Université de Montréal

Differentiable World Programs

par

Krishna Murthy Jatavallabhula

Département d'informatique et de recherche opérationnelle Faculté des arts et des sciences

Thèse présentée en vue de l'obtention du grade de Philosophiæ Doctor (Ph.D.) en Informatique

January 31, 2022

 $^{\odot}$ Krishna Murthy Jatavallabhula, 2022

Université de Montréal

Faculté des arts et des sciences

Cette thèse intitulée

Differentiable World Programs

présentée par

Krishna Murthy Jatavallabhula

a été évaluée par un jury composé des personnes suivantes :

Aaron Courville (président-rapporteur)

Liam Paull

(directeur de recherche)

Christopher Pal (membre du jury)

Michael Kaess

(examinateur externe)

(représentant du doyen de la FESP)

Résumé

L'intelligence artificielle (IA) moderne a ouvert de nouvelles perspectives prometteuses pour la création de robots intelligents. En particulier, les architectures d'apprentissage basées sur le gradient (réseaux neuronaux profonds) ont considérablement amélioré la compréhension des scènes 3D en termes de perception, de raisonnement et d'action. Cependant, ces progrès ont affaibli l'attrait de nombreuses techniques "classiques" développées au cours des dernières décennies. Nous postulons qu'un mélange de méthodes "classiques" et "apprises" est la voie la plus prometteuse pour développer des modèles du monde flexibles, interprétables et exploitables : une nécessité pour les agents intelligents incorporés.

La question centrale de cette thèse est : "Quelle est la manière idéale de combiner les techniques classiques avec des architectures d'apprentissage basées sur le gradient pour une compréhension riche du monde 3D ?". Cette vision ouvre la voie à une multitude d'applications qui ont un impact fondamental sur la façon dont les agents physiques perçoivent et interagissent avec leur environnement. Cette thèse, appelée "programmes différentiables pour modèler l'environnement", unifie les efforts de plusieurs domaines étroitement liés mais actuellement disjoints, notamment la robotique, la vision par ordinateur, l'infographie et l'IA.

Ma première contribution—gradSLAM— est un système de localisation et de cartographie simultanées (SLAM) dense et entièrement différentiable. En permettant le calcul du gradient à travers des composants autrement non différentiables tels que l'optimisation non linéaire par moindres carrés, le raycasting, l'odométrie visuelle et la cartographie dense, gradSLAM ouvre de nouvelles voies pour intégrer la reconstruction 3D classique et l'apprentissage profond.

Ma deuxième contribution - taskography - propose une sparsification conditionnée par la tâche de grandes scènes 3D encodées sous forme de graphes de scènes 3D. Cela permet aux planificateurs classiques d'égaler (et de surpasser) les planificateurs de pointe basés sur l'apprentissage en concentrant le calcul sur les attributs de la scène pertinents pour la tâche.

Ma troisième et dernière contribution—gradSim— est un simulateur entièrement différentiable qui combine des moteurs physiques et graphiques différentiables pour permettre l'estimation des paramètres physiques et le contrôle visuomoteur, uniquement à partir de vidéos ou d'une image fixe. Mots clés. apprentissage profond, robotique, programmation différentiable, SLAM, compréhension de la scène 3D, reconstruction 3D, planification des tâches, réseaux de neurones graphiques, simulation différentiable, rendu différentiable, identification du système, contrôle visuomoteur

Abstract

Modern artificial intelligence (AI) has created exciting new opportunities for building intelligent robots. In particular, gradient-based learning architectures (deep neural networks) have tremendously improved 3D scene understanding in terms of perception, reasoning, and action. However, these advancements have undermined many "classical" techniques developed over the last few decades. We postulate that a blend of "classical" and "learned" methods is the most promising path to developing flexible, interpretable, and actionable models of the world: a necessity for intelligent embodied agents.

"What is the ideal way to combine classical techniques with gradient-based learning architectures for a rich understanding of the 3D world?" is the central question in this dissertation. This understanding enables a multitude of applications that fundamentally impact how embodied agents perceive and interact with their environment. This dissertation, dubbed "differentiable world programs", unifies efforts from multiple closely-related but currently-disjoint fields including robotics, computer vision, computer graphics, and AI.

Our first contribution—gradSLAM—is a fully differentiable dense simultaneous localization and mapping (SLAM) system. By enabling gradient computation through otherwise nondifferentiable components such as nonlinear least squares optimization, ray casting, visual odometry, and dense mapping, gradSLAM opens up new avenues for integrating classical 3D reconstruction and deep learning.

