial x_i} H (\dot{x}_{P2} - F D_0) + (\dot{x}_{P2} - F D_0)^T H^T \frac{\partial H}{\partial x_i} (\dot{x}_{P2} - F D_0) \quad (\text{A.22})$$

Notice β_i is a scalar, and so are each of its constituent terms. Therefore the second term of β_i can be expressed in the following way

$$\begin{aligned}
(\dot{x}_{P2} - FD_0)^T H^T \frac{\partial H}{\partial x_i} (\dot{x}_{P2} - FD_0) &= \left((\dot{x}_{P2} - FD_0)^T H^T \frac{\partial H}{\partial x_i} (\dot{x}_{P2} - FD_0) \right)^T \\
&= (\dot{x}_{P2} - FD_0)^T \frac{\partial H^T}{\partial x_i} H (\dot{x}_{P2} - FD_0)
\end{aligned} \tag{A.23}$$

Using (A.23) to simplify β_i gives

$$\beta_i = 2 (\dot{x}_{P2} - FD_0)^T \frac{\partial H^T}{\partial x_i} H (\dot{x}_{P2} - FD_0) = 2 \left[\frac{\partial H}{\partial x_i} (\dot{x}_{P2} - FD_0) \right]^T H (\dot{x}_{P2} - FD_0) \tag{A.24}$$

Applying this to (A.20) yields

$$(\dot{x} - F_d)^T \frac{\partial G}{\partial x} (\dot{x} - F_d) = \begin{bmatrix} 2 \left[\frac{\partial H}{\partial x_{P1}} (\dot{x}_{P2} - FD_0) \right]^T H (\dot{x}_{P2} - FD_0) \\ \mathbf{0} \end{bmatrix} \tag{A.25}$$

Note that to ensure that (A.25) evaluates to the same terms as (A.20) the 3rd dimension of the tensor $\frac{\partial H}{\partial x_{P1}}$ must be used in the tensor-vector multiplication with $\dot{x}_{P2} - FD_0$.

Lastly, multiplying (A.25) by G^{-1} gives us Ω_4 :

$$G^{-1} (\dot{x} - F_d)^T \frac{\partial G}{\partial x} (\dot{x} - F_d) = \begin{bmatrix} 2^{\frac{1}{k}} \left[\frac{\partial H}{\partial x_{P1}} (\dot{x}_{P2} - FD_0) \right]^T H (\dot{x}_{P2} - FD_0) \\ \mathbf{0} \end{bmatrix} = \Omega_4 \tag{A.26}$$

Combining these terms we have

$$\frac{\partial x}{\partial s} = G^{-1} \left(\frac{d}{dt} \frac{\partial L}{\partial \dot{x}_s} - \frac{\partial L}{\partial x_s} \right) = \Omega_1 - (\Omega_2 - \Omega_3 + \Omega_4) = \Omega \left(x, \dot{x}, \ddot{x}, k \right) \tag{A.27}$$

□

A.2 Proof of Theorem 6

Proof. Let $J_{\Xi_i}(s)$ be the Jacobian of $\frac{d\xi_i}{ds}(s)$ with respect to $\xi_i(s)$ given by:

$$J_{\Xi_i} = \frac{d\left(\frac{d\xi_i}{ds}(s)\right)}{d\xi_i(s)} = \frac{d\Omega(\xi_i(s), [D\xi]_i(s), [D^2\xi]_i(s), k)}{d\xi_i(s)}. \quad (\text{A.28})$$

Applying the chain rule for multivariable functions we obtain

$$J_{\Xi_i} = \frac{d\Omega}{d\xi_i} \frac{d\xi_i}{d\xi_i} + \frac{d\Omega}{d[D\xi]_i} \frac{d[D\xi]_i}{d\xi_i} + \frac{d\Omega}{d[D^2\xi]_i} \frac{d[D^2\xi]_i}{d\xi_i} \quad (\text{A.29})$$

From Theorem 5 we have that

$$\frac{\partial x}{\partial s} = \Omega\left(x, \dot{x}, \ddot{x}, k\right) = \Omega_1 - (\Omega_2 - \Omega_3 + \Omega_4), \quad (\text{A.30})$$

And recall that $\xi_i(s) = x^T(t_i, s)$. Using these two results, we have that

$$\frac{d\Omega}{d\xi_i} = \frac{d\Omega}{dx(t_i, s)} = \frac{d\Omega_1}{dx} - \left(\frac{d\Omega_2}{dx} - \frac{d\Omega_3}{dx} + \frac{d\Omega_4}{dx}\right) \quad (\text{A.31})$$

Computing the derivatives of each of the Ω terms yields the following:

$$\frac{d\Omega_1}{dx} = 2 \begin{bmatrix} 0_{N \times N} & 0_{N \times N} \\ \frac{\partial(H^T H)^{-1}}{\partial x_{P1}} \gamma + (H^T H)^{-1} \frac{\partial \gamma}{\partial x_{P1}} & (H^T H)^{-1} \frac{\partial \gamma}{\partial x_{P2}} \end{bmatrix} \quad (\text{A.32})$$

$$\frac{d\Omega_2}{dx} = \begin{bmatrix} \frac{1}{k} I_{N \times N} \frac{\partial \alpha_1}{\partial x_{P1}} & \frac{1}{k} I_{N \times N} \frac{\partial \alpha_1}{\partial x_{P2}} \\ \frac{\partial(H^T H)^{-1}}{\partial x_{P1}} \alpha_2 + (H^T H)^{-1} \frac{\partial \alpha_2}{\partial x_{P1}} & (H^T H)^{-1} \frac{\partial \alpha_2}{\partial x_{P2}} \end{bmatrix} \quad (\text{A.33})$$

$$\frac{d\Omega_3}{dx} = 2k \begin{bmatrix} 0_{N \times N} & 0_{N \times N} \\ \frac{\partial(H^T H)^{-1}}{\partial x_{P1}} (\dot{x}_{P1} - x_{P2}) & -(H^T H)^{-1} I_{N \times N} \end{bmatrix} \quad (\text{A.34})$$

$$\frac{d\Omega_4}{dx} = \begin{bmatrix} \frac{2}{k} I_{N \times N} \frac{\partial \Gamma}{\partial x_{P1}} & \frac{2}{k} I_{N \times N} \frac{\partial \Gamma}{\partial x_{P2}} \\ 0_{N \times N} & 0_{N \times N} \end{bmatrix} \quad (\text{A.35})$$

where,

$$\gamma = (\dot{H}^T H + H^T \dot{H}) \dot{x}_{P2} + H^T H \ddot{x}_{P2} + \dot{H}^T C + H^T \dot{C} \quad (\text{A.36})$$

$$\alpha_1 = -\frac{\partial F D_0^T}{\partial x_{P1}} H^T \left(2C + 2H \dot{x}_{P2} \right) \quad (\text{A.37})$$

$$\alpha_2 = -\frac{\partial F D_0^T}{\partial x_{P2}} H^T \left(2C + 2H \dot{x}_{P2} \right) \quad (\text{A.38})$$

$$\Gamma = (\dot{x}_{P2} - F D_0)^T \frac{\partial H^T}{\partial x_{P1}} H (\dot{x}_{P2} - F D_0) \quad (\text{A.39})$$

$$\begin{aligned} \frac{\partial \gamma}{\partial x_{P1}} &= \frac{\partial \dot{H}^T}{\partial x_{P1}} \left(H \dot{x}_{P2} + C \right) + \dot{H}^T \left(\frac{\partial H}{\partial x_{P1}} \dot{x}_{P2} + \frac{\partial C}{\partial x_{P1}} \right) + \frac{\partial H^T}{\partial x_{P1}} \left(\dot{H} \dot{x}_{P2} + H \ddot{x}_{P2} + \dot{C} \right) \\ &\quad + H^T \left(\frac{\partial \dot{H}}{\partial x_{P1}} \dot{x}_{P2} + \frac{\partial H}{\partial x_{P1}} \ddot{x}_{P2} + \frac{\partial \dot{C}}{\partial x_{P1}} \right) \end{aligned} \quad (\text{A.40})$$

$$\frac{\partial \gamma}{\partial x_{P2}} = \dot{H}^T \frac{\partial C}{\partial x_{P2}} + H^T \frac{\partial \dot{C}}{\partial x_{P2}} \quad (\text{A.41})$$

$$\begin{aligned} \frac{\partial \alpha_1}{\partial x_{P1}} &= \frac{d}{dx_{P1}} \left(-\frac{\partial F D_0^T}{\partial x_{P1}} \right) H^T \left(2C + 2H \dot{x}_{P2} \right) \\ &\quad - \frac{\partial F D_0^T}{\partial x_{P1}} \left(\frac{\partial H^T}{\partial x_{P1}} \left(2C + 2H \dot{x}_{P2} \right) + H \left(2 \frac{\partial C}{\partial x_{P1}} + 2 \frac{\partial H}{\partial x_{P1}} \dot{x}_{P2} \right) \right) \end{aligned} \quad (\text{A.42})$$

$$\begin{aligned} \frac{\partial \alpha_2}{\partial x_{P1}} &= \frac{d}{dx_{P1}} \left(-\frac{\partial F D_0^T}{\partial x_{P2}} \right) H^T \left(2C + 2H \dot{x}_{P2} \right) \\ &\quad - \frac{\partial F D_0^T}{\partial x_{P2}} \left(\frac{\partial H^T}{\partial x_{P1}} \left(2C + 2H \dot{x}_{P2} \right) + H \left(2 \frac{\partial C}{\partial x_{P1}} + 2 \frac{\partial H}{\partial x_{P1}} \dot{x}_{P2} \right) \right) \end{aligned} \quad (\text{A.43})$$

$$\frac{\partial \alpha_1}{\partial x_{P2}} = \frac{d}{dx_{P2}} \left(-\frac{\partial F D_0^T}{\partial x_{P1}} \right) H^T \left(2C + 2H \dot{x}_{P2} \right) - \frac{\partial F D_0^T}{\partial x_{P1}} 2H \frac{\partial C}{\partial x_{P2}} \quad (\text{A.44})$$

$$\frac{\partial \alpha_2}{\partial x_{P2}} = \frac{d}{dx_{P2}} \left(-\frac{\partial F D_0^T}{\partial x_{P2}} \right) H^T \left(2C + 2H \dot{x}_{P2} \right) - \frac{\partial F D_0^T}{\partial x_{P2}} 2H \frac{\partial C}{\partial x_{P2}} \quad (\text{A.45})$$

$$\begin{aligned} \frac{\partial \Gamma}{\partial x_{P1}} = & \left(-\frac{\partial F D_0^T}{\partial x_{P1}} \frac{\partial H}{\partial x_{P1}} + (\dot{x}_{P2} - F D_0)^T \frac{\partial^2 H}{\partial x_{P1}^2} \right) H (\dot{x}_{P2} - F D_0) \\ & + (\dot{x}_{P2} - F D_0)^T \frac{\partial H}{\partial x_{P1}} \left(\frac{\partial H}{\partial x_{P1}} (\dot{x}_{P2} - F D_0) - H \frac{\partial F D_0}{\partial x_{P1}} \right) \end{aligned} \quad (\text{A.46})$$

$$\frac{\partial \Gamma}{\partial x_{P2}} = -\frac{\partial F D_0^T}{\partial x_{P2}} \frac{\partial H}{\partial x_{P1}} H (\dot{x}_{P2} - F D_0) + (\dot{x}_{P2} - F D_0)^T \frac{\partial H}{\partial x_{P1}} \left(-H \frac{\partial F D_0}{\partial x_{P2}} \right) \quad (\text{A.47})$$

The derivative of Ω wrt $[D\xi]_i$ similarly is given by

$$\frac{d\Omega}{d[D\xi]_i} = \frac{d\Omega_1}{d\dot{x}} - \left(\frac{d\Omega_2}{d\dot{x}} - \frac{d\Omega_3}{d\dot{x}} + \frac{d\Omega_4}{d\dot{x}} \right) \quad (\text{A.48})$$

$$\frac{d\Omega_1}{d\dot{x}} = 2 \begin{bmatrix} 0_{N \times N} & -I_{N \times N} \\ (H^T H)^{-1} \frac{\partial \gamma}{\partial \dot{x}_{P1}} & (H^T H)^{-1} \frac{\partial \gamma}{\partial \dot{x}_{P2}} \end{bmatrix} \quad (\text{A.49})$$

$$\frac{d\Omega_2}{d\dot{x}} = \begin{bmatrix} 0_{N \times N} & \frac{1}{k} I_{N \times N} \frac{\partial \alpha_1}{\partial \dot{x}_{P2}} \\ 0_{N \times N} & (H^T H)^{-1} \frac{\partial \alpha_2}{\partial \dot{x}_{P2}} \end{bmatrix} \quad (\text{A.50})$$

$$\frac{d\Omega_3}{d\dot{x}} = \begin{bmatrix} 0_{N \times N} & 0_{N \times N} \\ 2k(H^T H)^{-1} I_{N \times N} & 0_{N \times N} \end{bmatrix} \quad (\text{A.51})$$

$$\frac{d\Omega_4}{d\dot{x}} = \begin{bmatrix} 0_{N \times N} & \frac{2}{k} I_{N \times N} \frac{\partial \Gamma}{\partial \dot{x}_{P2}} \\ 0_{N \times N} & 0_{N \times N} \end{bmatrix} \quad (\text{A.52})$$

where

$$\frac{\partial \alpha_1}{\partial \dot{x}_{P2}} = -\frac{\partial F D_0^T}{\partial x_{P1}} 2H^T H \quad (\text{A.53})$$

$$\frac{\partial \alpha_2}{\partial \dot{x}_{P2}} = -\frac{\partial F D_0^T}{\partial x_{P2}} 2H^T H \quad (\text{A.54})$$

$$\frac{\partial \gamma}{\partial \dot{x}_{P1}} = \frac{\partial \dot{H}^T}{\partial \dot{x}_{P1}} \left(H \dot{x}_{P2} + C \right) + H^T \left(\frac{\partial \dot{H}}{\partial \dot{x}_{P1}} \dot{x}_{P2} + \frac{\partial \dot{C}}{\partial \dot{x}_{P1}} \right) \quad (\text{A.55})$$

$$\frac{\partial \gamma}{\partial \dot{x}_{P2}} = (\dot{H}^T H + H^T \dot{H}) + H^T \frac{\partial \dot{C}}{\partial \dot{x}_{P2}} \quad (\text{A.56})$$

$$\frac{\partial \Gamma}{\partial \dot{x}_{P2}} = I_{N \times N} \frac{\partial H}{\partial x_{P1}} H (\dot{x}_{P2} - F D_0) + (\dot{x}_{P2} - F D_0)^T \frac{\partial H}{\partial x_{P1}} H \quad (\text{A.57})$$

$\frac{\partial \gamma}{\partial \dot{x}_{P1}}$ (A.55) and $\frac{\partial \gamma}{\partial \dot{x}_{P2}}$ (A.56) can be simplified even further. Next, we highlight the following relations that will be used to simplify these terms. First, $\dot{H}$ can be computed using the chain rule in the following way:

$$\dot{H} = \frac{\partial H}{\partial x_{P1}} \frac{\partial x_{P1}}{\partial t} = \frac{\partial H}{\partial x_{P1}} \dot{x}_{P1} \quad (\text{A.58})$$

Second, $\dot{C}$ can also be computed similarly using the chain rule:

$$\dot{C} = \frac{\partial C}{\partial x_{P1}} \dot{x}_{P1} + \frac{\partial C}{\partial x_{P2}} \dot{x}_{P2} \quad (\text{A.59})$$

Lastly, recall that H is symmetric, therefore:

$$\frac{\partial H}{\partial x_{P1}} = \frac{\partial H^T}{\partial x_{P1}} \quad (\text{A.60})$$

Using these three relations we can obtain the following:

$$\frac{\partial \dot{H}}{\partial \dot{x}_{P1}} = \frac{\partial H}{\partial x_{P1}} \quad (\text{A.61})$$

$$\frac{\partial \dot{C}}{\partial \dot{x}_{P1}} = \frac{\partial C}{\partial x_{P1}} \quad (\text{A.62})$$

Using (A.61) and (A.62) to simplify $\frac{\partial \gamma}{\partial \dot{x}_{P1}}$ (A.55) yields:

$$\frac{\partial \gamma}{\partial \dot{x}_{P1}} = \frac{\partial H}{\partial x_{P1}} \left(H \dot{x}_{P2} + C \right) + H \left(\frac{\partial H}{\partial x_{P1}} \dot{x}_{P2} + \frac{\partial C}{\partial x_{P1}} \right) \quad (\text{A.63})$$

we can apply similar line of reasoning to obtain a simplification for $\frac{\partial \gamma}{\partial \dot{x}_{P2}}$ (A.56)

$$\frac{\partial \gamma}{\partial \dot{x}_{P2}} = \dot{H} H + H \left(\dot{H} + \frac{\partial C}{\partial x_{P2}} \right) \quad (\text{A.64})$$