Our second contribution—taskography—proposes a task-conditioned sparsification of large 3D scenes encoded as 3D scene graphs. This enables classical planners to match (and surpass) state-of-the-art learning-based planners by focusing computation on task-relevant scene attributes.

Our third and final contribution—gradSim—is a fully differentiable simulator that composes differentiable physics and graphics engines to enable physical parameter estimation and visuomotor control, solely from videos or a still image.

Keywords. Deep learning, robotics, differentiable programming, SLAM, 3D scene understanding, 3D reconstruction, task planning, graph neural networks, differentiable simulation, differentiable rendering, system identification, visuomotor control

Contents

Résumé	5
Abstract	7
List of tables	15
List of figures	19
List of abbreviations	25
Acknowledgements	27
Chapter 1. Introduction	29
Chapter 2. Background	31
2.1. Computational subsystems of a robot	31
 2.2. Robot perception	33 33 35
 2.3. Robot planning	38 39 41 43 44
 2.4. Robot control 2.4.1. Model-predictive control (MPC) 2.4.2. Differentiable physics for model-based control 	44 45 46
2.5. Computer vision2.5.1. Structure-from-Motion2.5.2. Visual SLAM	46 46 47
2.6. Nonlinear least squares optimization	50

	51
2.6.2. The Gauss-Newton (GN) method	51
2.6.3. The Levenberg-Marquardt (LM) method	51
2.7. Machine learning background	52
2.7.1. Machine learning problem	53
2.7.2. Deep learning: Neural networks	53
2.7.3. Backpropagation	54
2.8. Differentiable programming	55
2.8.1. Computational graphs and explicit autodifferentiation	55
2.8.2. The adjoint method and implicit autodifferentiation	56
2.8.3. Autodifferentiation implementation: Tracing and program transformation .	57
2.9. Computer graphics	57
2.9.1. Realistic image synthesis (Rendering)	57
2.9.2. Differentiable rendering	58
2.10. Context and open challenges	58
Chapter 3. Prologue to Article 1	61
Chapter 4. Article 1: gradSLAM: Dense SLAM meets automatic	
Chapter 4. Article 1: gradSLAM: Dense SLAM meets automatic differentiation	63
Chapter 4. Article 1: gradSLAM: Dense SLAM meets automatic differentiation	63 63
Chapter 4. Article 1: gradSLAM: Dense SLAM meets automatic differentiation Abstract 4.1. Introduction	63 63 63
Chapter 4. Article 1: gradSLAM: Dense SLAM meets automatic differentiation Abstract 4.1. Introduction 4.2. Related Work	63 63 63 65
Chapter 4. Article 1: gradSLAM: Dense SLAM meets automatic differentiation Abstract 4.1. Introduction 4.2. Related Work 4.2.1. Learning-based SLAM approaches	 63 63 63 65 65
Chapter 4. Article 1: gradSLAM: Dense SLAM meets automatic differentiation Abstract A.1. Introduction 4.2. Related Work 4.2.1. Learning-based SLAM approaches 4.2.2. Differentiable visual odometry	 63 63 63 65 65 66
Chapter 4. Article 1: gradSLAM: Dense SLAM meets automatic differentiation Abstract. Abstract. 4.1. Introduction 4.2. Related Work. 4.2.1. Learning-based SLAM approaches. 4.2.2. Differentiable visual odometry. 4.2.3. Differentiable optimization	 63 63 63 65 65 66 66
Chapter 4. Article 1: gradSLAM: Dense SLAM meets automatic differentiation Abstract. Abstract. 4.1. Introduction 4.2. Related Work. 4.2.1. Learning-based SLAM approaches. 4.2.2. Differentiable visual odometry. 4.2.3. Differentiable optimization	 63 63 63 65 65 66 66 67
Chapter 4. Article 1: gradSLAM: Dense SLAM meets automatic differentiation. Abstract. Abstract. 4.1. Introduction. 4.2. Related Work. 4.2.1. Learning-based SLAM approaches. 4.2.2. Differentiable visual odometry. 4.2.3. Differentiable optimization. 4.3. ∇SLAM 4.3.1. Preliminaries: Computational graphs.	 63 63 63 65 65 66 66 67 67
 Chapter 4. Article 1: gradSLAM: Dense SLAM meets automatic differentiation. Abstract. 4.1. Introduction . 4.2. Related Work . 4.2.1. Learning-based SLAM approaches. 4.2.2. Differentiable visual odometry. 4.2.3. Differentiable optimization . 4.3. ∇SLAM. 4.3.1. Preliminaries: Computational graphs. 4.3.2. Method Overview . 	 63 63 63 65 65 66 66 67 67 67
Chapter 4. Article 1: gradSLAM: Dense SLAM meets automatic differentiation Abstract. 4.1. Introduction 4.2. Related Work 4.2.1. Learning-based SLAM approaches 4.2.2. Differentiable visual odometry. 4.2.3. Differentiable optimization 4.3. ∇SLAM 4.3.1. Preliminaries: Computational graphs. 4.3.3. ⊽LM: A differentiable nonlinear least squares solver.	 63 63 63 65 66 66 67 67 67 68
 Chapter 4. Article 1: gradSLAM: Dense SLAM meets automatic differentiation. Abstract. 4.1. Introduction 4.2. Related Work. 4.2.1. Learning-based SLAM approaches. 4.2.2. Differentiable visual odometry. 4.2.3. Differentiable optimization 4.3. ∇SLAM 4.3.1. Preliminaries: Computational graphs. 4.3.2. Method Overview. 4.3.3. ∇LM: A differentiable nonlinear least squares solver. 4.3.4. Differentiable mapping. 	 63 63 63 65 65 66 66 67 67 67 68 69
Chapter 4. Article 1: gradSLAM: Dense SLAM meets automatic differentiation Abstract. Abstract. 4.1. Introduction 4.2. Related Work. 4.2.1. Learning-based SLAM approaches. 4.2.2. Differentiable visual odometry. 4.2.3. Differentiable optimization 4.3. ∇SLAM 4.3.1. Preliminaries: Computational graphs 4.3.2. Method Overview 4.3.3. ⊽LM: A differentiable nonlinear least squares solver 4.3.4. Differentiable mapping 4.3.5. Differentiable map fusion.	 63 63 63 65 66 66 67 67 67 68 69 70

4.4. Case Studies: KinectFusion, PointFusion, and <i>ICP-SLAM</i>	72
4.4.1. KinectFusion	72
4.4.2. PointFusion	73
4.4.3. <i>ICP-SLAM</i>	73
4.5. Experiments and results	74
4.5.1. Differentiable optimization	74
4.5.2. Comparitive analysis of case studies	75
4.5.3. Qualitative results	75
4.5.4. Analysis of Gradients	76
4.5.5. Application: RGB and depth completion	77
4.6. Conclusion	78
Chapter 5. Prologue to Article 2	81
Chapter 6. Article 2: Taskography: Evaluating robot task planning over	
large 3D scene graphs	83
Abstract	83
6.1. Introduction	83
6.2. Related work	85
6.3. Background	86
6.4. Taskography	87
6.4.1. Robot planning domains: Case studies	89
6.4.2. Benchmarking classical and learned planners on TASKOGRAPHY	90
6.5. SCRUB: Principled sparsification of 3DSGs for efficient planning	92
6.5.1. Impact of SCRUB on modern task planners	94
6.6. SEEK: A procedure for efficient learning-based planning	95
6.7. Concluding remarks	96
Chapter 7. Prologue to Article 3	99
Chapter 8. Article 3: gradSim: Differentiable simulation for system identification and visuomotor control	101
Abstract	101
	TOT

8.1. Introduction	
8.2. ∇Sim : A unified differentiable simulation engine	
8.2.1. Differentiable physics engine	
8.2.2. Differentiable rendering engine	
8.3. Experiments	
8.3.1. Physical parameter estimation from video	
8.3.1.1. Rigid bodies (rigid)	
8.3.1.2. Deformable Bodies (deformable)	
8.3.1.3. Smoothness of the loss landscape in ∇Sim	
8.3.2. Visuomotor control	
8.3.2.1. Deformable solids	
8.3.2.2. Cloth (control-cloth)	
8.3.3. Impact of imperfect dynamics and rendering models	
8.3.3.1. Unmodeled dynamics phenomenon	
8.3.3.2. Unmodeled rendering phenomenon	
8.3.3.3. Impact of shading and texture cues	
8.3.3.4. Timing analysis	
8.4 Related work	113
0.4. Itelated work	
8.5. Conclusion	
Chapter 9. Conclusion	117
9.1. Discussion	
9.2 Known limitations and future directions	110
Final remarks	
	101
References	••••••• 121
Appendix A. Taskography: Supplementary material	141
A.1. Benchmark Details	
A.1.1. TASKOGRAPHY domain construction: Parsing Gibson 3DSGs	
A.1.2. Domain specifications	
A.1.3. Symbolic environment interaction	
A.2. SCRUB: Discussion and analysis	144