Lastly, the derivative of Ω wrt $[D^2 \xi]_i$ is given by

$$\frac{d\Omega}{d[D^2 \xi]_i} = 2 \begin{bmatrix} I_{N \times N} & 0_{N \times N} \\ 0_{N \times N} & (H^T H)^{-1} \frac{\partial \gamma}{\partial \ddot{x}_{P2}} \end{bmatrix} \quad (\text{A.65})$$

where

$$\frac{\partial \gamma}{\partial \ddot{x}_{P2}} = H^T H \quad (\text{A.66})$$

This simplifies to

$$\frac{d\Omega}{d[D^2 \xi]_i} = 2 \begin{bmatrix} I_{N \times N} & 0_{N \times N} \\ 0_{N \times N} & I_{N \times N} \end{bmatrix} \quad (\text{A.67})$$

Observing the terms from Equations (A.36) – (A.47), we have that $\frac{d\Omega}{d\xi_i}$ (A.31) is a function of the rigid body dynamics and its higher-order derivatives. Namely,

$$\frac{d\Omega}{d\xi_i} \left(H, \dot{H}, C, \dot{C}, FD_0, \frac{\partial H}{\partial x_{P1}}, \frac{\partial^2 H}{\partial x_{P1}^2}, \frac{\partial \dot{H}}{\partial x_{P1}}, \frac{\partial FD_0}{\partial x_{P1}}, \frac{\partial FD_0}{\partial x_{P2}}, \right. \\ \left. \frac{\partial C}{\partial x_{P1}}, \frac{\partial C}{\partial x_{P2}}, \frac{\partial \dot{C}}{\partial x_{P1}}, \frac{\partial \dot{C}}{\partial x_{P2}}, \frac{\partial^2 FD_0}{\partial x_{P1}^2}, \frac{\partial^2 FD_0}{\partial x_{P2}^2}, \frac{\partial^2 FD_0}{\partial x_{P1} \partial x_{P2}} \right)$$

Similarly, observing the terms from Equations (A.53), (A.54), (A.57), (A.63) and (A.64), we have that $\frac{d\Omega}{d[D\xi]_i}$ (A.48) is a function of the rigid body dynamics and its first-order derivatives. Specifically,

$$\frac{d\Omega}{d[D\xi]_i} \left(H, C, \dot{H}, FD_0, \frac{\partial H}{\partial x_{P1}}, \frac{\partial FD_0}{\partial x_{P1}}, \frac{\partial FD_0}{\partial x_{P2}}, \frac{\partial C}{\partial x_{P1}}, \frac{\partial C}{\partial x_{P2}} \right)$$

and from (A.67), we have

$$\frac{d\Omega}{d[D^2\xi]_i} = 2 \begin{bmatrix} I_{N \times N} & 0_{N \times N} \\ 0_{N \times N} & I_{N \times N} \end{bmatrix}$$

□

A.3 PHLAME Results Appendix

This section contains all the varied parameters that we grid search over and use for the experiments in Section 2.6:

Parameter	Grid values
w_u	$[10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}]$
w_x	$[10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 0.0, 1, 10]$
w_{xf}	$[10^{-4}, 1.0, 10, 1000]$
δ_t	$[10^{-2}, 10^{-3}, 10^{-4}]$
<i>Initial guess</i>	[Zeros, Line and RNEA, Constant]

Table A.1: Parameter Values for Crocoddyl unconstrained

Parameter	Grid values
w_u	$[10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}]$
w_x	$[10^{-6}, 10^{-4}, 10^{-3}, 0.0, 1, 100, 1000]$
w_{xf}	$[10^{-6}, 10^{-4}, 1.0, 10, 1000]$
δ_t	$[10^{-2}, 10^{-3}]$
ϵ_{tol}	$[10^{-3}, 10^{-4}, 10^{-7}]$
μ_{init}	$[10^{-2}, 10^{-7}, 10^{-8}]$
<i>Initial guess</i>	[Zeros, Line and RNEA, Rollout and Zero, Rollout and Constant]

Table A.2: Parameter Values for Aligator with obstacles

Parameter	Grid values
s_{max}	$[10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 5, 10, 25, 50, 100]$
k	$[10^3, 10^4, 10^5, 10^6, 10^7, 10^8, 10^9, 10^{10}]$
p	$[5, 6, 7, 8, 9, 10, 15]$
<i>Initial guess</i>	[Line]

Table A.3: Parameter Values for PHLAME for pendulum swingup

Parameter	Grid values
s_{max}	$[10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 25, 50, 100]$
k	$[10^4, 10^5, 10^6, 10^7, 10^8, 10^9, 10^{10}]$
p	$[5, 6, 7, 8, 9, 10, 15]$
<i>Initial guess</i>	[Line]

Table A.4: Parameter Values for PHLAME for Kinova without obstacles

Parameter	Grid values
s_{max}	$[10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 25, 50, 100]$
k	$[10^3, 10^4, 10^5, 10^6, 10^7, 10^8, 10^9, 10^{10}]$
p	$[5, 6, 7, 8, 9, 10]$
<i>Initial guess</i>	[Line]

Table A.5: Parameter Values for PHLAME for Digit without obstacles

Parameter	Grid values
s_{max}	$[10^{-4}, 10^{-2}, 10^{-1}, 1, 5, 10, 25, 50, 100]$
k	$[10^5, 10^6, 10^7, 10^8, 10^9, 10^{10}]$
p	$[5, 6, 7, 8, 9, 10, 15]$
c_{cons}	$[1, 50, 200]$
k_{cons}	$[10^5, 10^6, 10^7, 10^8, 10^9, 10^{10}]$
<i>Initial guess</i>	[Line]

Table A.6: Parameter Values for PHLAME Kinova with Constraints

Parameter	Grid values
s_{max}	$[10^{-4}, 10^{-2}, 10^{-1}, 1, 5, 10, 25, 50, 100]$
k	$[10^5, 10^6, 10^7, 10^8, 10^9, 10^{10}]$
p	$[5, 6, 7, 8, 9, 10, 15]$
c_{cons}	$[1, 50, 200]$
k_{cons}	$[10^5, 10^6, 10^7, 10^8, 10^9, 10^{10}]$
<i>Initial guess</i>	[Line]

Table A.7: Parameter Values for PHLAME for Digit with Constraints

APPENDIX B

BLAZE Appendices

B.1 Proof of Lemma 12

Proof. To prove this result, we construct the variation of the Action Function with respect to s using Taylor Expansion and Integration by Parts. This enables us to leverage the definition of the AGHF PDE to prove the desired result.

Consider the variation of (3.5):

$$\mathcal{A}(x_{s+\delta}) = \int_0^T L(x(t, s + \delta), \dot{x}(t, s + \delta)) dt. \quad (\text{B.1})$$

The Taylor Expansion of the integrand of (3.5) is:

$$\begin{aligned} L(x(t, s + \delta), \dot{x}(t, s + \delta)) &= L(x_s(t), \dot{x}_s(t)) + \delta \frac{\partial L}{\partial x_s} \frac{\partial x_s}{\partial s} \\ &\quad + \delta \frac{\partial L}{\partial \dot{x}_s} \frac{\partial \dot{x}_s}{\partial s} + o(\delta) \end{aligned} \quad (\text{B.2})$$

where for convenience we have dropped the arguments in the terms on the right hand side. Substituting (B.2) into (B.1)

$$\mathcal{A}(x_{s+\delta}) = \mathcal{A}(x_s) + \delta \int_0^T \left(\frac{\partial L}{\partial x_s} \frac{\partial x_s}{\partial s} + \frac{\partial L}{\partial \dot{x}_s} \frac{\partial \dot{x}_s}{\partial s} + o(\delta) \right) dt. \quad (\text{B.3})$$

Integrating (B.3) by parts and applying the boundary conditions (3.7), we obtain:

$$\begin{aligned} \mathcal{A}(x_{s+\delta}) &= \mathcal{A}(x_s) + \delta \int_0^T \frac{\partial x_s}{\partial s} \left(\frac{\partial L}{\partial x_s}(x_s(t), \dot{x}_s(t)) \right. \\ &\quad \left. - \frac{d}{dt} \frac{\partial L}{\partial \dot{x}_s}(x_s(t), \dot{x}_s(t)) \right) dt + o(\delta). \end{aligned}$$

Rearranging the terms, we obtain

$$\begin{aligned} \frac{\mathcal{A}(x_{s+\delta}) - \mathcal{A}(x_s)}{\delta} &= \int_0^T \frac{\partial x_s}{\partial s} \left(\frac{\partial L}{\partial x_s}(x_s(t), \dot{x}_s(t)) \right. \\ &\quad \left. - \frac{d}{dt} \frac{\partial L}{\partial \dot{x}_s}(x_s(t), \dot{x}_s(t)) \right) dt + \frac{o(\delta)}{\delta} \end{aligned} \quad (\text{B.4})$$

Treating $\Delta s = \delta$, we take $\Delta s \rightarrow 0$ to obtain

$$\frac{\partial \mathcal{A}}{\partial s} = \int_0^T \frac{\partial x_s}{\partial s} \left(\frac{\partial L}{\partial x_s}(x_s(t), \dot{x}_s(t)) - \frac{d}{dt} \frac{\partial L}{\partial \dot{x}_s}(x_s(t), \dot{x}_s(t)) \right) dt \quad (\text{B.5})$$

Substituting (3.6) into (B.5)

$$\frac{\partial \mathcal{A}}{\partial s} = - \int_0^T \frac{\partial x_s^T}{\partial s} M \frac{\partial x_s}{\partial s}. \quad (\text{B.6})$$

Thus we see that

$$\frac{\partial \mathcal{A}}{\partial s} \leq 0 \iff M \succeq 0. \quad (\text{B.7})$$

□

B.2 Proof of Theorem 15

Before proving this result, we prove the following result that is used in the proof of the result:

Lemma 54. *Suppose the Cost Function in the definition of the Action Functional (3.11) is coercive with constants α and β . Then for any $C \in \mathbb{R}$, we have $\{x \in C^1([0, T] \rightarrow \mathbb{R}^{2N}) \mid \mathcal{A}(x) \leq C\} \subseteq \{x \in C^1([0, T] \rightarrow \mathbb{R}^{2N}) \mid \|\dot{x}\|_{L^\infty}^2 + \|x\|_{L^\infty}^2 \leq \frac{\beta + CT}{\alpha}\}$*

Proof. We prove this result by contradiction. We first assume that x or $\dot{X}$ are not essentially bounded. By utilizing the definition of coercivity, we can lower bound the integrand with something that grows arbitrarily large which leads to a contradiction.

Consider $x \in \{x \in C^1([0, T] \rightarrow \mathbb{R}^{2N}) \mid \mathcal{A}(x) \leq C\}$ such that either $x \notin L^\infty$ or $\dot{x} \notin L^\infty$. Without loss of generality assume that $x \notin L^\infty$. This implies that for every large $M \geq 0$ there is a set of positive measure $I \subset [0, T]$, where $\|x\|_{L^\infty} \geq M$, which implies that through coercivity that

$$k_d \|\dot{x}_{P1}(t) - x_{P2}(t)\|_2^2 + c(x(t), \dot{x}(t), u(t)) \geq \alpha M - \beta, \quad (\text{B.8})$$

for $t \in I$. Integrating the Lagrangian over I gives:

$$\int_I (k_d \|\dot{x}_{P1}(t) - x_{P2}(t)\|_2^2 + c(x(t), \dot{x}(t), u(t))) dt \geq (\alpha M - \beta) \lambda(I), \quad (\text{B.9})$$

where $\lambda(I) > 0$ describes the size of I . Because M can be made arbitrarily large, this leads to a contradiction. To construct the bound, note that the largest violation would occur if $I = T$. Using this observation with the previous argument provides the desired result. $\square$

Now we use the previous lemma to prove Theorem 15:

Proof. To prove this result, we begin by defining a sublevel set of the Action Functional that we prove is invariant under the AGHF Flow. Next, we compute the norm of the error between a trajectory belonging to this invariant set and the integrated trajectory. By applying the Bellman-Gronwall Inequality and the fact that follows from Lemma 54 we are able to prove our result.

Because the path planning problem is feasible, there exists x^* that satisfies the dynamics (3.2). Plug x^* into the Action Functional and pick some C_1 such that $C_1 > \mathcal{A}(x^*)$. Note C_1 is independent of k_d . Let $X = \{x \in AC([0, T] \rightarrow \mathbb{R}^{2N}) \mid x(0) = \mathbf{x}_0, x(T) = \mathbf{x}_f\}$ where AC denotes the space of absolutely continuous functions. Let $\Omega_{k_d}^{AC} = \{x \in X \mid \mathcal{A}(x) < C_1\}$ which is not empty because it at least contains x^* . Because $\mathcal{A}$ is continuous over X with respect to the norm on absolutely continuous functions, $\Omega_{k_d}^{AC}$ is open. Because $C^1([0, T] \rightarrow \mathbb{R}^{2N})$ is dense in $AC([0, T] \rightarrow \mathbb{R}^{2N})$, $\Omega'_{k_d} := \Omega_{k_d}^{AC} \cap C^1([0, T] \rightarrow \mathbb{R}^{2N})$ is open as well. From Lemma 12, we know that $\mathcal{A}$ is non-increasing so Ω'_{k_d} is invariant. Let Ω_{k_d} be the region of attraction to Ω'_{k_d} ; that is $\Omega'_{k_d} \subset \Omega_{k_d} \subset C^1([0, T] \rightarrow \mathbb{R}^{2N})$ and all AGHF solutions x_s with an initial condition starting from Ω_{k_d} will converge to the invariant set Ω'_{k_d} when s increases. Consequently, when s_{max} is sufficiently large, $\mathcal{A}(x_{s_{max}}) \leq C_1$ and $x_{s_{max}}$ is continuous.

Before moving on to compute the error trajectory, we make one observation.

Define the curve $x(t) = x_{s_{max}}(t)$ and $\bar{u}(t) \in \mathbb{R}^{2N}$ be given by:

$$\bar{u}(t) = \begin{bmatrix} u_c(t) \\ u(t) \end{bmatrix} = \bar{F}(x(t))^{-1}(\dot{x}(t) - F_d(x(t))). \quad (\text{B.10})$$

Substituting (B.10) into (3.11)

$$C_1 \geq \int_0^T (k_d \|u_c(t)\|_2^2 + c(x(t), \dot{x}(t), u(t))) dt. \quad (\text{B.11})$$

From which we conclude

$$\int_0^T \|u_c(t)\|_2^2 dt \leq \frac{C_1}{k_d}. \quad (\text{B.12})$$

Next, define the dynamics of the integrated system:

$$\begin{aligned} \tilde{x}(t) &= F_d(\tilde{x}(t)) + F(\tilde{x}(t))u(t), \\ \tilde{x}(0) &= \mathbf{x}_0. \end{aligned} \quad (\text{B.13})$$

Define the error $e(t) = x(t) - \tilde{x}(t)$ and then the error dynamics are

$$\begin{aligned} \dot{e}(t) &= \dot{x}(t) - \dot{\tilde{x}}(t) \\ &= (F_d(x(t)) - F_d(\tilde{x}(t))) \\ &\quad + (F(x(t)) - F(\tilde{x}(t)))u(t) \\ &\quad + F_c(x(t))u_c(t). \end{aligned} \quad (\text{B.14})$$

Integrating both sides

$$\begin{aligned} \|e(t)\|_2^2 &= \int_0^t \left((F_d(x(\tau)) - F_d(\tilde{x}(\tau))) \right. \\ &\quad \left. + (F(x(\tau)) - F(\tilde{x}(\tau)))u(\tau) \right. \\ &\quad \left. + F_c(x(\tau))u_c(\tau) \right) d\tau. \end{aligned} \quad (\text{B.15})$$

Squaring the norm on both sides

$$\begin{aligned} \|e(t)\|_2^2 &= \int_0^t \left(\|F_d(x(\tau)) - F_d(\tilde{x}(\tau))\| \right. \\ &\quad \left. + \|F(x(\tau)) - F(\tilde{x}(\tau))\|u(\tau) \right. \\ &\quad \left. + \|F_c(x(\tau))u_c(\tau)\| \right)^2 d\tau. \end{aligned} \quad (\text{B.16})$$

Applying the Cauchy-Schawrtz Inequality

$$\begin{aligned} \|e(t)\|_2^2 &\leq t \int_0^t \left(\|F_d(x(\tau)) - F_d(\tilde{x}(\tau))\| \right. \\ &\quad \left. + \|F(x(\tau)) - F(\tilde{x}(\tau))\|u(\tau) \right. \\ &\quad \left. + \|F_c(x(\tau))u_c(\tau)\| \right)^2 d\tau. \end{aligned} \quad (\text{B.17})$$