A.3. Additional results on TASKOGRAPHY domains	
Appendix B. ∇Sim : Supplementary material	
B.1.Differentiable physics engineB.1.1.Physical models	
B.2. Discrete Adjoint Method	
 B.3. Physical Models B.3.1. Finite element method B.3.2. Contact B.3.3. Pendula B.3.4. Incompressible fluids 	
B.4. Source-code transformation for automatic differentiation	
B.5. MPC Controller Architecture	
 B.6. Loss landscapes for parameter estimation of deformable sol B.6.1. Elasticity parameter B.6.2. Loss landscape in PyBullet (REINFORCE) B.6.3. Impact of the length of a video sequence 	lids
B.7. Dataset details	
B.8. BaselinesB.8.1. PyBullet + REINFORCEB.8.2. CNN for direct parameter estimation	
B.9. Compute and timing details	
B.10. Overview of available differentiable simulations	160
B.11. Limitations	
B.12. Broader impact	

List of tables

2.1	Taxonomy of symbolic planning by varying each component in a planning problem	40
4.1	∇LM performs quite similarly to its non-differentiable counterpart, on a variety of non-linear functions, and at various stages of optimization. Here, GD , GN , and LM refer to gradient descent, Gauss-Newton, and Levenberg-Marquardt optimizers respectively.	73
4.2	Performance of ∇ SLAM . The differentiable counterparts perform nearly similar to their non-differentiable counterparts (ATE: Absolute Trajectory Error, RPE: Relative Pose Error).	76
6.1	Taskography benchmark results on select grounded and lifted <i>Rearrangement</i> (Rearr) and <i>Courier</i> (Cour) 3DSG domains. Planning times are reported in seconds and do not incorporate planner-specific domain translation times (factored into planning timeouts). A '-' indicates planning timeouts or failures (10 minutes for optimal planners, 30 seconds for all others). Results are aggregated over 10 random seeds. Optimal task planning is infeasible in larger problem instances or for more complex domains, while most satisficing planners are unable to achieve real-time performance. PLOI [36], a recent learning-based planner consistently performs the best across all domains.	90
6.2	Interestingly, task complexity does not correlate strongly with scene complexity. It is instead determined by the number of operators, and avg. branch factor	91
6.3	Planner statistics evaluated over 70 test problems on $Lifted Rearrangement(5)$	95
6.4	SEEK significantly reduces the number of replanning steps required by state-of- the-art learning-based planners. For each planner, we report average <i>wall time</i> (including translation time)	96
8.1	Mass estimation : ∇Sim obtains <i>precise</i> mass estimates, comparing favourably even with approaches that require 3D supervision (<i>diffphysics</i>). We report the mean abolute error and absolute relative errors for all approaches evaluated 1	107

- 8.3 **Parameter estimation of deformable objects**: We estimate per-particle masses and material properties (for solid def. objects) and per-particle velocities for cloth. In the case of cloth, there is a perceivable performance drop in *diffphysics*, as the center of mass of a cloth is often outside the body, which results in ambiguity. 108

- A.3.4 Performance of planners over the *Rearrangement(k)*-Medium tasks. For all metrics, lower values indicate better performance......146

- A.3.7 Performance of planners over the *Lifted Rearrangement(k)* domains. For all metrics, lower values indicate better performance......147

A.3.8 Performance	of planners over the Lifted $Courier(n, k)$ domains.	For all metrics,
lower values i	indicate better performance	
B.8.1 PyBullet-RE	INFORCE hyperparameters	
B.10.2An overview	of optimizable parameters in ∇Sim . Table column	ns are (in order,
from left to r	right): Initial particle positions (pos), Initial particle	velocities (vel),
Per-particle n	mass (mass), Initial object orientation (rot), Spring res	t lengths (rest),
Spring stiffne	sses (stiff), Spring damping coefficients (damp), Actuat	tion parameters
(actuation), G	Gravity (g), Friction parameters μ , Elasticity parameter	rs (e), External
force parame	ters (ext forces)	

List of figures

2.1	The sense-plan-act (SPA) architecture is the most widely used paradigm for designing subsystems of a robot. A robot observes the environment via its sensors. These sensor observations are fed into the <i>perception</i> subsystem that processes them to produce state estimates. These are subsequently fed into the <i>planning</i> subsystem, which computes a sequence of actions that enable the robot to execute a task. The <i>control</i> subsystem executes these actions by mapping them to actuator	
	commands.	32
2.2	Dynamic Bayes network for a typical state estimation problem in robotics (<i>cf.</i> Thrun <i>et al.</i> [6]). Here, x_* denotes a state, u_* denotes a control action, and z_* denotes a sensor observation	34
2.3	Graphical model illustrating the full SLAM problem for three time steps. (<i>cf.</i> Thrun <i>et al.</i> [6]). Here, x_* denotes a state, u_* denotes a control action, z_* denotes a sensor observation, and m denotes the map. Nodes shaded in gray are observations, while the other nodes are variables that are estimated in SLAM.	36
4.1	∇ SLAM (gradSLAM) is a <i>fully differentiable</i> dense simultaneous localization and mapping (SLAM) system. The central idea of ∇ SLAM is to construct a computational graph representing every operation in a dense SLAM system. We propose differentiable alternatives to several non-differentiable components of traditional dense SLAM systems, such as optimization, odometry estimation, raycasting, and map fusion. This creates a pathway for gradient-flow from 3D map elements to sensor observations (e.g., <i>pixels</i>). We implement differentiable variants of three dense SLAM systems that operate on voxels, surfels, and pointclouds respectively. ∇ SLAM thus is a novel paradigm to integrate representation learning approaches with classical SLAM.	64
4.2	A computational graph. Nodes in red represent variables. Nodes in blue represent operations on variables. Edges represent data flow. This graph computes the function $3(xy + z)$. Dashed lines indicate (local, i.e., per-node) gradients in the	
	backward pass	67

4.3	Computational graph for ∇LM	68
4.4	An example curve fitting problem, showing that ∇ LM performs near-identical to LM, with the added advantage of being fully differentiable	68
4.5	Computation graph for the differentiable mapping module. The uncolored boxes indicate intermediate variables, while the colored boxes indicate processing blocks. Note that the specific choice of the functions for update surface measurement and map fusion depend on the map representation used	69
4.6	Ray differentials : Inset shows the computation graph of the ray value computation. The dashed rectangle is not differentiable, and its derivatives are approximated as shown in Eq 4.3.2	71
4.7	∇LM performs comparably to LM optimizers. In this figure, we show example curve fitting problems from the test suite	74
4.8	Qualitative results: On the living room lr kt0 sequence of the ICL- NUIM dataset [151]. The reconstructions are near-identical to their non- differentiable counterparts. However, distinct from classic SLAM approaches, these reconstructions allow for gradients to flow from a 3D map element all the way to the entire set of pixel-space measurements of that element	75
4.9	Reconstruction obtained upon running the differentiable ICP-Odometry pipeline on a subsection of the rgbd_dataset_freiburg1_xyz sequence	76
4.10	Qualitative results on sequences from the ScanNet [153] dataset. Owing to GPU memory constraints, we use each of the differentiable SLAM systems $(\nabla \text{KinectFusion}, \nabla \text{PointFusion}, \text{ and } \nabla \text{ICP-SLAM})$ to reconstruct parts of the scene. We also show outputs from BundleFusion [154] for reference	77
4.11	In-house sequence collected from an Intel RealSense D435 camera. The reconstruction (right) is obtained by running ∇ PointFusion. Note that we do not perform any noise removal. Differentiable noise filtering is left for future work	78
4.12	Analysis of gradients: ∇ SLAM enables gradients to flow all the way back to the input images. <i>Top</i> : An RGB-D image pair (depth not shown) is passed through ∇ SLAM, and reconstruction error is computed using a precise fused map. Backpropagation passes these gradients all the way back to the depth map (blue map). <i>Bottom</i> : An explicit occluder added to the center of the RGB-D pair. This occluder distorts the construction by creating a gaping hole through it. But, using	