Applying the Triangle Inequality

$$\begin{aligned} \|e(t)\|_2^2 &\leq t \int_0^t \left(\|F_d(x(\tau)) - F_d(\tilde{x}(\tau))\| \right. \\ &\quad \left. + \|(F(x(\tau)) - F(\tilde{x}(\tau)))u(\tau)\| \right. \\ &\quad \left. + \|F_c(x(\tau))u_c(\tau)\| \right)^2 d\tau. \end{aligned} \quad (\text{B.18})$$

Using the Power Mean Inequality

$$\begin{aligned} \|e(t)\|_2^2 &\leq 3t \int_0^t \left(\|F_d(x(\tau)) - F_d(\tilde{x}(\tau))\|_2^2 \right. \\ &\quad \left. + \|(F(x(\tau)) - F(\tilde{x}(\tau)))u(\tau)\|_2^2 \right. \\ &\quad \left. + \|F_c(x(\tau))u_c(\tau)\|_2^2 \right) d\tau. \end{aligned} \quad (\text{B.19})$$

Because F_c is continuous and x is bounded in the L^∞ sense because of Lemma 54, we know there exists a $C_2 \geq 0$ such that almost everywhere:

$$\|F_c(x(\tau))u_c(\tau)\|_2 \leq \|F_c(x(\tau))\|_2 \|u_c(\tau)\|_2 \leq C_2 \|u_c(\tau)\|_2. \quad (\text{B.20})$$

Using this observation, the fact that F_d and F are globally Lipschitz, and applying (B.12):

$$\begin{aligned} \|e(t)\|_2^2 &\leq 3t \int_0^t (Y_1^2 + Y_2^2 \|u(\tau)\|_2^2) \|e(t)\|_2^2 d\tau \\ &\quad + \frac{3tC_1C_2^2}{k_d} \end{aligned} \quad (\text{B.21})$$

Finally, we can apply the Bellman-Grönwall Inequality:

$$\|e(t)\|_2^2 \leq \left(\frac{3tC_1C_2^2}{k_d} \right) \exp \left(3t \int_0^t (Y_1^2 + Y_2^2 \|u(\tau)\|_2^2) d\tau \right) \quad (\text{B.22})$$

To bound the extracted control input u , recall its Definition 13 and note that all functions that define the operation are continuous. Because of the boundedness of x in the L^∞ sense which follows from Lemma 54 one can prove that there exists a C_3 such that

$$\int_0^T \|u(\tau)\|_2^2 d\tau \leq C_3 \quad (\text{B.23})$$

Substituting the inequalities (B.12) and (B.23) into (B.22) we conclude that:

$$\|\tilde{x}(t) - x(t)\|_2 \leq \sqrt{\frac{3tC_1C_2^2}{k_d} \exp\left(3t(Y_1^2t + Y_2^2C_3)\right)} \quad (\text{B.24})$$

and by substituting $t = T$ we obtain the desired conclusion. $\square$

B.3 Proof of Lemma 19

Proof. Recall from (3.8) that $u_s(t)$ is given by:

$$u_s(t) = \begin{bmatrix} 0_{N \times N} & I_{N \times N} \end{bmatrix} \bar{F}(x_s(t))^{-1} (\dot{x}_s(t) - F_d(x_s(t))) \quad (\text{B.25})$$

We begin by substituting (3.3) and (3.4) into (B.25)

$$u_s(t) = \begin{bmatrix} 0_{N \times N} & I_{N \times N} \end{bmatrix} \cdot \underbrace{\begin{bmatrix} I_{N \times (2N-m)} & 0_{N \times m} \\ 0_{N \times (2N-m)} & H(x_{P1}(s, t)) \end{bmatrix}^T}_{(\bar{F}(x_s(t))^{-1})^T} \cdot \underbrace{\begin{bmatrix} \dot{x}_{P1} - x_{P2} \\ \dot{x}_{P2} + H^{-1}(x_{P1}(s, t))C(x_{P1}(s, t), x_{P2}(s, t)) \end{bmatrix}}_{\dot{x} - F_d} \quad (\text{B.26})$$

Expanding and simplifying yields

$$u_s(t) = H(x_{P1}(s, t))\dot{x}_{P2}(s, t) + C(x_{P1}(s, t), x_{P2}(s, t)). \quad (\text{B.27})$$

Note that this equation corresponds exactly to the robot dynamics presented in (3.1) when $B = I$. $\square$

B.4 Experiment Hyperparameters

As pointed out in [3], Aligator and Crocoddyl have several parameters that affect the performance. The parameter that is common to both methods and affects the solve time most is the time discretization Δt . The methods also allow the user to choose a running cost through the choice of weights w_u , w_x . The running cost is the 2-norm of the input summed to the

Parameter	Value
Bezier Polynomial Degree (p)	7
Constraint Evaluation Points (N)	20
IPOPT mu Strategy (<code>mu_strategy</code>)	adaptive
IPOPT Linear Solver (<code>linear_solver</code>)	ma57

Table B.1: Parameter settings for RAPTOR in the swing-up experiments.

2-norm of the state with weights w_u and w_x respectively. The methods allow the user to weigh a terminal cost through the weight w_{xf} . Crocoddyl allows penalizing state inequality constraint by a factor w_{state} and the terminal value by $w_{state,f}$. Finally, Aligator also allows one to specify ϵ_{tol} and μ_{init} which control the convergence properties of the optimization.

Tables B.2, B.3, B.4 summarizes the parameters used by Aligator in the scalability experiment, the sphere obstacle avoidance experiment, and the realistic obstacle avoidance scenario respectively. Table B.5 describes the parameters used by Crocoddyl for the scalability experiment.

RAPTOR optimizes over Bezier polynomial coefficients representing the state trajectory. The primary hyperparameters for RAPTOR include the Bezier polynomial degree p and the number of points N where constraints are evaluated along the trajectory. Additionally, since RAPTOR utilizes IPOPT as its optimizer, several IPOPT-specific hyperparameters can be tuned, such as `linear_solver`, which influences the speed of Newton steps, and `mu_strategy`, which affects convergence speed and stability. For both the swing-up experiments and the obstacle avoidance experiments, we used the same parameter set across all scenarios, following the recommendations provided in RAPTOR’s official GitHub repository. Table B.1 details these parameters.

BLAZE implements a pseudospectral MOL algorithm to solve the AGHF, which is similar to the approach adopted by [3]. As a result, we refer to [3] for a complete exposition of the typical parameters of the AGHF. In this paper we extend the parameters of the constraints k_{cons} and c_{cons} to ones designed per constraint. For instance, c_{cuboid} corresponds to c_{cons} for the cuboid constraints. Similarly, k_{sph} corresponds to the weight on the constraint that encodes collision avoidance from sphere obstacles. The parameters we choose for each experiment for BLAZE are summarized in Tables B.6, B.7, and B.8.

DOF	w_u	w_x	w_{xf}	Δt	ϵ_{tol}	μ_{init}
1	10^{-4}	10^{-3}	10^{-4}	10^{-2}	10^{-2}	10^{-8}
2	10^{-4}	10^{-2}	10^{-2}	10^{-2}	10^{-2}	10^{-7}
3	10^{-3}	1	1	10^{-2}	10^{-2}	10^{-7}
4	10^{-3}	1	10^{-4}	10^{-2}	10^{-2}	10^{-7}
5	10^{-3}	1	10^{-6}	10^{-2}	10^{-2}	10^{-7}
7	10^{-2}	1	1	10^{-2}	10^{-3}	10^{-7}
14	10^{-2}	1	1	10^{-2}	10^{-3}	10^{-7}
21	10^{-2}	1	1	10^{-2}	10^{-3}	10^{-7}

Table B.2: Parameter settings for Aligator for the Scalability Experiment

DOF	w_u	w_x	w_{xf}	Δt	ϵ_{tol}	μ_{init}
7	10^{-2}	1	1	10^{-2}	10^{-3}	10^{-7}

Table B.3: Parameter settings for Aligator for the Sphere Obstacle Avoidance experiment

DOF	w_u	w_x	w_{xf}	w_{state}	$w_{state,f}$	Δt
1	10^{-3}	10^{-2}	10	10	10	10^{-2}
2	10^{-4}	10^{-4}	10^3	10	10	10^{-2}
3	10^{-4}	10^{-3}	1	10	10	10^{-2}
4	10^{-3}	10^{-2}	10	10	10	10^{-3}
5	10^{-4}	10^{-2}	10	10	10	10^{-3}
7	10^{-4}	10^{-2}	10^2	10	10^{-1}	10^{-3}
14	10^{-3}	10^{-2}	10^2	1	10^{-1}	10^{-3}
21	10^{-3}	10^{-2}	10^2	1	10^{-1}	10^{-3}

Table B.5: Parameter settings for Crocoddyl for Scalability experiment

DOF	w_u	w_x	w_{xf}	Δt	ϵ_{tol}	μ_{init}
7	10^{-2}	1	1	10^{-2}	10^{-3}	10^{-4}

Table B.4: Parameter settings for Aligator for the 2s Task-Based scenario experiment

DOF	k	c_{state}	k_{state}	c_{input}	k_{input}	s_{max}
1	10^9	50	10^5	50	10^5	0.01
2	10^{10}	50	10^5	50	10^5	0.01
3	10^6	50	10^{10}	50	10^{10}	1
4	10^6	50	10^7	50	10^7	1
5	10^6	1	10^5	1	10^5	1
7	10^6	200	10^9	200	10^9	100
14	10^{14}	200	10^6	200	10^6	10
21	10^{14}	200	10^6	200	10^6	10

Table B.6: Parameter settings for BLAZE Scalability Experiment

Parameter	Phase 1	Phase 2
s_{max}	10	10
k	10^4	10^4
p	7	7
c_{sph}	100	100
c_{state}	100	100
c_{input}	100	100
k_{sph}	10^7	10^7
k_{state}	10^6	10^6
k_{input}	10^6	10^6

Table B.7: Parameter settings for BLAZE for Sphere Obstacle avoidance experiment.

Parameter	Phase 1	Phase 2
s_{max}	500	500
k	10^3	10^5
p	7	7
c_{sph}	150	-
c_{cuboid}	-	250
c_{state}	150	250
c_{input}	150	250
k_{sph}	10^6	-
k_{cuboid}	150	10^4
k_{state}	10^5	10^3
k_{input}	10^5	10^3

APPENDIX C

RADIUS Appendices

C.1 Proof of Lemma 32

Proof. To relax the integrand $q_{i,j}$, we start by computing the 2nd-order Taylor Expansion of $q_{i,j}$ centered at $w_{\mathcal{T}}^{\text{ctr}} + A_j p$ and applying Mean Value Theorem (MVT) [92, Theorem 4.1] to eliminate higher order terms in the Taylor Expansion as:

$$\begin{aligned} q_{i,j}(w) = & q_{i,j}(w_{\mathcal{S}}^{\text{ctr}} + A_j p) + \frac{\partial q_{i,j}}{\partial w}(w_{\mathcal{S}}^{\text{ctr}} + A_j p) \cdot \\ & \cdot (w - w_{\mathcal{S}}^{\text{ctr}} - A_j p) + \frac{1}{2}(w - w_{\mathcal{S}}^{\text{ctr}} - A_j p)^{\top} \cdot \\ & \cdot \text{Hess}_{q_{i,j}}(w') \cdot (w - w_{\mathcal{S}}^{\text{ctr}} - A_j p) \end{aligned} \quad (\text{C.1})$$

where $w' \in \mathcal{S} + A_j p$ is some point on the line segment joining points $w_{\mathcal{S}}^{\text{ctr}} + A_j p$ and w in $\mathcal{S} + A_j p$, and $\text{Hess}_{q_{i,j}}$ gives the Hessian of $q_{i,j}$. Because MVT does not provide w' in a closed-form, we then drop the dependency of w' in (C.1) by bounding the Hessian of $q_{i,j}$ as a matrix $H_{\mathcal{S}} \in \mathbb{R}^{2 \times 2}$ where $H_{\mathcal{S}}$ is generated by taking element-wise supremum of $\text{Hess}_{q_{i,j}}$ over $\mathcal{S} \oplus A_j \mathcal{P}$ using Interval Arithmetic [44]. Note by construction of $\mathcal{S}$ and $w_{\mathcal{S}}^{\text{ctr}}$, it is guaranteed that either $w \geq w_{\mathcal{S}}^{\text{ctr}} + A_j p$ for all $w \in \mathcal{S} + A_j p$ or $w \leq w_{\mathcal{S}}^{\text{ctr}} + A_j p$ for all $w \in \mathcal{S} + A_j p$, thus by definition of $H_{\mathcal{S}}$ the desired inequality holds for all $w \in \mathcal{S} + A_j p$. $\square$

C.2 Additional Details

Specifications of the full-size FWD vehicle and the Rover robot used in this work can be found in [70, Section IX], from which we also adapt control gains and zonotope reachable sets.

C.2.1 Desired Trajectories

In this work we select 4 families of desired trajectories for driving maneuvers often observed during daily driving: speed changes, direction changes, lane changes and left turning. Each desired trajectory is the concatenation of a driving maneuver and a contingency braking maneuver. Note that the duration t_m of driving maneuvers remains constant within each trajectory family, but can vary among different trajectory families. Each desired trajectory is parameterized by $p = [p_u, p_y]^\top \in \mathcal{P} \subset \mathbb{R}^2$ where p_u and p_y decide desired longitudinal speed and lateral displacement respectively. The trajectory families associated with speed change, direction change and lane change are detailed in [70, Section IX-A], thus we describe desired trajectories that achieve left turning here.

To achieve a left turning maneuver, we set the desired trajectory for longitudinal speed as

$$u^{\text{des}}(t, p) = \begin{cases} \frac{11}{2}t, & \text{if } 0 \leq t < \frac{1}{4}t_m \text{ and } \frac{11}{8}t_m < p_u \\ p_u, & \text{if } 0 \leq t < \frac{1}{4}t_m \text{ and } \frac{11}{8}t_m \geq p_u \\ p_u, & \text{if } \frac{1}{4}t_m \leq t < t_m \\ u^{\text{brake}}(t, p), & \text{if } t \geq t_m \end{cases} \quad (\text{C.2})$$

where $u^{\text{brake}} : [t_m, t_f] \times \mathcal{P} \rightarrow \mathbb{R}$ describes the desired trajectory of longitudinal speed during contingency braking and shares the same formulation with the other 3 trajectory families.

The desired trajectory of the yaw rate is as follows:

$$r^{\text{des}}(t, p) = \begin{cases} \frac{1}{2}p_y \left(1 - \cos \left(\frac{4\pi}{t_m}t \right) \right), & \text{if } 0 \leq t < \frac{1}{4}t_m \\ p_y, & \text{if } \frac{1}{4}t_m \leq t < \frac{3}{4}t_m \\ \frac{1}{2}p_y \left(1 - \cos \left(\frac{4\pi}{t_m}t \right) \right), & \text{if } \frac{3}{4}t_m \leq t < t_m \\ 0, & \text{if } t \geq t_m \end{cases} \quad (\text{C.3})$$

And desired trajectory of heading is set as $h^{\text{des}}(t, p) = h_0 + \int_0^t r^{\text{des}}(\tau, p) d\tau$ for any $t \in [0, t_f]$ and $p \in \mathcal{P}$, where $h_0 \in [-\pi, \pi]$ is the initial heading of the ego vehicle at time 0.

The duration t_m of driving maneuvers for every trajectory family is 3[s] for speed change, 3[s] for direction change, 6[s] for lane change, and 4[s] for left change. Because we do not know which desired trajectory ensures not-at-fault *a priori*, to guarantee real-time performance, t_{plan} should be no greater than the smallest duration of a driving maneuver. Therefore in this work we set $t_{\text{plan}} = 3[\text{s}]$.

C.2.2 Computational Efficiency of Risk Approximation

In this section we add to the evaluation presented in Section 4.7.1 of the main text. We compare the computation times of the proposed method’s approximation of the risk of collision against the Cantelli inequality, and the Chance-Constrained Parallel Bernstein Algorithm (CCPBA) on the same 3000 randomly generated test cases. In each test case, (i, j, z_0, p) is randomly chosen from $\mathcal{I} \times \mathcal{J} \times \mathcal{Z}_0 \times \mathcal{P}$. Table C.1 summarizes the computation times for generating the risk of collision by approximating the following the integral: $\int_{\xi_j(z_0, p) \oplus < 0, G^{\text{obs}} >} q_{i,j}(w) dw$ when $q_{i,j}$ is modeled as a Gaussian. Table C.1 shows that RADIUS is able to compute the approximation of the risk of collision about 60% faster than CCPBA, but not as fast as evaluating the Cantelli inequality.

Method	Mean Solve Time [ms]	Maximum Solve Time [ms]
RADIUS	0.4372	2.3150
Cantelli	0.0181	2.5223
CCPBA	1.1012	3.5820

Table C.1: Results comparing the computation times for generating an approximation for the risk of collision using RADIUS, CCPBA, and the Cantelli Inequality when the obstacle uncertainty is represented as a Gaussian distribution.