8.7	Results on control-cloth environment (5 randomseeds; each controls the dimensions and initial/target poses of the cloth). <i>diffphysics</i> converges to a suboptimal solution due to ambiguity in specifying the pose of a cloth via its center-of-mass. ∇Sim solves the environment using a single target image 11	11
8.8	Including shading and texture cues lead to faster convergence. Inset plot has a logarithmic Y-axis	13
B.3.1	Mesh Discretization: We use triangular (a) and tetrahedral (b) FEM models with angle-based and volumetric activation parameters, α . These mesh-based discretizations are a natural fit for our differentiable rasterization pipeline, which is designed to operate on triangles	52
B.3.2	Contact Model : To model non-interpenetration constraints we use a relaxed model of contact that replaces a delta function with a linear hinge corresponding to a quadratic penalty energy (a). To model friction we use a relaxed Coulomb model, that replaces the step function with a symmetric hinge (b)	53
B.5.3	Our simple network architecture used the for control-walker and control-fem tasks	55
B.6.4	Loss Landscapes: (left) when optimizing for the elasicity parameters of a deformable FEM solid. Both the Lamé parameters λ and μ are set to 1000, where the MSE loss has a unique, dominant minimum. (right) when optimizing for the mass, the reward (negative normalized MSE) has a maximum close to the ground truth maximum but the negative log likelihood of each mass sample that's multiplied with the reward only shows a local minimum that's sensitive to the center of the current mass estimate	57
B.6.5	Impact of the length of a video sequence on the loss landscape. Notice how the loss landscape is much steeper for smaller videos (e.g., MSE of first and last frames). Nonetheless, all cases have a smooth loss landscape with the same unique minimum.	57
B.7.6	Objects used in our rigid-body experiments. All of these meshes have been simplified to contain 250 or fewer vertices, for faster collision detection times	58

List of abbreviations

3DSG	3D Scene Graph
AI	Artificial Intelligence
BoW	Bag-of-Words
CNN	Convolutional Neural Network
DAG	Directed Acyclic Graph
DL	Deep Learning
DNN	Deep Neural Network
EKF	Extended Kalman Filter
FD-Plan	Fast-Downward Planner
FF-Plan	Fast-Forward Planner
GD	Gradient Descent
GN	Gauss-Newton
GNN	Graph Neural Network
HSP	Heuristic Search Planner
ICP	Iterative Closest Point
KF	Kalman Filter
LM	Levenberg-Marquardt
LQR	Linear Quadratic Legulator
MAP	Maximum a Posteriori
MC	Monte Carlo
MCMC	Markov Chain Monte Carlo
MLE	Maximum Likelihood Estimation
MLP	Multi-layer Perceptron
MSE	Mean Squared Error
NN	Neural Network

PDF	Probability Density Function
PDDL	Planning Domain Definition Language
PMF	Probability Mass Function
RGB	Red-Green-Blue
RGB-D	Red-Green-Blue - Depth
SGD	Stochastic Gradient Descent
SAM	Smoothing and Mapping
iSAM	Incremental Smoothing and Mapping
SfM	Structure from Motion
SLAM	Simultaneous Localization and Mapping
SPA	Sense-Plan-Act
STRIPS	Stanford Research Institute Problem Solver
TAMP	Task and Motion Planning
VO	Visual Odometry
VPR	Visual Place Recognition

Acknowledgements

I would like to acknowledge a number of people who have shaped my research trajectory over the last few years.

I am indebted to Prof. Liam Paull, my advisor, who took a punt on my cold email and welcomed me to his freshly-formed group at Universite de Montreal and Mila. Liam is the most supportive advisor one could ever hope for – he has always encouraged me in every aspect of grad school life, and went above and beyond to support me at the times that I really needed it. Liam encouraged me to embrace the idea of slow science; allowing me to spend much of my first year-and-a-half reading literature from the 1980s and 1990s, which to this day inspires much of what I work on. He has been very accommodating when I ventured into research areas that were probably not the lab's primary focus at the time, and has been extremely supportive of my collaborations with other researchers over the years. He has always advocated for a healthy work-life balance for his students and has fostered an extremely friendly and collaborative culture within his research group. His mentorship qualities have set a really high bar that I someday hope to emulate.

I have been fortunate to work very closely with Prof. Florian Shkurti (University of Toronto) and Prof. Derek Nowrouzezahrai (McGill University) over the last two years of my PhD. Both Florian and Derek have offered me invaluable mentorship and advice and welcomed me to their groups with open arms. They were my (un)official co-supervisors who played a crucial role in shaping my work. I am also fortunate to have worked closely with Prof. Sanja Fidler over the course of a year-and-a-half. The association with her group at NVIDIA marked a turning point in my PhD. Sanja always gave me thoughtful feedback on how I could frame and present projects better. Her group had several other students and interns from whom I learned a lot, including Edward Smith, Jun Gao, Wenzheng Chen, and Daiqing Li, Amlan Kar, and David Acuna.

I thoroughly enjoyed my time at Liam's group. Vincent Mai, Manfred Diaz, and Bhairav Mehta have always helped my work by indulging in brainstorming and constructive criticism. A special mention to Vincent who has helped me with a number of document translations to French, including the abstract of this dissertation. I quite enjoyed hanging out at the lab with other masters students at the group (Sai Krishna, Gunshi Gupta, Dhaivat Bhatt, Dishank Bansal).

I had great fun collaborating with Ganesh Iyer on gradSLAM and other subsequent research efforts. I also thorougly enjoyed mentoring (and learning from) the rest of the gradSLAM team: Soroush Saryazdi, Nikhil Keetha, Sree Sai Harsha, and Abhishek Jain.

I would also like to acknowledge the support of all the authors on the gradSim paper, for their perseverance and support over nearly two years. In particular, Florian Shkurti, who remained the most enthusiastic of the lot despite several early setbacks, and Florian Golemo and Vikram Voleti, who were incredibly helpful in getting the submission to the finish line. A special mention to Miles Macklin, who has been an amazing collaborator and him joining the project provided a major impetus.

I found yet another fantastic collaborator in Chris Agia. He made working on the Taskography project a lot of fun. I am also grateful to all of the other collaborators (Mohamed, Vibhav, Ondrej, Mustafa) for supporting the project.

I have enjoyed co-organizing multiple workshops at ML, CV, and robotics venues alongside Kelsey Allen, Victoria Dean, Johanna Hansen, Mandana Samiei, Aditya Kusupati, Breandan Considine, Amy Tabb, Anna Rogers, Bhairav Mehta, Khimya Khetarpal, Sara Hooker, Tegan Maharaj, Edward Smith, Vikram Voleti, Rana Hanocka, Lior Yariv, Yiyi Liao, Shalini De Mello, Kevin Smith, Rika Antonova, and Fish Tung.

I have been fortunate to work with a wonderful set of interns over the course of my PhD. My mentees (Sarthak, Mark, Nikhil, Harsha, Shivam, Junaid, Shubodh, Mayank, Abhishek, Saksham, Gokul, Karnik, SaiShankar, Kaustubh, Swapnil) have helped me broaden my research perspectives and have always brought a fresh wave of enthusiasm to all projects.

I would also like to express my gratitude to Prof. Avinash Gautam (BITS Pilani, India) and Prof. Madhava Krishna (IIIT Hyderabad, India) who mentored me during my undergraduate and masters studies respectively, and helped me find my feet in robotics research.

This section would be incomplete if I did not acknowledge projects that I was enthusiastic about, but did not work out. By design, this dissertation only presents the projects where everything went well. The projects that shaped my research also include several others that never really saw the light of day.

Finally, I would like to thank my family and in particular my wife, Swetha, for their unconditional love and support.

Chapter 1

Introduction

The last decade (2012 - 2021) has been a defining one for modern artificial intelligence (AI). Significant progress has been achieved in the areas of computer vision, speech processing, and natural language processing. Underlying this progress is the resurgence of *deep learning* – the idea that representations for several tasks involving machine intelligence may be *learned* from large volumes of carefully curated data by leveraging multiple layers of sequential computations. While many of the ideas behind deep learning have persisted for several decades, key driving factors – including the widespread availability of large-scale data-parallel compute infrastructures, stochastic numerical optimization routines, and the use of very large scale computational models as general purpose function approximators – have led to the rapid progress and commercialization of AI technologies at an unprecedented scale and rate.

While modern learning-based systems have shown experimentally promising results in visuolingistic pattern recognition, these systems fail in unintuitive and unpredictable ways when applied to problems in robotics (embodied intelligence). A key reason for such failures is that most learning-based approaches do not work in conjunction with "classical"¹ approaches to robotics, instead attempting to replace them in entirety. We posit that a blend of "classical" and "learned" methods is the most promising path to developing flexible, interpretable, and actionable models of the world: a necessity for intelligent embodied agents².

"What is the ideal way to combine classical techniques with gradient-based learning architectures for a rich understanding of the 3D world?" is the central question that we address in this dissertation. Understanding this enables a multitude of applications that fundamentally impact how embodied agents perceive and interact with their environment. A core, recurring idea in this dissertation is to leverage our prior knowledge of various phenomena such as 3D geometry, image formation, and dynamics to design effective learning methods for embodied perception, reasoning, and action.

¹The adjective "classical" is not to be misconstrued as "old" or "outdated". We use this to denote approaches that do not leverage modern machine learning; relying rather on specialized domain knowledge. ²In this dissertation, we use the terms *robot* and *embodied agent* interchangeably.