C.2.3 Additional Simulation details

We discuss more in detail the experimental setup for the simulation experiments discussed in Section 4.7.2. In both the 3-lane highway and left turning scenarios, the uncertainty in the locations and predictions of obstacle motions for each obstacle is represented as a Gaussian distribution $q_{i,j}$ for the stochastic methods for any $(i, j) \in \mathcal{I} \times \mathcal{J}$. More exactly, $q_{i,j}$ is a Gaussian distribution that represents the possible location of the i -th obstacle during the j -th time interval $\mathcal{T}_j$, and is constructed such that its mean, $\mu_{i,j}$, is at the center of a lane for all $\mathcal{T}_j$. As time increases we translate this uncertain region forward at a constant speed while keeping it centered in the lane. In other words, $\mu_{i,j} - \mu_{i,j-1}$ is constant for all $j > 1$ in J . Recall, from Definition 24, that L and W are the length and width of each obstacle, respectively. Then for any $(i, j) \in \mathcal{I} \times \mathcal{J}$, the standard deviation $\sigma_{i,j}$ of $q_{i,j}$ is chosen such that the $3\sigma_{i,j}$ -region of the Gaussian distribution covers the area of width $3.7 - W$ and length $L + u_i(0) \cdot \Delta_t$, where $u_i(0)$ is the speed of the i -th obstacle at time 0 and Δ_t is the time interval defined in Section 4.3.4. Note the choice of width $3.7 - W$ ensures that the footprint of the obstacle always stays inside the lane.

To capture the stochastic nature of each scenario, we simulate each scenario for 10 trials. For each given scenario and for any $i \in \mathcal{I}$, the i -th obstacle is always initialized with the same state, but follows different trajectories among different trials of this scenario. These trajectories are selected such that the locations of the i -th obstacle during the j -th time interval $\mathcal{T}_j$ are randomly sampled from $q_{i,j}$ for each trial, where $q_{i,j}$ is kept constant for all trials of the same scenario for any $(i, j) \in \mathcal{I} \times \mathcal{J}$. This variability in the trajectories allows us to capture trials where a specific obstacle trajectory may cause a crash, whereas in other trials of the same scenario, different trajectories do not cause crashes.

Because REFINE is a deterministic motion planning algorithm, it assumes perfect knowledge of the locations of obstacles. In scenarios where there is uncertainty in the location of obstacles, deterministic algorithms like REFINE often generate motion plans to avoid the entire uncertain region. However, because Gaussian distributions have non-trivial probability density over the entire space $\mathcal{W}$, applying REFINE naively would require avoiding the entire world space. As such, we require REFINE to avoid a box that over approximates the $5\text{-}\sigma_{i,j}$ region of the Gaussian distribution for any $(i, j) \in \mathcal{I} \times \mathcal{J}$ as shown in Figure 4.3d. Note such $5\text{-}\sigma_{i,j}$ region accounts for 99.99999% of the probability mass.

C.3 Additional Experiments

In addition to the experiments reported in the main body of this work, we perform two additional experiments to evaluate the proposed method.

C.3.1 Single Planning Horizon 3-Lane Highway Environment

This experiment compares the performance of RADIUS to CCPBA, Cantelli MPC, and REFINE in a 3-lane highway driving scenario in simulation over 10 randomly generated scenarios. The aim of this experiment is to see how RADIUS performs in comparison to the other methods when trying to execute a single lane change in a small stretch of dense highway. Each scenario contains 1 static obstacle and between 4 to 9 dynamic obstacles, where the number of dynamic obstacles was randomly selected for each scenario. It is important to note that in these scenarios, since it was only over a single planning iteration, these obstacles were all spawned within a 100[m] radius of the ego vehicle. This resulted in scenarios that are denser than the situations typically encountered in the 3-lane experiment described in Section 4.7.2.1. The initial speeds of all dynamic obstacles are also randomly sampled from between 15[m/s] to 20[m/s] in each scenario. In each of these scenarios, the ego vehicle is randomly initialized in a lane, and is commanded to get as close as possible to a given

Method	Success [%]	Crash [%]	Other Action [%]
RADIUS	81.8	0.0	18.2
CCPBA	55.2	0.0	44.8
Cantelli MPC	9.1	35.4	55.5
REFINE	21.2	0.0	78.8

Table C.2: Single planning iteration results using RADIUS, CCPBA and Cantelli MPC. “Success” encompasses the trials where each method was able to successfully execute the lane change. “Other Action” encompasses the trials where a method did not complete the lane change maneuver, but instead either executed a safe stop maneuver or decided to keep driving in lane.

waypoint in a different lane. The ego vehicle is expected to achieve the task in a single planning iteration with allowable risk threshold $\epsilon = 0.05$ and $t_{\text{plan}} = 3[\text{sec}]$. Success in this experiment is characterized by being able to find a feasible lane change maneuver during the planning time and execute it without crashing. Each scenario is simulated for 200 trials, resulting in a total of 2000 simulation cases. The obstacle uncertainties for the stochastic methods and REFINE are represented in the same way as in Section 4.7.2.1. Table C.2 summarizes RADIUS, CCPBA, Cantelli MPC, and REFINE’s performance on the 3-lane single planning iteration scenarios. In this experiment we see that RADIUS was able to successfully execute this lane change maneuver more often than the comparisons.

C.3.2 Ablation Study

This section performs an ablation study to evaluate the effectiveness of each component of RADIUS. In this ablation study, we examine the performance of 3 algorithms, each of which has a certain aspect of RADIUS removed. Namely, the 3 algorithms are: RADIUS (No Analytical Gradient), Monte-Carlo RTD (Discrete) and Monte-Carlo RTD (Continuous). For RADIUS (No Analytical Gradient), we examine the performance of RADIUS without providing the constraint gradient described in Section 4.6.3. Instead, we allow IPOPT to compute a numerical gradient during the optimization. In the other 2 algorithms, instead of leveraging our over-approximation to the risk of collision from Section 4.6, we instead replace this with a Monte Carlo style sampling method to estimate the risk of collision as defined in (4.14). For Monte-Carlo RTD (Discrete), to find a viable control parameter, we sample discrete points from the control parameter space $\mathcal{P}$ in a similar fashion to [77]. To approximate the risk of collision in a Monte-Carlo fashion, we sample 100 points from $q_{i,j}$ and count the number of points that fall in the zonotope $\xi_j(z_0, p) \oplus <0, G^{\text{obs}}>$ for

Algorithm	Success [%]	Crash [%]	Other Action [%]	Solve Time [s]
MCRTD(D)	59.0	0.0	41.0	4.781
MCRTD(C)	60.0	0.0	40.0	3.334
RADIUS(NAG)	80.0	0.0	20.0	4.931
RADIUS	80.0	0.0	20.0	0.412

Table C.3: Single planning iteration results using the RADIUS variants considered in the ablation study. “Other Action” encompasses the trials where each method did not complete the lane change maneuver, but instead either executed a safe stop maneuver or decided to keep driving in the lane. RADIUS (No Analytical Gradient) is abbreviated as RADIUS (NAG), Monte-Carlo RTD (Discrete) is abbreviated as MCRTD(D) and Monte-Carlo RTD (Continuous) is abbreviated as MCRTD(C).

any $(i, j) \in \mathcal{I} \times \mathcal{J}$, where p corresponds to a sampled control parameter. We then sum up these integrals over all $i \in \mathcal{I}$ and $j \in \mathcal{J}$ to estimate the risk of collision with (4.14). Monte-Carlo RTD (Continuous) estimates the risk of collision in the same way as Monte-Carlo RTD (Discrete), however it searches for feasible not-at-fault plans over the continuous control parameter space $\mathcal{P}$ and requires IPOPT to compute a numerical gradient during only planning.

We evaluate these algorithms in another set of 10 randomly generated 3-lane highway scenarios, with the same experimental setup from C.3.1. Each scenario is simulated for 50 trials resulting in a total of 500 test cases. The results of the ablation study can be found in Table C.3. From the ablation study, it is evident that replacing the closed-form over-approximation with Monte-Carlo sampling methods causes an 8-10 $\times$ increase in the solve time due to a longer risk of collision estimation time. Additionally, both Monte-Carlo methods achieve a roughly 20% lower success rate than RADIUS, which is too slow to be able to achieve real-time performance. We note that increasing the number of samples from 100 would likely increase the success rate as the estimation of the risk of collision would get more accurate. However, this increase in the number of samples will cause the solve time to be even slower. Without the analytical gradients being provided, RADIUS(NAG) requires an average time of about 10 $\times$ slower than RADIUS. It is evident from these results that both the analytical gradient and the closed-form over-approximation play important roles in being able to generate motion plans in real time.

APPENDIX D

SPLANNING Appendices

D.1 Trajectory Design

PHLAME computes risk-aware trajectories in a receding-horizon manner by solving an optimization at each planning iteration. Offline, we pre-specify a continuum of trajectories over a compact set $K \subset \mathbb{R}^{n_k}$, $n_k \in \mathbb{N}$. Then each trajectory, defined over a compact time interval T , is uniquely determined by a by a *trajectory parameter* $k \in K$. Note that we design K for safe manipulator motion planning, but K may also be designed for other tasks and robot morphologies [45, 60, 59, 70] as long as it satisfies the following properties:

Definition 55 (Trajectory Parameters). *For each $k \in K$, a parameterized trajectory is an analytic function $q(\cdot; k) : T \rightarrow Q$ that satisfies the following properties:*

1. *The parameterized trajectory starts at a specified initial condition $(q_0, \dot{q}_0)$, so that $q(0; k) = q_0$, and $\dot{q}(0; k) = \dot{q}_0$.*
2. *Each parameterized trajectory brakes to a stop such that $\dot{q}(t_f; k) = 0$.*

PHLAME performs real-time receding horizon planning by executing the desired trajectory computed at the previous planning iteration while constructing the next desired trajectory for the subsequent time interval. Therefore, the first property ensures each parameterized trajectory is generated online and begins from the appropriate future initial condition of the robot. The second property ensures that a braking maneuver is always available to bring the robot safely to a stop.

D.2 Reachability Analysis

In Section D.2.1 we discuss how to over-approximate the robot’s desired position and velocity trajectories using polynomial zonotopes [58]. In Section D.2.2 we use the over-approximated

trajectories to construct an over-approximation of the forward occupancy in (5.10) called the Spherical Forward Occupancy.

D.2.1 Parameterized Trajectories

PHLAME enables risk-aware motion planning by constructing safety constraints that over-approximate the robot’s position, velocity, and forward occupancy over continuous time intervals. In this subsection, we briefly discuss how to overapproximate the parameterized trajectories introduced in Def. 55.

We start by choosing a timestep Δt that divides the compact time horizon $T \subset \mathbb{R}$ into $n_t := \frac{T}{\Delta t}$ time subintervals

$$\mathbf{T}_i = \left\{ t \in T \mid t = \frac{(i-1)+i}{2}\Delta t + \frac{1}{2}\Delta t x_{t_i}, \ x_{t_i} \in [-1, 1] \right\}, \quad (\text{D.1})$$

where each subinterval is indexed by the set $N_t := \{1, \dots, n_t\}$. Note that because intervals are a special case of polynomial zonotopes [58], each $\mathbf{T}_i$ is also a polynomial zonotope. We can similarly represent the compact, n_k -dimensional trajectory parameter set $K \subset \mathbb{R}^{n_k}$ as a polynomial zonotope $\mathbf{K}$. Then, the rules of polynomial zonotope arithmetic [58, 81] allow one to overapproximate the desired position and velocity trajectories by plugging $\mathbf{T}_i$ and $\mathbf{K}$ into the formulas for $q_j(t; k)$ and $\dot{q}_j(t; k)$. We restate this result as a lemma whose proof can be found in [81].

Lemma 56 (Parameterized Trajectory PZs). *The parameterized trajectory polynomial zonotopes $\mathbf{q}_j(\mathbf{T}_i; \mathbf{K})$ are overapproximative, i.e., for each $j \in n_q$ and $k \in \mathbf{K}_j$,*

$$q_j(t; k) \in \mathbf{q}_j(\mathbf{T}_i; k) \quad \forall t \in \mathbf{T}_i \quad (\text{D.2})$$

Similarly, one can define $\dot{\mathbf{q}}_j(\mathbf{T}_i; \mathbf{K})$ that are also overapproximative.

D.2.2 Spherical Forward Occupancy

This subsection briefly describes how to overapproximate the forward occupancy introduced in Section 5.3.2 with a collection of three-dimensional balls. The key idea is that one can plug the polynomial zonotope trajectories (D.2) into a polynomial zonotope version of the robot’s forward kinematics to get an overapproximation of the position of each joint over the time interval $\mathbf{T}_i$. Each joint position can then be overapproximated by a three-dimensional ball and added to the nominal joint ball (Assum. 46) using the Minkowski sum operation. Then by Assum. 46, each link is contained in the tapered capsule $TC_j(\mathbf{q}(\mathbf{T}_i; k))$ of two successive

joint balls over the time interval $\mathbf{T}_i$. The following theorem, which originally appeared in [84, Theorem 10], summarizes the results of applying these steps and proves that $TC_j(\mathbf{q}(\mathbf{T}_i; k))$ can be overapproximated by a finite number of n_s balls.

Theorem 57 (Spherical Forward Occupancy). *Let $n_s \in \mathbb{N}$ be given. Then for each $m \in N_s = \{1, \dots, n_s\}$ there exists a closed ball with center $\bar{c}_{j,i,m}(k)$ and finite radius $\bar{r}_{j,i,m}(k)$ denoted*

$$\bar{S}_{j,i,m}(\mathbf{q}(\mathbf{T}_i; k)) = B_2(\bar{c}_{j,i,m}(k), \bar{r}_{j,i,m}(k)) \quad (\text{D.3})$$

such that

$$FO_j(\mathbf{q}(\mathbf{T}_i; k)) \subset TC_j(\mathbf{q}(\mathbf{T}_i; k)) \subset \bigcup_{m=1}^{n_s} \bar{S}_{j,i,m}(\mathbf{q}(\mathbf{T}_i; k)) \quad (\text{D.4})$$

for each $j \in N_q$, $k \in \mathbf{K}$, and $t \in \mathbf{T}_i$.

For convenience, we refer to the union of balls on the right-hand side of (D.4) as the *Spherical Forward Occupancy* ($\mathcal{SFO}$) for the j^{th} link at the i^{th} timestep.

Remark 58. *Thm. 47 introduced the notation in (5.11) to simplify the description of the Spherical Forward Occupancy in the main portion of this document. However, we note here that $S_{j,i,m}(\mathbf{q}(\cdot; k))$ represents the same closed ball as $\bar{S}_{j,i,m}(\mathbf{q}(\mathbf{T}_i; k))$ described in Thm. 57.*

D.3 Splanning Proofs

This section re-states and provides proofs for the key theorems in the collision constraint of SPLANNING.

D.3.1 Proof of Collision Bound

Theorem 49. *Consider, without loss of generality, a ball $S = B_2(0, \rho)$ of radius ρ that is centered at the origin. Let α denote the risk threshold defined in (5.17) for the probability of collision between a ray and the environment. The probability that the ball S collides with the environment is bounded above by*

$$\mathbb{P}(C(S)) \leq \frac{1}{\alpha} \left[1 - \exp\left(-\frac{1}{4\pi} \int_S \sigma(\mathbf{x}) d\mathbf{x}\right) \right] \quad (\text{D.5})$$

where the integral denotes a volume integral over the sphere S and $\alpha \in \mathbb{R}^+$ is a risk threshold.

Proof. Consider the probability that a sphere is in collision with the scene, denoted $P(C(S))$ and defined in (5.17) as

$$P(C(S)) = P\left(1 - \mathcal{T}_0^\rho(\omega_{\mathbf{V}}) \geq \alpha\right), \quad \omega_{\mathbf{V}} \in \Omega_{\mathbf{V}}$$

where $\mathcal{T}_0^\rho(\omega_{\mathbf{V}})$, the probability space $(\Omega_{\mathbf{V}}, \mathcal{F}_{\mathbf{V}}, \mathcal{P}_{\mathbf{V}})$, and random variable $\mathbf{V}$ are as defined in Section 5.4.2. From Markov's Inequality, we have that

$$\begin{aligned} P(C(S)) &= P\left(1 - \mathcal{T}_0^\rho(\omega_{\mathbf{V}}) \geq \alpha\right) \\ &\leq \frac{\mathcal{E}_{\omega_{\mathbf{V}} \in \Omega_{\mathbf{V}}}[1 - \mathcal{T}_0^\rho(\omega_{\mathbf{V}})]}{\alpha}. \end{aligned} \quad (\text{D.6})$$

Next, we compute the expectation

$$\mathcal{E}_{\omega_{\mathbf{V}} \in \Omega_{\mathbf{V}}}[1 - \mathcal{T}_0^\rho(\omega_{\mathbf{V}})] = \int_{\Omega_{\mathbf{V}}} (1 - \mathcal{T}_0^\rho(\omega_{\mathbf{V}})) d\mathcal{P}_{\mathbf{V}} \quad (\text{D.7})$$

$$= 1 - \int_{\Omega_{\mathbf{V}}} \exp\left(-\int_0^\rho \sigma(\Phi(t, \omega_{\mathbf{V}})) dt\right) d\mathcal{P}_{\mathbf{V}} \quad (\text{D.8})$$

$$\leq 1 - \exp\left(-\int_{\Omega_{\mathbf{V}}} \int_0^\rho \sigma(\Phi(t, \omega_{\mathbf{V}})) dt d\mathcal{P}_{\mathbf{V}}\right) \quad (\text{D.9})$$

The equality in (D.8) results from (5.12), and the inequality in (D.9) results from Jensen's inequality.

Since $\mathbf{V} : \Omega_{\mathbf{V}} \rightarrow \mathbb{S}^2$ is a random variable that maps $\omega_{\mathbf{V}}$ to ray directions $v_\varphi \in \mathbb{S}^2$, and $\mathbf{V}$ is uniformly distributed over $\mathbb{S}^2$ under $\mathcal{P}_{\mathbf{V}}$, we may push forward the measure $d\mathcal{P}_{\mathbf{V}}$ to the uniform measure on $\mathbb{S}^2$. As a result, we may rewrite the outer integral in (D.7) as an integral over $\mathbb{S}^2$. To do so, we introduce the normalizing factor $\frac{1}{4\pi}$, corresponding to the surface area of $\mathbb{S}^2$. That is,

$$\begin{aligned} \mathcal{E}_{\omega_{\mathbf{V}} \in \Omega_{\mathbf{V}}}[1 - \mathcal{T}_0^\rho(\omega_{\mathbf{V}})] \\ \leq 1 - \exp\left(-\frac{1}{4\pi} \int_{\mathbb{S}^2} \int_0^\rho \sigma(\varphi(t; v_\varphi)) dt dv_\varphi\right) \end{aligned} \quad (\text{D.10})$$

Above, $\varphi(t, v_\varphi) = tv_\varphi$. Then, the double integral in (D.10) may be recognized as the volume integral over the ball S , and (D.10) is equivalent to

$$\mathcal{E}_{\omega_{\mathbf{V}} \in \Omega_{\mathbf{V}}}[1 - \mathcal{T}_0^\rho(\omega_{\mathbf{V}})] \leq 1 - \exp\left(-\frac{1}{4\pi} \int_S \sigma(\mathbf{x}) d\mathbf{x}\right). \quad (\text{D.11})$$

Combining (D.11) with (D.6) results in the theorem. $\square$

D.3.2 Proof of Closed-Form Bound

Theorem 50. *Let $S = B_2(0, \rho)$ be a closed ball of radius ρ that is centered at the origin. Suppose the density function $\sigma : \mathbb{R}^3 \rightarrow \mathbb{R}$ is represented by a set of $n_G \in \mathbb{N}$ normalized Gaussian basis functions $\{G_n\}_{n=1}^{n_G}$ with means $\{\mu_n\}_{n=1}^{n_G}$ and covariance matrices $\{\Sigma_n\}_{n=1}^{n_G}$ with Eigendecompositions given by $\Sigma_n = R_n \Lambda_n R_n^T$. Finally, let the diagonal elements of Λ_n be denoted as $\lambda_{n,1}, \lambda_{n,2}, \lambda_{n,3}$ and represent the eigenvalues of Σ_n . Then*

$$\int_S \sigma(\mathbf{x}) d\mathbf{x} \leq \mathcal{H}(S) \quad (\text{D.12})$$

where

$$\begin{aligned} \mathcal{H}(S) &= \sum_{n=1}^{n_G} \eta'_n w_n \cdot \\ &\prod_{\ell=1}^3 \left[\sqrt{\frac{\pi \lambda'_{n,\ell}}{2}} \left(\operatorname{erf} \left(\frac{\rho - \mu'_{n,\ell}}{\sqrt{2\lambda'_{n,\ell}}} \right) - \operatorname{erf} \left(\frac{\rho + \mu'_{n,\ell}}{\sqrt{2\lambda'_{n,\ell}}} \right) \right) \right]. \end{aligned} \quad (\text{D.13})$$

Above, erf denotes the error function and μ'_n, λ'_n , and η'_n correspond to the mean, eigenvalues, and normalization constant of the normalized Gaussian G'_n obtained by rotating G_n by R_n^T . That is, G'_n has mean $\mu'_n = R_n^T \mu_n$ and covariance $\Sigma'_n = R_n^T \Sigma_n R_n$. Furthermore,

$$P(C(S)) \leq \frac{1}{\alpha} \left[1 - \exp \left(-\frac{1}{4\pi} \int_S \sigma(\mathbf{x}) d\mathbf{x} \right) \right] \quad (\text{D.14})$$

$$\leq \frac{1}{\alpha} \left[1 - \exp \left(-\frac{1}{4\pi} \mathcal{H}(S) \right) \right]. \quad (\text{D.15})$$

Thus, we may constrain the risk of collision by enforcing

$$\left[1 - \exp \left(-\frac{1}{4\pi} \mathcal{H}(S) \right) \right] < \alpha \cdot \beta \quad (\text{D.16})$$

for a given risk threshold $\beta \in (0, 1]$.

Proof. We assume without loss of generality that the ball S is at the origin of the coordinate

system. Then,

$$\begin{aligned}\int_S \sigma(\mathbf{x}) d\mathbf{x} &= \int_S \left(\sum_{n=1}^{n_G} w_n G_n(\mathbf{x}) \right) d\mathbf{x} \\ &= \sum_{n=1}^{n_G} w_n \int_S G_n(\mathbf{x}) d\mathbf{x}.\end{aligned}\tag{D.17}$$

For each Gaussian G_n , let $\Sigma_n = R_n \Lambda_n R_n^T$ be the Eigen decomposition of Σ_n . Transforming G_n by the rotation matrix R_n^T , we obtain an axis-aligned Gaussian G'_n with mean $\mu'_n = R_n^T \mu$ and covariance matrix $\Sigma'_n = \Lambda_n = \text{diag}\{\lambda'_1, \lambda'_2, \lambda'_3\}$. Because rotating the ball S by R_n^T leaves the domain of integration unchanged, we have

$$\begin{aligned}\int_S \sigma(\mathbf{x}) d\mathbf{x} &= \sum_{n=1}^{n_G} w_n \int_S G_n(\mathbf{x}) d\mathbf{x} \\ &= \sum_{n=1}^{n_G} w_n \int_S G'_n(\mathbf{x}) d\mathbf{x}.\end{aligned}\tag{D.18}$$

Note that the integral of an axis-aligned Gaussian over a closed ball does not necessarily have a closed-form solution. Therefore, we overapproximate the domain of integration by replacing the closed L_2 ball S with the closed L_∞ ball $S_\infty = B_\infty(0, \rho)$.

Since the covariance matrix $\Sigma'_n = \text{diag}\{\lambda'_1, \lambda'_2, \lambda'_3\}$ is diagonal, we obtain

$$\begin{aligned}\int_S \sigma(\mathbf{x}) d\mathbf{x} &= \sum_{n=1}^{n_G} w_n \int_S G'_n(\mathbf{x}) d\mathbf{x} \\ &\leq \sum_{n=1}^{n_G} w_n \int_{S_\infty} G'_n(\mathbf{x}) d\mathbf{x} \\ &= \sum_{n=1}^{n_G} \eta'_n w_n \prod_{\ell=1}^3 \left[\sqrt{\frac{\pi \lambda'_{n,\ell}}{2}} \left(\text{erf} \left(\frac{\rho - \mu'_{n,\ell}}{\sqrt{2 \lambda'_{n,\ell}}} \right) + \right. \right. \\ &\quad \left. \left. - \text{erf} \left(\frac{\rho + \mu'_{n,\ell}}{\sqrt{2 \lambda'_{n,\ell}}} \right) \right) \right] \\ &= \mathcal{H}(S),\end{aligned}\tag{D.19}$$

where erf denotes the error function and μ'_n , λ'_n , and η'_n correspond to the mean, eigenvalues, and normalization constant of the normalized Gaussian G'_n . The result in Theorem 50 is now obtained by substituting (D.19) into (5.18). $\square$

D.4 Baseline Implementation Details

This section details the implementation of the baselines in the constraint comparisons of Section 5.5.3.

D.4.1 CATNIPS

CATNIPS is a chance-constrained planning algorithm for robot navigation in a NeRF environment. At a high level, CATNIPS relates a NeRF to a Poison Point Process (PPP) and uses this relation to create an occupancy grid called a PURR (Probabilistically Unsafe Robot Region).

D.4.1.1 Intensity Grid

First, a NeRF representation of the PyRender scene is built using Nerfstudio [106]. The space is then voxelized using a uniform scale of 150 voxels per side, as outlined in the CATNIPS paper [18], and the NeRF density field is transformed into a discretized Cell Intensity Grid from this voxelized representation according to [18, Eq. 16-17].

D.4.1.2 Robot Kernel

A robot kernel is created by performing a Minkowski sum between the robot’s minimal bounding sphere and a single voxel. This kernel is convolved with the intensity grid, resulting in the robot intensity grid. This robot intensity grid is transformed into the PURR, as described in [18, Section 5].

To evaluate collisions between the robot and the scene, we check whether the center of each sphere in the arm’s SFO intersects the PURR. Due to the convolution of the intensity grid with the robot kernel, this is equivalent to checking if the sphere intersects a discretized form of the intensity grid.

D.4.2 Splat-Nav

Splat-Nav evaluates collision in an un-normalized 3D Gaussian Splat. Hence, splatting models of each evaluation scene were constructed using the standard 3DGS formulation [56]. However, to ensure a fair comparison, a modified version of 3DGS was used that includes depth supervision.

We then evaluated the collisions along each sphere using the $K(s)$ Bisection Search introduced in [19, Algorithm 1]. We note that the provided implementation checks for collisions

along linear motions with a scaling factor of δ_x ; to adapt this methodology to collision-checking individual spheres we set $\delta_x = 10^{-8}$, effectively setting the linear motion to a constant 0. For convenience, this result is summarized below.

Splat-Nav models the robot $\mathcal{R}$ and Gaussian Splat element G as ellipsoids $\mathcal{E}_{\mathcal{R}}$ and $\mathcal{E}_G$, with means $\mu_{\mathcal{R}}, \mu_G$ and covariances $\Sigma_{\mathcal{R}}, \Sigma_G$ respectively. They additionally model motion by parameterizing $\mathcal{E}_{\mathcal{R}}$ as moving over a line segment, $\mu_{\mathcal{R}}(t) = \mu_{\mathcal{R}0} + t\delta_x$ for $t \in [0, 1]$. Let Σ_G be decomposed by $\Sigma_G = R\Lambda GR^T$. Further, denote $\Lambda_G = \text{diag}\{\lambda_G\}$, where $\lambda_G \in \mathbb{R}^3$ are the eigenvalues of Σ_G . Then, Splat-Nav states that $\mathcal{E}_{\mathcal{R}} \cap \mathcal{E}_G = \emptyset$ if and only if

$$\min_{t \in [0, 1]} \max_{s \in (0, 1)} K(s, t) > 1, \quad (\text{D.20})$$

where

$$K(s, t) = w^T \text{diag} \left(\frac{s(1-s)}{\kappa^2 + s(\lambda_G - \kappa^2)} \right) w, \quad (\text{D.21})$$

$\kappa^2 \in \mathbb{R}$ is the maximum eigenvalue of $\Sigma_{\mathcal{R}}$, and $w(t) = R^T(\mu_{\mathcal{R}0} + t\delta_x - \mu_G)$.

D.5 Normalized 3D Gaussian Splatting

This appendix provides a detailed derivation of the rasterization process of a normalized 3D Gaussian Splat and the computation of its derivative with respect to the splatting parameters. In order to remain consistent with the splatting literature, the notation in this section is self-contained and distinct from the notation used in the rest of the document. In practice, the reference implementation of the original 3D Gaussian Splatting method [56] was modified to include normalization in both the forward and backward pass.

D.5.1 Radiance Field Preliminaries

First, we re-state the definition of a radiance field. A radiance field models the image formation process by approximating a five-dimensional function that maps from a point and viewing directions to the differential opacity $\sigma : \mathbb{R}^3 \rightarrow \mathbb{R}$ and the color $c : \mathbb{R}^3 \times S^2 \rightarrow \mathbb{R}^3$. Intuitively, σ represents the likelihood that a particle traveling along a ray will stop when it collides with an individual point in space. Under a radiance field model, an image is rendered by considering separately the ray that intersects each pixel. Consider a ray $r(s) : \mathbb{R} \rightarrow \mathbb{R}^3$. We define an additional quantity, the *transmittance*, which defines the probability that a particle traveling along ray r from $s = a$ to $s = b$ will experience a collision. This transmittance, denoted as $\mathcal{T}_a^b[r]$, is computed by

$$\mathcal{T}_a^b[r] = \exp \left(- \int_a^b \sigma(r(s)) ds \right) \quad (\text{D.22})$$

Equivalently, $\mathcal{T}_a^b[r]$ may be interpreted by considering a light source aligned with ray r and located at $s = a$. Then $\mathcal{T}_a^b[r]$ is the proportion of light that remains by point $s = b$.

From the transmittance, the expected color of the pixel corresponding to ray r can be computed as

$$\mathcal{C}[r] = \int_{z_n}^{z_f} c(r(s), r_d) \sigma(r(s)) T_{z_n}^s[r] ds \quad (\text{D.23})$$

where z_n, z_f are the near and far clipping planes. The expected depth can be analogously computed as

$$\mathcal{D}[r] = \int_{z_n}^{z_f} s \cdot \sigma(r(s)) T_{z_n}^s[r] ds \quad (\text{D.24})$$

D.5.2 Summary of Splatting Methods

This section summarizes splatting methods, while the rest of the Appendix provides a detailed derivation of Normalized 3DGS. We refer the interested reader to [129] for a more comprehensive treatment of splatting-based rendering. In a splatting framework, 3D Gaussian functions are used as a basis set to approximate σ and c . We call each Gaussian function a 3D reconstruction kernel $\kappa(x)$ in world coordinates. Then, each 3D function $\kappa(x)$ is transformed into a coordinate system aligned with each camera ray. This yields a 3D reconstruction kernel $\kappa'(x)$ in ray coordinates. Next, each kernel is integrated along the depth dimension of the ray to yield a 2D image-space reconstruction kernel $q(x)$. Finally, the 2D kernels are queried at the center of each pixel and blended to compute the final pixel color.

D.5.3 Representation

Let n_G 3D Gaussian density functions be reconstruction kernels to represent the scene. Let each function parameterized by mean μ_k and covariance Σ_k , both expressed in world coordinates. Denote $S_k = \Sigma_k^{-1}$. Then, the 3D reconstruction kernel is given by the 3D Gaussian density

$$G_k(x) = \frac{1}{\sqrt{(2\pi)^3 |\Sigma_k|}} \exp \left(-\frac{1}{2} \|x - \mu_k\|_{S_k}^2 \right) \quad (\text{D.25})$$

where the represented norm is the Mahalanobis distance. For notational simplicity, we denote

$$n_k = \frac{1}{\sqrt{(2\pi)^3 |\Sigma_k|}} \quad (\text{D.26})$$

as the normalizing constant of the Gaussian. Finally, the differential opacity σ at a point x is given by

$$\begin{aligned}\sigma(x) &= \sum_{k=1}^{n_G} w_k G_k(x) \\ &= w_k n_k \exp\left(-\frac{1}{2} \|x - \mu_k\|_{S_k}^2\right)\end{aligned}\tag{D.27}$$

where $w_k \in \mathbb{R}^+$ are positive weights.

Additionally, assume without loss of generality that the Gaussian carries with it a viewpoint-independent color $c_k \in \mathbb{R}^3$. This assumption is made without loss of generality because, in practice, a viewpoint-dependent color may be converted to a constant color once a viewpoint is selected. Additionally, we assume the covariance Σ_k is provided. In implementation, Σ_k is parameterized by its Eigendecomposition; this detail is omitted for brevity.

In summary, the parameters of each Gaussian G_k in the scene representation are the mean of the Gaussian $\mu_k \in \mathbb{R}^3$; the covariance of the Gaussian $\Sigma_k \in \mathbb{R}^{3 \times 3}$; the color of each Gaussian $c_k \in \mathbb{R}^3$; and the weight of each Gaussian $w_k \in \mathbb{R}^+$.

D.5.4 Integration of an Individual Gaussian

During the forward pass, each Gaussian is transformed into a coordinate frame aligned with each camera ray, which reduces the integration problem to a set of Gaussian marginalizations.

This section closely follows Section 6.2 of [129], with slight modifications for coordinate systems and notation. Further, [119] provides a similar derivation for the more standard un-normalized 3D Gaussian Splatting. We consider the integration of an individual reconstruction kernel $\kappa_k(x) = G_k(x)$ along an individual camera ray. Let G_k have mean μ_k and covariance Σ_k expressed in world coordinates. Denote the homogenous $SE(3)$ transformation from world coordinates to camera coordinates as

$$T_{CW} = \begin{bmatrix} R_{CW} & d_{CW} \\ 0 & 1 \end{bmatrix},\tag{D.28}$$

where $R_{CW} \in SO(3)$ is a rotation matrix and $d_{CW} \in \mathbb{R}^3$ is a translation vector.

Consider transforming a point $x \in \mathbb{R}^3$ to image coordinates. First, we transform it to camera coordinates by $t = \varphi(x) = R_{CW}x + d_{CW}$, where $t \in \mathbb{R}^3$. Denote the mean in camera coordinates as $\mu_k^t = \varphi(\mu_k)$. Then, points are transformed to a ray coordinate system via a projective transformation. We use the standard projection matrix from camera to clip space,

defined as

$$P = \begin{bmatrix} 2f_x/W & 0 & 0 & 0 \\ 0 & 2f_y/H & 0 & 0 \\ 0 & 0 & \frac{z_f+z_n}{z_f-z_n} & -2\frac{z_f z_n}{z_f-z_n} \\ 0 & 0 & 1 & 0 \end{bmatrix}. \quad (\text{D.29})$$

We simultaneously convert the first two components to pixel units via the camera's focal lengths f_x and f_y , image size (W, H) , and principal point (c_x, c_y) :

$$\phi(t) = \begin{bmatrix} \frac{WP \cdot t_x}{2P \cdot t_w} + 0.5 + c_x \\ \frac{HP \cdot t_y}{2P \cdot t_w} + 0.5 + c_y \\ \|t\| \end{bmatrix} \quad (\text{D.30})$$

$$= \begin{bmatrix} f_x t_x / t_z + 0.5 + c_x \\ f_y t_y / t_z + 0.5 + c_y \\ \|t\| \end{bmatrix}. \quad (\text{D.31})$$

Since the transformation (D.31) is not affine, it is approximated with a first-order Taylor approximation around μ^t :

$$\phi_{\mu_k^t}(t) = \phi(\mu_k^t) + J_k(t - \mu_k^t) \quad (\text{D.32})$$

where J_k is the Jacobian of the transformation in (D.31). In particular,

$$J_k = \begin{bmatrix} f_x / \mu_{k,z}^t & 0 & -f_x \mu_{k,x}^t / \mu_{k,z}^t{}^2 \\ 0 & f_y / \mu_{k,z}^t & -f_y \mu_{k,y}^t / \mu_{k,z}^t{}^2 \\ \mu_{k,x}^t / \|t_k\| & \mu_{k,y}^t / \|t_k\| & \mu_{k,z}^t / \|\mu_k^t\| \end{bmatrix} \quad (\text{D.33})$$

Applying these transformations results in a ray-space reconstruction kernel given by

$$\kappa'(u) = |R_{CW}| |J_k| G'_k(u) \quad (\text{D.34})$$

$$\mu'_k = \phi(\varphi(\mu_k)) \quad (\text{D.35})$$

$$\Sigma'_k = J_k R_{CW} \Sigma R_{CW}^\top J_k^\top \quad (\text{D.36})$$

where G'_k is a Gaussian density with mean μ'_k and covariance Σ'_k . We refer the interested reader to [129, Section 6.2.2] for a detailed justification of the above. Note that R_{CW} is a rotation matrix, so its determinant $|R_{CW}| = 1$ will be excluded moving forward.

Finally, the image-space 2D reconstruction kernel is computed by marginalizing the third

dimension of κ' . As demonstrated in [129, Section 6.2.3], this is equivalent to

$$q_k(\hat{u}) = |J_k| \hat{G}'_k(\hat{u}) \quad (\text{D.37})$$

where $\hat{u} \in \mathbb{R}^2$ indicates a point on the image plane in pixel coordinates, and $\hat{G}'$ is a 2D Gaussian function with mean $\hat{\mu}'_k$ and covariance $\hat{\Sigma}'_k$ constructed by the first two rows (and columns) of the corresponding parameters of G'_k . We will denote the normalizing constant of $\hat{G}'_k$ as $\hat{n}_k$.

D.5.5 Compositing the Integrated Gaussians

The per-Gaussian integration yields 2D reconstruction kernels which are then queried and blended onto the image plane to complete the rasterization. In this description, we omit implementation-specific details of rasterization, including how the pixels are split into tiles, which Gaussians are included for rasterization in which tiles, and how the Gaussians are sorted. Instead, we focus only on mathematically relevant aspects of the process, and assume without loss of generality that every Gaussian is included for rasterization in every pixel.

With this assumption, we may write a rasterization form of (D.23) to compute the color of a pixel with 2D coordinates $\hat{u}$

$$\mathcal{C}(\hat{u}) = \sum_{k=1}^{n_G} w_k c_k q_k(\hat{u}) \prod_{j=1}^{k-1} (1 - w_j q_j(\hat{u})). \quad (\text{D.38})$$

Zwicker et. al. [129] provides a more thorough derivation of the above. We may recognize the product on the right-hand-side as the transmittance $\mathcal{T}_k(\hat{u})$ and define $\alpha_k(\hat{u}) = w_k q_k(\hat{u})$ to simplify (D.38) to

$$\mathcal{C}(\hat{u}) = \sum_{k=1}^{n_G} \alpha_k(\hat{u}) c_k \mathcal{T}_k(\hat{u}). \quad (\text{D.39})$$

In practice, the above is implemented by iterating over the Gaussian functions front-to-back once per pixel, and keeping running track of the product on the right-hand-side. The depth is analogously rendered by

$$\mathcal{D}(\hat{u}) = \sum_{k=1}^{n_G} \alpha_k(\hat{u}) \mu_{k,z}^t \mathcal{T}_k(\hat{u}) \quad (\text{D.40})$$

where $\mu_{k,z}^t$ denotes the z component of the k^{th} mean in the camera's frame.

D.5.6 Backward Pass

In this section, we derive the gradient of the rendered color $\mathcal{C}(\hat{u})$ and depth $\mathcal{D}(\hat{u})$ of the pixel $\hat{u}$ with respect to the parameters of the Gaussian Splat. We derive the gradient for a single pixel $\hat{u} \in \mathbb{R}^2$; in practice, the total gradient for each Gaussian parameter results from the summation of the below derivation over all the pixels.

We may write the contribution of the k^{th} Gaussian to $\mathcal{C}(\hat{u})$ and $\mathcal{D}(\hat{u})$ as

$$\mathcal{C}_k(\hat{u}) = \alpha_k(\hat{u})c_k\mathcal{T}_k(\hat{u}) \quad (\text{D.41})$$

$$\mathcal{D}_k(\hat{u}) = \alpha_k(\hat{u})\mu_{k,z}^t\mathcal{T}_k(\hat{u}) \quad (\text{D.42})$$

Next, (D.39) and (D.40) give that

$$\frac{\partial \mathcal{C}(\hat{u})}{\partial \mathcal{C}_k(\hat{u})} = \frac{\partial \mathcal{D}(\hat{u})}{\partial \mathcal{D}_k(\hat{u})} = 1. \quad (\text{D.43})$$

Thus, we generally consider the impact of a single Gaussian on the rendered color and depth. The dependence of variables on $\hat{u}$ will be omitted for notational simplicity. Finally, we assume that an arbitrary loss function $\mathcal{L}$ is used in training. Further, we assume that the backward pass of the optimization provides the gradients of the loss function with respect to the rendered depth $\mathcal{D}_k$ and the rendered color $\mathcal{C}_k$. We denote these input gradients as

$$\frac{\partial \mathcal{L}}{\partial \mathcal{D}_k}, \frac{\partial \mathcal{L}}{\partial \mathcal{C}_k}$$

respectively. The remainder of this document specifies how to propagate these given gradients to the parameters of the Gaussian Splat.

D.5.7 Gradient of the Render Process

We begin at the end of the forward pass by differentiating (D.39), in which the integrated 2D Gaussians are composited to form pixels. The input to the forward render pass are the 2D means $\hat{\mu}'_k$, inverses of the 2D covariances $\hat{S}'_k$, RGB colors c_k , normalizing constants $\hat{n}_k$, and the determinant of the projection operation $|J_k|$. For convenience, denote

$$\tilde{n}_k = |J_k|\hat{n}_k. \quad (\text{D.44})$$

The backwards render pass propagates gradients with respect to $\hat{\mu}'_k$, $\hat{S}'_k$, c_k , and $\tilde{n}_k$.

In implementation, we iterate over the Gaussians back-to-front. During the forward pass, we store the final transmittance $\mathcal{T}_{n_G}$ for each pixel such that during the backward pass, we

may iteratively compute

$$\mathcal{T}_{k-1} = \frac{\mathcal{T}_k}{1 - \alpha_{k-1}}. \quad (\text{D.45})$$

Starting with the color parameters c_k , we find

$$\frac{\partial \mathcal{C}_k}{\partial c_k} = \alpha_k \mathcal{T}_k. \quad (\text{D.46})$$

$$\frac{\partial \mathcal{L}}{\partial c_k} = \alpha_k \mathcal{T}_k \cdot \frac{\partial \mathcal{L}}{\partial \mathcal{C}_k}. \quad (\text{D.47})$$

This completes the gradient with respect to the color parameter c_k .

Similarly, we may differentiate (D.40) with respect to the input depth of the Gaussian, $\mu_{k,z}^t$:

$$\frac{\partial \mathcal{D}_k}{\partial \mu_{k,z}^t} = \alpha_k \mathcal{T}_k \quad (\text{D.48})$$

$$\left. \frac{\partial \mathcal{L}}{\partial \mu_{k,z}^t} \right|_{\mathcal{D}_k} = \alpha_k \mathcal{T}_k \cdot \frac{\partial \mathcal{L}}{\partial \mathcal{D}_k}, \quad (\text{D.49})$$

where the bottom term denotes the contribution to the gradient due to the rendered depth. We will put this term aside and revisit it later in the backward pass.

Next, we consider the derivative of the rendered color with respect to α_k .

$$\frac{\partial \mathcal{C}_k}{\partial \alpha_k} = c_k \mathcal{T}_k. \quad (\text{D.50})$$

We additionally must consider the impact of α_k on all future colors. Consider that the rendered color of a pixel can be written as

$$\mathcal{C} = \sum_{j=1}^k \mathcal{C}_j + \bar{\mathcal{C}}_k \quad (\text{D.51})$$

where $\bar{\mathcal{C}}_k$ denotes the contributions to $\mathcal{C}$ of all Gaussians after the k^{th} Gaussian. In particular,

$$\bar{\mathcal{C}}_k = \sum_{j=k+1}^{n_G} c_j \alpha_j \mathcal{T}_j. \quad (\text{D.52})$$

Differentiating with respect to α_k gives

$$\frac{\partial \bar{\mathcal{C}}_k}{\partial \alpha_k} = -\frac{\bar{\mathcal{C}}_k}{1 - \alpha_k} \quad (\text{D.53})$$

$$\left. \frac{\partial \mathcal{L}}{\partial \alpha_k} \right|_{\mathcal{C}} = \frac{\partial \mathcal{L}}{\partial \mathcal{C}} \left(c_k \mathcal{T}_k - \frac{\bar{\mathcal{C}}_k}{1 - \alpha_k} \right) \quad (\text{D.54})$$

Note that $\bar{\mathcal{C}}_k$ may also be computed iteratively backwards beginning with $\bar{\mathcal{C}}_{n_G} = 0$. Analogously,

$$\bar{\mathcal{D}}_k = \sum_{j=k+1}^{n_G} \mu_{j,z}^t \alpha_j \mathcal{T}_j \quad (\text{D.55})$$

$$\left. \frac{\partial \mathcal{L}}{\partial \alpha_k} \right|_{\mathcal{D}} = \frac{\partial \mathcal{L}}{\partial \mathcal{D}_k} \left(\mu_{k,z}^t \mathcal{T}_k - \frac{\bar{\mathcal{D}}_k}{1 - \alpha_k} \right) \quad (\text{D.56})$$

This allows

$$\frac{\partial \mathcal{L}}{\partial \alpha_k} = \left. \frac{\partial \mathcal{L}}{\partial \alpha_k} \right|_{\mathcal{C}} + \left. \frac{\partial \mathcal{L}}{\partial \alpha_k} \right|_{\mathcal{D}} \quad (\text{D.57})$$

We now continue to backpropagate through α_k . Recall that

$$\alpha_k = w_k \tilde{n}_k \exp \left(-\frac{1}{2} \|\hat{u} - \hat{\mu}'_k\|_{\hat{S}'_k}^2 \right). \quad (\text{D.58})$$

Since it will be frequently used, denote

$$\rho_k := \exp \left(-\frac{1}{2} \|\hat{u} - \hat{\mu}'_k\|_{\hat{S}'_k}^2 \right). \quad (\text{D.59})$$

From (D.58), we find

$$\frac{\partial \alpha_k}{\partial w_k} = \tilde{n}_k \rho_k, \quad \frac{\partial \alpha_k}{\partial \tilde{n}_k} = w_k \rho_k, \quad \frac{\partial \alpha_k}{\partial \rho_k} = w_k \tilde{n}_k, \quad (\text{D.60})$$

which gives

$$\frac{\partial \mathcal{L}}{\partial w_k} = \frac{\partial \mathcal{L}}{\partial \alpha_k} \tilde{n}_k \rho_k \quad (\text{D.61})$$

$$\frac{\partial \mathcal{L}}{\partial \tilde{n}_k} = \frac{\partial \mathcal{L}}{\partial \alpha_k} w_k \rho_k \quad (\text{D.62})$$

$$\frac{\partial \mathcal{L}}{\partial \rho_k} = \frac{\partial \mathcal{L}}{\partial \alpha_k} w_k \tilde{n}_k. \quad (\text{D.63})$$

The second equation (D.62) completes the backward pass of the render process with respect

to the input $\tilde{n}_k$.

We now continue to backpropagate through ρ_k . Denote

$$\Delta u_k = \hat{u} - \hat{\mu}'_k. \quad (\text{D.64})$$

Then we may compute the partial derivative of ρ_k with respect to the 2D mean $\hat{\mu}'_k$ via

$$\frac{\partial \rho_k}{\partial \Delta u_k} = -\rho_k \hat{S}' \Delta u_k \quad (\text{D.65})$$

and

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \hat{\mu}'_k} &= \frac{\partial \mathcal{L}}{\partial \alpha_k} \cdot \frac{\partial \alpha_k}{\partial \rho_k} \cdot \frac{\partial \rho_k}{\partial \Delta u_k} \cdot \frac{\partial \Delta u_k}{\partial \hat{\mu}'_k} \\ &= -\frac{\partial \mathcal{L}}{\partial \alpha_k} \cdot \frac{\partial \alpha_k}{\partial \rho_k} \cdot \frac{\partial \rho_k}{\partial \Delta u_k}. \end{aligned} \quad (\text{D.66})$$

This completes the gradient of the loss with respect to the 2D mean.

We may also compute the partial derivative of ρ_k with respect to the elements of $\hat{S}'_k$, the inverse of the 2D covariance, via

$$\frac{\partial \rho_k}{\partial \hat{S}'_k} = \frac{\rho_k}{2} \Delta u_k^\top \Delta u_k, \quad (\text{D.67})$$

which allows

$$\frac{\partial \mathcal{L}}{\partial \hat{S}'_k} = \frac{\partial \mathcal{L}}{\partial \alpha_k} \cdot \frac{\partial \alpha_k}{\partial \rho_k} \cdot \frac{\partial \rho_k}{\partial \hat{S}'_k}. \quad (\text{D.68})$$

This completes the gradient of the rendered color with respect to the inverse 2D covariance.

We have now computed the gradients of $\mathcal{L}$ with respect to $\hat{\mu}'_k$, $\hat{S}'_k$, c_k , and $\tilde{n}_k$, which completes the backward pass of the rendering function.

D.5.8 Gradients of the Gaussian Integration

The second step of the backward pass is to propagate the gradients through the coordinate transformations and marginalization described in Section D.5.4. We differentiate the outputs of the integration step, which are $\hat{\mu}'_k$, $\hat{S}'_k$, and $\tilde{n}_k$.

Derivatives of $\tilde{n}_k$ We begin by considering the derivatives of $\tilde{n}_k$. Recall that

$$\tilde{n}_k = \frac{|J_k|}{2\pi \sqrt{|\hat{\Sigma}'_k|}}. \quad (\text{D.69})$$

We start by considering the derivative of $\tilde{n}_k$ with respect to the elements of J_k . Recall that J_k appears not just in the numerator of (D.69), but also in the definition of $\hat{\Sigma}'_k$. First, from [88, Eq. 49], we have the derivative of $\tilde{n}_k$ with respect to J_k due to the inclusion of $|J_k|$ is

$$\left. \frac{\partial \tilde{n}_k}{\partial J_k} \right|_{|J_k|} = \frac{|J_k| J_k^{-T}}{2\pi \sqrt{|\hat{\Sigma}'_k|}}. \quad (\text{D.70})$$

We now pause dealing with J_k , and turn our attention to the $|\Sigma'_k|$ term in the denominator of (D.69). By the same identity from [88, 49],

$$\frac{\partial \tilde{n}_k}{\partial \hat{\Sigma}'_k} = - \frac{|J_k| \left(\hat{\Sigma}'_k \right)^{-T}}{4\pi \sqrt{|\hat{\Sigma}'_k|}}. \quad (\text{D.71})$$

Continuing,

$$\frac{\partial \tilde{n}_k}{\partial \Sigma'_k} = \begin{bmatrix} \frac{\partial \tilde{n}_k}{\partial \hat{\Sigma}'_k} & 0 \\ 0 & 0 \end{bmatrix} \quad (\text{D.72})$$

$$\left. \frac{\partial \mathcal{L}}{\partial \Sigma'_k} \right|_{\tilde{n}_k} = \frac{\partial \mathcal{L}}{\partial \tilde{n}_k} \cdot \frac{\partial \tilde{n}_k}{\partial \Sigma'_k} \quad (\text{D.73})$$

where the second term denotes the contribution to the partial derivative by $\tilde{n}_k$.

Derivatives of $\hat{S}'_k$ Given the derivative $\frac{\partial \mathcal{L}}{\partial \hat{S}'_k}$ computed in (D.68), we may repeatedly apply [88, Eq. 60] to find the partial derivatives with respect to $\hat{\Sigma}'_k = \left(\hat{S}'_k \right)^{-1}$. The results are omitted for brevity.

From this, it is trivial to apply the chain rule element-wise to obtain the partial derivatives of $\mathcal{L}$ with respect to $\hat{\Sigma}'_k$ due to $\hat{S}'_k$, which we denote

$$\left. \frac{\partial \mathcal{L}}{\partial \hat{\Sigma}'_k} \right|_{\hat{S}'_k}$$

Continuing, we observe

$$\left. \frac{\partial \mathcal{L}}{\partial \Sigma'_k} \right|_{\hat{S}'_k} = \begin{bmatrix} \left. \frac{\partial \mathcal{L}}{\partial \hat{\Sigma}'_k} \right|_{\hat{S}'_k} & 0 \\ 0 & 0 \end{bmatrix}. \quad (\text{D.74})$$

Finally, we arrive at the derivative of the loss $\mathcal{L}$ with respect to the transformed covariance

Σ'_k

$$\frac{\partial \mathcal{L}}{\partial \Sigma'_k} = \frac{\partial \mathcal{L}}{\partial \Sigma'_k} \Big|_{\hat{S}'_k} + \frac{\partial \mathcal{L}}{\partial \Sigma'_k} \Big|_{\tilde{n}_k} \quad (\text{D.75})$$

We conclude the derivatives with respect to Σ_k by transforming this back to the source coordinate system by

$$\frac{\partial \mathcal{L}}{\partial \Sigma_k} = R_{CW}^\top J_k^\top \frac{\partial \mathcal{L}}{\partial \Sigma'_k} J_k R_{CW}. \quad (\text{D.76})$$

Now, we continue to backpropagate the gradients with respect to J_k . Denote $T_k = J_k R_{CW}$. Then

$$\frac{\partial \mathcal{L}}{\partial T_k} = \left(\frac{\partial \mathcal{L}}{\partial \Sigma'_k}^\top + \frac{\partial \mathcal{L}}{\partial \Sigma'_k} \right) T_k \Sigma_k \quad (\text{D.77})$$

$$\frac{\partial T_k}{\partial J_k} = R_{CW}. \quad (\text{D.78})$$

This brings us to

$$\frac{\partial \mathcal{L}}{\partial J_k} = \frac{\partial \mathcal{L}}{\partial \tilde{n}_k} \frac{\partial \tilde{n}_k}{\partial J_k} \Big|_{|J_k|} + \frac{\partial \mathcal{L}}{\partial T_k} \frac{\partial T_k}{\partial J_k} \quad (\text{D.79})$$

where the first term is given by (D.62) and (D.70), and the second is given by (D.77) and (D.78). J_k depends on μ_k^t , so we continue to backpropagate.

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \mu_k^t} &= \left[\sum_{i=1}^3 \sum_{j=1}^3 \frac{\partial \mathcal{L}}{\partial J_{k,(i,j)}} \frac{\partial J_{k,(i,j)}}{\partial \mu_{k,n}^t} \right]_{n=\{1,2,3\}}^\top \\ &\quad + \left[0 \quad 0 \quad \frac{\partial \mathcal{L}}{\partial \mu_{k,z}^t} \Big|_{\mathcal{D}_k} \right]^\top, \end{aligned} \quad (\text{D.80})$$

where $J_{k,(i,j)}$ denotes the (i,j) entry of J_k . In practice, each term in (D.80) was computed symbolically, and the result is omitted here for brevity. Finally, we arrive at the gradient of $\mathcal{L}$ with respect to μ_k due to the contribution from J_k as

$$\frac{\partial \mathcal{L}}{\partial \mu_k} \Big|_{J_k} = R_{CW}^\top \frac{\partial \mathcal{L}}{\partial \mu_k^t} \quad (\text{D.81})$$

Gradient of $\hat{\mu}'_k$ The final term to differentiate is $\hat{\mu}'_k$. Immediately, we find that

$$\frac{\partial \hat{\mu}'_k}{\partial \mu_k} = \frac{\partial \hat{\mu}'_k}{\partial \mu'_k} \cdot \frac{\partial \mu'_k}{\partial \mu_k^t} \cdot \frac{\partial \mu_k^t}{\partial \mu_k} \quad (\text{D.82})$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \cdot J_k \cdot R_{CW}. \quad (\text{D.83})$$

This may be used to compute

$$\left. \frac{\partial \mathcal{L}}{\partial \mu_k} \right|_{\hat{\mu}'_k} = \frac{\partial \mathcal{L}}{\partial \hat{\mu}'_k} \cdot \frac{\partial \hat{\mu}'_k}{\partial \mu_k}. \quad (\text{D.84})$$

This gives the final gradient,

$$\frac{\partial \mathcal{L}}{\partial \mu_k} = \left. \frac{\partial \mathcal{L}}{\partial \mu_k} \right|_{\hat{\mu}'_k} + \left. \frac{\partial \mathcal{L}}{\partial \mu_k} \right|_{J_k} \quad (\text{D.85})$$

The above gradients were incorporated into a PyTorch automatic differentiation framework, based largely on the reference implementation provided by [56].

BIBLIOGRAPHY

- [1] Michal Adamkiewicz, Timothy Chen, Adam Caccavale, Rachel Gardner, Preston Culbertson, Jeannette Bohg, and Mac Schwager. Vision-only robot navigation in a neural radiance world. *IEEE Robotics and Automation Letters*, 7(2):4606–4613, 2022.
- [2] Challen Enniful Adu, César E Ramos Chuquiure, Yutong Zhou, Pearl Lin, Ruikai Yang, Bohao Zhang, Shubham Singh, and Ram Vasudevan. Phasing through the flames: Rapid motion planning with the aghf pde for arbitrary objective functions and constraints. *arXiv preprint arXiv:2505.01589*, 2025.
- [3] Challen Enniful Adu, César E. Ramos Chuquiure, Bohao Zhang, and Ram Vasudevan. Bring the heat: Rapid trajectory optimization with pseudospectral techniques and the affine geometric heat flow equation. *IEEE Robotics and Automation Letters*, 10(4):4148–4155, 2025.
- [4] Philip E. Agre and David Chapman. What are plans for? *Robotics and Autonomous Systems*, 6(1):17–34, 1990. Designing Autonomous Agents.
- [5] Mohamadreza Ahmadi, Xiaobin Xiong, and Aaron D Ames. Risk-averse control via cvar barrier functions: Application to bipedal robot locomotion. *IEEE Control Systems Letters*, 6:878–883, 2021.
- [6] Jorge Almeida and Vitor Manuel Santos. Real time egomotion of a nonholonomic vehicle using lidar measurements. *Journal of Field Robotics*, 30(1):129–141, 2013.
- [7] Matthias Althoff. An introduction to cora 2015. In *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems*, 2015.
- [8] Søren Asmussen and Peter W Glynn. *Stochastic simulation: algorithms and analysis*, volume 57. Springer, 2007.
- [9] Dimitri Bertsekas. *Dynamic programming and optimal control: Volume I*, volume 4. Athena scientific, 2012.
- [10] John P Boyd. *Chebyshev and Fourier spectral methods*. Courier Corporation, 2001.
- [11] Zachary Brei, Jonathan Michaux, Bohao Zhang, Patrick Holmes, and Ram Vasudevan. Serving time: Real-time, safe motion planning and control for manipulation of unsecured objects. *IEEE Robotics and Automation Letters*, 9(3):2383–2390, 2024.

- [12] Bruno Brito, Boaz Floor, Laura Ferranti, and Javier Alonso-Mora. Model predictive contouring control for collision avoidance in unstructured dynamic environments. *IEEE Robotics and Automation Letters*, 4(4):4459–4466, 2019.
- [13] Minh Bui, George Giovanis, Mo Chen, and Arrvindh Shriraman. Optimizeddp: An efficient, user-friendly library for optimal control and dynamic programming. *arXiv preprint arXiv:2204.05520*, 2022.
- [14] Russel E Caflisch. Monte carlo and quasi-monte carlo methods. *Acta numerica*, 7:1–49, 1998.
- [15] Francesco Paolo Cantelli. Sui confini della probabilita. In *Atti del Congresso Internazionale dei Matematici: Bologna del 3 al 10 de settembre di 1928*, pages 47–60, 1929.
- [16] Justin Carpentier and Nicolas Mansard. Analytical Derivatives of Rigid Body Dynamics Algorithms. In *Robotics: Science and Systems (RSS 2018)*, Pittsburgh, United States, June 2018.
- [17] Justin Carpentier, Guilhem Saurel, Gabriele Buondonno, Joseph Mirabel, Florent Lamiriaux, Olivier Stasse, and Nicolas Mansard. The pinocchio c++ library : A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives. In *2019 IEEE/SICE International Symposium on System Integration (SII)*, pages 614–619, 2019.
- [18] Timothy Chen, Preston Culbertson, and Mac Schwager. Catnips: Collision avoidance through neural implicit probabilistic scenes, 2023.
- [19] Timothy Chen, Ola Shorinwa, Weijia Zeng, Joseph Bruno, Philip Dames, and Mac Schwager. Splat-nav: Safe real-time robot navigation in gaussian splatting maps, 2024.
- [20] Jared Di Carlo, Patrick M. Wensing, Benjamin Katz, Gerardo Bledt, and Sangbae Kim. Dynamic locomotion in the mit cheetah 3 through convex model-predictive control. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9, 2018.
- [21] S Dieter, M Hiller, and R Baradini. Vehicle dynamics: Modeling and simulation, 2018.
- [22] Anushri Dixit, Mohamadreza Ahmadi, and Joel W Burdick. Risk-sensitive motion planning using entropic value-at-risk. In *2021 European Control Conference (ECC)*, pages 1726–1732. IEEE, 2021.
- [23] Chioniso Dube. Self collision avoidance for humanoids using circular and elliptical capsule bounding volumes. In *2013 Africon*, pages 1–6, 2013.
- [24] Simon Duenser, James M. Bern, Roi Poranne, and Stelian Coros. Interactive robotic manipulation of elastic objects. In *Proc. of the International Conference on Intelligent Robots and Systems (IROS)*, pages 3476–3481, 2018.

- [25] Antonio El Khoury, Florent Lamiroux, and Michel Taïx. Optimal motion planning for humanoid robots. In *2013 IEEE International Conference on Robotics and Automation*, pages 3136–3141, 2013.
- [26] Oscar Eriksson. Fast collision mapping for industrial manipulators: A comparative study of collision mapping methods using sensors, 2024.
- [27] Lawrence C Evans. *Partial differential equations*, volume 19. American Mathematical Society, 2022.
- [28] Parker Ewen, Hao Chen, Seth Isaacson, Joey Wilson, Katherine A Skinner, and Ram Vasudevan. These magic moments: Differentiable uncertainty quantification of radiance field models. *arXiv preprint arXiv:2503.14665*, 2025.
- [29] Parker Ewen, Gitesh Gunjal, Joey Wilson, Jinsun Liu, Challen Enniful Adu, and Ram Vasudevan. Not all actions are created equal: Bayesian optimal experimental design for safe and optimal nonlinear system identification. *arXiv preprint arXiv:2308.01829*, 2023.
- [30] Parker Ewen, Jean-Pierre Sleiman, Yuxin Chen, Wei-Chun Lu, Marco Hutter, and Ram Vasudevan. Generating continuous motion and force plans in real-time for legged mobile manipulation. In *2021 IEEE international conference on robotics and automation (ICRA)*, pages 4933–4939. IEEE, 2021.
- [31] Yinai Fan. *Robot motion planning via curve shortening flows*. PhD thesis, University of Illinois at Urbana-Champaign, 2021.
- [32] Yinai Fan, Shenyu Liu, and Mohamed-Ali Belabbas. Mid-air motion planning of robot using heat flow method with state constraints. *Mechatronics*, 66:102323, 2020.
- [33] Roy Featherstone. *Rigid body dynamics algorithms*. Springer, 2014.
- [34] Martin Fevre, Patrick M. Wensing, and James P. Schmiedeler. Rapid bipedal gait optimization in casadi. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3672–3678, 2020.
- [35] Jaime F Fisac, Andrea Bajcsy, Sylvia L Herbert, David Fridovich-Keil, Steven Wang, Claire J Tomlin, and Anca D Dragan. Probabilistically safe robot planning with confidence-based human predictions. *arXiv preprint arXiv:1806.00109*, 2018.
- [36] Magnus Gaertner, Marko Bjelonic, Farbod Farshidian, and Marco Hutter. Collision-free mpc for legged robots in static and dynamic scenes, 2021.
- [37] Milan Ganai, Sicun Gao, and Sylvia L. Herbert. Hamilton-jacobi reachability in reinforcement learning: A survey. *IEEE Open Journal of Control Systems*, 3:310–324, 2024.
- [38] Ricardo Garcia-Rosas, Jhon M. Portella-Delgado, Ying Tan, and Dragan Nešić. Non-linear observer based control design for an under-actuated compliant robotic hand. In *2016 Australian Control Conference (AuCC)*, pages 21–26, 2016.

- [39] E.G. Gilbert, D.W. Johnson, and S.S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 4(2):193–203, 1988.
- [40] Thomas D Gillespie. Fundamentals of vehicle dynamics. Technical report, SAE Technical Paper, 1992.
- [41] Leonidas J Guibas, An Thanh Nguyen, and Li Zhang. Zonotopes as bounding volumes. In *SODA*, volume 3, pages 803–812, 2003.
- [42] Astghik Hakobyan, Gyeong Chan Kim, and Insoon Yang. Risk-aware motion planning and control using cvar-constrained optimization. *IEEE Robotics and Automation Letters*, 4(4):3924–3931, 2019.
- [43] Ayonga Hereid, Omar Harib, Ross Hartley, Yukai Gong, and Jessy W. Grizzle. Rapid trajectory optimization using c-frost with illustration on a cassie-series dynamic walking biped. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4722–4729, 2019.
- [44] Timothy Hickey, Qun Ju, and Maarten H Van Emden. Interval arithmetic: From principles to implementation. *Journal of the ACM (JACM)*, 48(5):1038–1068, 2001.
- [45] Patrick Holmes, Shreyas Kousik, Bohao Zhang, Daphna Raz, Corina Barbalata, Matthew Johnson Roberson, and Ram Vasudevan. Reachable Sets for Safe, Real-Time Manipulator Trajectory Design. In *Robotics: Science and Systems*, Corvallis, Oregon, USA, 7 2020.
- [46] Seungwoo Hong, Yong Um, Jaejun Park, and Hae-Won Park. Agile and versatile climbing on ferromagnetic surfaces with a quadrupedal robot. *Science Robotics*, 7(73):eadd1017, 2022.
- [47] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013. Software available at <https://octomap.github.io>.
- [48] Alain Horé and Djemel Ziou. Image quality metrics: Psnr vs. ssim. In *2010 20th International Conference on Pattern Recognition*, pages 2366–2369, 2010.
- [49] Jeffrey Ichnowski, Yahav Avigal, Justin Kerr, and Ken Goldberg. Dex-nerf: Using a neural radiance field to grasp transparent objects, 2021.
- [50] Seth Isaacson, Pou-Chun Kung, Mani Ramanagopal, Ram Vasudevan, and Katherine A. Skinner. Loner: Lidar only neural representations for real-time slam. *IEEE Robotics and Automation Letters*, 8(12):8042–8049, 2023.
- [51] Wilson Jallet, Antoine Bambade, Etienne Arlaud, Sarah El-Kazdadi, Nicolas Mansard, and Justin Carpentier. PROXDDP: Proximal Constrained Trajectory Optimization, 2023. <https://inria.hal.science/hal-04332348v1>.

- [52] Lucas Janson, Edward Schmerling, and Marco Pavone. Monte carlo motion planning for robot trajectory optimization under uncertainty. In *Robotics Research*, pages 343–361. Springer, 2018.
- [53] Eduardo S. Zeron Jean B. Lasserre. Solving a class of multivariate integration problems via laplace techniques. *Applicationes Mathematicae*, 28(4):391–405, 2001.
- [54] Saimouli Katragadda, Woosik Lee, Yuxiang Peng, Patrick Geneva, Chuchu Chen, Chao Guo, Mingyang Li, and Guoquan Huang. Nerf-vins: A real-time neural radiance field map-based visual-inertial navigation system, 2024.
- [55] Benjamin Katz, Jared Di Carlo, and Sangbae Kim. Mini cheetah: A platform for pushing the limits of dynamic quadruped control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6295–6301, 2019.
- [56] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), July 2023.
- [57] Tae-Yun Kim, Samuel Jung, and Wan-Suk Yoo. Advanced slip ratio for ensuring numerical stability of low-speed driving simulation: Part ii—lateral slip ratio. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of automobile engineering*, 233(11):2903–2911, 2019.
- [58] Niklas Kochdumper and Matthias Althoff. Sparse polynomial zonotopes: A novel set representation for reachability analysis. *IEEE Transactions on Automatic Control*, 66(9):4043–4058, 2020.
- [59] Shreyas Kousik, Patrick Holmes, and Ram Vasudevan. Safe, aggressive quadrotor flight via reachability-based trajectory design. In *ASME 2019 Dynamic Systems and Control Conference*. American Society of Mechanical Engineers Digital Collection, 2019.
- [60] Shreyas Kousik, Sean Vaskov, Fan Bu, Matthew Johnson-Roberson, and Ram Vasudevan. Bridging the gap between safety and real-time performance in receding-horizon trajectory design for mobile robots. *The International Journal of Robotics Research*, 39(12):1419–1469, 2020.
- [61] Shreyas Kousik, Sean Vaskov, Matthew Johnson-Roberson, and Ram Vasudevan. Safe trajectory synthesis for autonomous driving in unforeseen environments. In *ASME 2017 Dynamic Systems and Control Conference*. American Society of Mechanical Engineers Digital Collection, 2017.
- [62] J.J. Kuffner and S.M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 2, pages 995–1001 vol.2, 2000.

- [63] Scott Kuindersma, Robin Deits, Maurice Fallon, Andrés Valenzuela, Hongkai Dai, Frank Permenter, Twan Koolen, Pat Marion, and Russ Tedrake. Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot. *Autonomous robots*, 40:429–455, 2016.
- [64] Jonas Kulhanek, Songyou Peng, Zuzana Kukelova, Marc Pollefeys, and Torsten Sattler. WildGaussians: 3D gaussian splatting in the wild. *arXiv*, 2024.
- [65] Jean B Lasserre. Simple formula for integration of polynomials on a simplex. *BIT Numerical Mathematics*, 61(2):523–533, 2021.
- [66] An T. Le, Georgia Chalvatzaki, Armin Biess, and Jan R Peters. Accelerating motion planning via optimal transport, 2023.
- [67] Andrew D Lewis. The maximum principle of pontryagin in control and in optimal control. *Handouts for the course taught at the Universitat Politecnica de Catalunya*, 2006.
- [68] Fu Li, Hao Yu, Ivan Shugurov, Benjamin Busam, Shaowu Yang, and Slobodan Ilic. Nerf-pose: A first-reconstruct-then-regress approach for weakly-supervised 6d object pose estimation, 2023.
- [69] Jinsun Liu, Challen Enniful Adu, Lucas Lymburner, Vishrut Kaushik, Lena Trang, and Ram Vasudevan. RADIUS: Risk-Aware, Real-Time, Reachability-Based Motion Planning. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023.
- [70] Jinsun Liu, Yifei Simon Shao, Lucas Lymburner, Hansen Qin, Vishrut Kaushik, Lena Trang, Ruiyang Wang, Vladimir Ivanovic, H Eric Tseng, and Ram Vasudevan. Refine: Reachability-based trajectory design using robust feedback linearization and zonotopes. *IEEE Transactions on Robotics*, 40:2060–2080, 2024.
- [71] Shenyu Liu and Mohamed Ali Belabbas. A homotopy method for motion planning. *arXiv preprint arXiv:1901.10094*, 2019.
- [72] Shenyu Liu, Yinai Fan, and Mohamed-Ali Belabbas. Affine geometric heat flow and motion planning for dynamic systems. *IFAC-PapersOnLine*, 52(16):168–173, 2019. 11th IFAC Symposium on Nonlinear Control Systems NOLCOS 2019.
- [73] Shenyu Liu, Yinai Fan, and Mohamed-Ali Belabbas. Geometric motion planning for affine control systems with indefinite boundary conditions and free terminal time. *arXiv preprint arXiv:2001.04540*, 2020.
- [74] Tomas Lozano-Perez. A simple motion-planning algorithm for general robot manipulators. *IEEE Journal on Robotics and Automation*, 3(3):224–238, 1987.
- [75] J. Y. S. Luh, M. W. Walker, and R. P. C. Paul. On-Line Computational Scheme for Mechanical Manipulators. *Journal of Dynamic Systems, Measurement, and Control*, 102(2):69–76, 06 1980.

- [76] Jan Lunze and Françoise Lamnabhi-Lagarigue. *Handbook of hybrid systems control: theory, tools, applications*. Cambridge University Press, 2009.
- [77] Stefanie Manzing, Christian Pek, and Matthias Althoff. Using reachable sets for trajectory planning of automated vehicles. *IEEE Transactions on Intelligent Vehicles*, 6(2):232–248, 2020.
- [78] Albert W Marshall and Ingram Olkin. Multivariate chebyshev inequalities. *The Annals of Mathematical Statistics*, pages 1001–1014, 1960.
- [79] Carlos Mastalli, Rohan Budhiraja, Wolfgang Merkt, Guilhem Saurel, Bilal Hammoud, Maximilien Naveau, Justin Carpentier, Ludovic Righetti, Sethu Vijayakumar, and Nicolas Mansard. Crocoddyl: An efficient and versatile framework for multi-contact optimal control. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2536–2542, 2020.
- [80] Jonathan Michaux, Qingyi Chen, Challen Enniful Adu, Jinsun Liu, and Ram Vasudevan. Reachability-based trajectory design via exact formulation of implicit neural signed distance functions. *arXiv preprint arXiv:2403.12280*, 2024.
- [81] Jonathan Michaux, Patrick Holmes, Bohao Zhang, Che Chen, Baiyue Wang, Shrey Sahgal, Tiancheng Zhang, Sidhartha Dey, Shreyas Kousik, and Ram Vasudevan. Can’t touch this: Real-time, safe motion planning and control for manipulators under uncertainty, 2023.
- [82] Jonathan Michaux, Seth Isaacson, Challen Enniful Adu, Adam Li, Rahul Kashyap Swayampakula, Parker Ewen, Sean Rice, Katherine A. Skinner, and Ram Vasudevan. Let us make a splan: Risk-aware trajectory optimization in a normalized gaussian splat. *IEEE Transactions on Robotics*, 41:4380–4397, 2025.
- [83] Jonathan Michaux, Yong Kwon, Qingyi Chen, and Ram Vasudevan. Reachability-based Trajectory Design with Neural Implicit Safety Constraints. In *Robotics: Science and Systems XIX*. Robotics: Science and Systems Foundation, July 2023.
- [84] Jonathan Michaux, Adam Li, Qingyi Chen, Che Chen, Bohao Zhang, and Ram Vasudevan. Safe planning for articulated robots using reachability-based obstacle avoidance with spheres. In *Robotics: Science and Systems*, Delft, Netherlands, 7 2024.
- [85] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Proceedings of the European Conference on Computer Vision*, pages 405–421, Cham, 2020. Springer International Publishing.
- [86] Ingram Olkin and Thomas A Trikalinos. Constructions for a bivariate beta distribution. *Statistics & Probability Letters*, 96:54–60, 2015.
- [87] Lorenzo Pareschi, Giovanni Russo, et al. Implicit-explicit runge-kutta schemes for stiff systems of differential equations. *Recent trends in numerical analysis*, 3:269–289, 2000.

- [88] K. B. Petersen and M. S. Pedersen. The matrix cookbook, November 2012.
- [89] Elijah Polak. *Optimization: algorithms and consistent approximations*, volume 124. Springer Science & Business Media, 2012.
- [90] David Powers. Evaluation: From precision, recall and f-factor to roc, informedness, markedness & correlation. *Mach. Learn. Technol.*, 2, 01 2008.
- [91] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10318–10327, 2021.
- [92] Prasanna Sahoo and Thomas Riedel. *Mean value theorems and functional equations*. World Scientific, 1998.
- [93] William E Schiesser. *The numerical method of lines: integration of partial differential equations*. Elsevier, 2012.
- [94] John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33(9):1251–1270, 2014.
- [95] Shubham Singh, Ryan P. Russell, and Patrick M. Wensing. Efficient analytical derivatives of rigid-body dynamics using spatial vector algebra. *IEEE Robotics and Automation Letters*, 7(2):1776–1783, April 2022.
- [96] Shubham Singh, Ryan P Russell, and Patrick M Wensing. On second-order derivatives of rigid-body dynamics: Theory & implementation. *IEEE Transactions on Robotics*, 2024.
- [97] Nils Smit-Anseeuw, Rodney Gleason, Ram Vasudevan, and C David Remy. The energetic benefit of robotic gait selection—a case study on the robot ramone. *IEEE Robotics and Automation Letters*, 2(2):1124–1131, 2017.
- [98] Nils Smit-Anseeuw, C David Remy, and Ram Vasudevan. Walking with confidence: Safety regulation for full order biped models. *IEEE Robotics and Automation Letters*, 4(4):4177–4184, 2019.
- [99] Mark W Spong, Seth Hutchinson, Mathukumalli Vidyasagar, et al. *Robot modeling and control*, volume 3. Wiley New York, 2006.
- [100] Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J. Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, Anton Clarkson, Mingfei Yan, Brian Budge, Yajie Yan, Xiaqing Pan, June Yon, Yuyang Zou, Kimberly Leon, Nigel Carter, Jesus Briales, Tyler Gillingham, Elias Mueggler, Luis Pesqueira, Manolis Savva, Dhruv Batra, Hauke M. Strasdat, Renzo De Nardi, Michael Goesele, Steven Lovegrove, and Richard Newcombe. The Replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797*, 2019.

- [101] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, October 2012.
- [102] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. <https://ompl.kavrakilab.org>.
- [103] Edgar Sucar, Shikun Liu, Joseph Ortiz, and Andrew Davison. iMAP: Implicit mapping and positioning in real-time. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021.
- [104] Balakumar Sundaralingam, Siva Kumar Sastry Hari, Adam Fishman, Caelan Garrett, Karl Van Wyk, Valts Blukis, Alexander Millane, Helen Oleynikova, Ankur Handa, Fabio Ramos, Nathan Ratliff, and Dieter Fox. curobo: Parallelized collision-free minimum-jerk robot motion generation, 2023.
- [105] Andrea Tagliasacchi and Ben Mildenhall. Volume rendering digest (for nerf), 2022.
- [106] Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Justin Kerr, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, Abhik Ahuja, David McAllister, and Angjoo Kanazawa. Nerfstudio: A modular framework for neural radiance field development. In *ACM SIGGRAPH 2023 Conference Proceedings*, SIGGRAPH ’23, 2023.
- [107] Lloyd N Trefethen. *Approximation Theory and Approximation Practice, Extended Edition*. SIAM, 2019.
- [108] Karthik Urs, Challen Enniful Adu, Elliott J Rouse, and Talia Y Moore. Alternative metrics to select motors for quasi-direct drive actuators. *arXiv preprint arXiv:2202.12365*, 2022.
- [109] Karthik Urs, Challen Enniful Adu, Elliott J Rouse, and Talia Y Moore. Design and characterization of 3d printed, open-source actuators for legged locomotion. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1957–1964. IEEE, 2022.
- [110] Gino Van Den Bergen. Proximity queries and penetration depth computation on 3d game objects. In *Game developers conference*, volume 170, page 209, 2001.
- [111] Sean Vaskov. Fast and Safe Trajectory Optimization for Autonomous Mobile Robots using Reachability Analysis. *PhD Thesis*, 2020.
- [112] Sean Vaskov, Shreyas Kousik, Hannah Larson, Fan Bu, James Ward, Stewart Worrall, Matthew Johnson-Roberson, and Ram Vasudevan. Towards provably not-at-fault control of autonomous robots in arbitrary dynamic environments. *Robotics: Science and Systems*, 2019.

- [113] Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106:25–57, 2006.
- [114] Allen Wang, Xin Huang, Ashkan Jasour, and Brian C Williams. Fast risk assessment for autonomous vehicles using learned models of agent futures. *Robotics: Science and Systems*, 2:10, 2020.
- [115] Allen Wang, Ashkan Jasour, and Brian C. Williams. Non-gaussian chance-constrained trajectory planning for autonomous vehicles under agent uncertainty. *IEEE Robotics and Automation Letters*, 5(4):6041–6048, 2020.
- [116] Guangming Wang, Lei Pan, Songyou Peng, Shaohui Liu, Chenfeng Xu, Yanzi Miao, Wei Zhan, Masayoshi Tomizuka, Marc Pollefeys, and Hesheng Wang. Nerf in robotics: A survey, 2024.
- [117] Patrick M. Wensing, Michael Posa, Yue Hu, Adrien Escande, Nicolas Mansard, and Andrea Del Prete. Optimization-based control for dynamic legged robots. *IEEE Transactions on Robotics*, 40:43–63, 2024.
- [118] Hongyi Xu, Thiemo Alldieck, and Cristian Sminchisescu. H-nerf: Neural radiance fields for rendering and temporal reconstruction of humans in motion. *Advances in Neural Information Processing Systems*, 34:14955–14966, 2021.
- [119] Vickie Ye and Angjoo Kanazawa. Mathematical supplement for the `gsplat` library, 2023.
- [120] Lin Yen-Chen, Pete Florence, Jonathan T. Barron, Alberto Rodriguez, Phillip Isola, and Tsung-Yi Lin. iNeRF: Inverting neural radiance fields for pose estimation. In *Proc. of the International Conference on Intelligent Robots and Systems (IROS)*, 2021.
- [121] Ming-Yuan Yu, Ram Vasudevan, and Matthew Johnson-Roberson. Occlusion-aware risk assessment for autonomous driving in urban environments. *IEEE Robotics and Automation Letters*, 4(2):2235–2241, 2019.
- [122] Ming-Yuan Yu, Ram Vasudevan, and Matthew Johnson-Roberson. Risk assessment and planning with bidirectional reachability for autonomous driving. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5363–5369. IEEE, 2020.
- [123] Wentao Yuan, Zhaoyang Lv, Tanner Schmidt, and Steven Lovegrove. Star: Self-supervised tracking and reconstruction of rigid objects in motion with neural rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13144–13152, 2021.
- [124] Bohao Zhang and Ram Vasudevan. Rapid and robust trajectory optimization for humanoids. *arXiv preprint arXiv:2409.00303*, 2024.

- [125] Xiaoshuai Zhang, Sai Bi, Kalyan Sunkavalli, Hao Su, and Zexiang Xu. Nerfusion: Fusing radiance fields for large-scale scene reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5449–5458, 2022.
- [126] Jianhao Zheng, Zihan Zhu, Valentin Bieri, Marc Pollefeys, Songyou Peng, and Iro Armeni. Wildgs-slam: Monocular gaussian splatting slam in dynamic environments. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 11461–11471, 2025.
- [127] Klaus Zindler, Niklas Geiß, Konrad Doll, and Sven Heinlein. Real-time ego-motion estimation using lidar and a vehicle model based extended kalman filter. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 431–438, 2014.
- [128] Matt Zucker, Nathan Ratliff, Anca D. Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M. Dellin, J. Andrew Bagnell, and Siddhartha S. Srinivasa. Chomp: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, 32(9-10):1164–1193, 2013.
- [129] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. Ewa splatting. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):223–238, 07 2002.