As a general recipe for designing intelligent embodied agents, we present multiple strategies that we jointly refer to as **differentiable world programs**: a design methodology where we encode our knowledge of the world into a *differentiable program*, and leverage (stochastic) gradient-based optimization to infer properties of the program's inputs and/or parameters. Explicitly encoding inductive biases into such a program enables the design of more efficient learning solutions, allowing classical and modern techniques to work in conjunction.

This dissertation is a culmination of three research articles, each bringing together components from classical robotics and modern gradient-based learning, and other areas including computer vision and graphics. Together these contributions enhance all the components of a typical robotics stack: perception (state estimation), reasoning (planning), and action (control).

- (1) The first article, ∇SLAM (gradSLAM), is a fully differentiable dense simultaneous localization and mapping (SLAM) system. By enabling gradient flow through otherwise non-differentiable components such as nonlinear least squares optimization, ray casting, visual odometry, and dense mapping, ∇SLAM opens up new avenues for integrating classical 3D reconstruction and deep learning for robot perception.
- (2) The second article, **Taskography**, addresses the problem of efficient robot task **planning** in large building-scale environments represented as 3D scene graphs (3DSGs) hierarchical symbolic-geometric graphical representations. We show that, for a general class of robotic planning problems with far-reaching applications, planning over large 3DSGs is impractical. We propose a task-conditioned 3DSG sparsification scheme that allows off-the-shelf classical task planners to match (and surpass) state-of-the-art learning-based planners.
- (3) The third and final article, ∇Sim (gradSim), infuses inductive priors from physics simulation and image synthesis (graphics) to solve system identification and visuomotor **control** problems. Our main contribution is to build a fully differentiable simulator that composes differentiable physics and graphics engines, enabling gradient-based optimization to solve challenging high-dimensional control problems solely from videos or a still image.

A prologue accompanies each article, contextualizing the contribution of the author of this dissertation. A broader discussion (including known critiques) of the ideas presented in each article is deferred to the final chapter.

Background

This dissertation brings together ideas from several communities including robotics, computer vision, computer graphics, and machine learning. In this chapter, we review the core ideas from each of these fields to establish context for the articles that follow. We also discuss key challenges surrounding existing work in these areas, and how these challenges may be overcome by blending knowledge from classical techniques with modern learning approaches.

2.1. Computational subsystems of a robot

Any intelligent robotic system needs to carry out a large number of computations to execute meaningful real-world tasks. Some of these computations are specific to the robotic system. E.g., sensing and actuation depend on the electromechanical composition of the robot. However, a vast majority of computations are common capabilities that are required in a variety of robotic systems. E.g., robots operating within an indoor environment and robots operating in the wild require both a mental representation of the world and the reasoning capabilites that enable them to complete a task. To facilitate the design and study of complex robotic systems and to maximize modularity and reuse, several computational models of robots have been proposed. These models abstract away the details of each computation and organize workflow into *subsystems* (also called layers), providing a framework for the holistic analysis and design of complex robotic systems.

Sense-Plan-Act (SPA): The most prominent computational model of a robot dates back to the late 1980s, proposed in Firby [1]. This architecture, commonly referred to as the *sense-plan-act* (SPA) architecture, is a three-layer architecture [2] comprising the perception, planning, and control subsystems. The *sense* (perception) subsystem observes the environment and produces *states*, which are passed to the *plan* subsystem. This results in a sequence of actions that accomplish the task, which are converted to low-level actuator commands by the *act* (control) subsystem. This affects the state of the robot and/or the environment, and typically results in a new set of observations at the next time step. At



Fig. 2.1. The sense-plan-act (SPA) architecture is the most widely used paradigm for designing subsystems of a robot. A robot observes the environment via its sensors. These sensor observations are fed into the *perception* subsystem that processes them to produce state estimates. These are subsequently fed into the *planning* subsystem, which computes a sequence of actions that enable the robot to execute a task. The *control* subsystem executes these actions by mapping them to actuator commands.

this point, the sense-plan-act loop is repeated, but with the new sensor observations, and usually a stored history of past states (or minimally, the most recent state). In this view, sense-plan-act is a conceptually simple architecture; the control flow across all modules and time steps is linear. We illustrate this architecture in Fig. 2.1.

The linear nature of the sense-plan-architecture also brings in a few shortcomings. First, errors in upstream modules and/or earlier time steps compound, leading to irrecoverable, catastrophic failures. Second, there exists no active communication among the modules. Downstream modules cannot directly influence the functioning of upstream modules; they may only do so indirectly. E.g., a planing failure due to incorrect state estimates is not immediately perceived by the perception subsystem; only after passing through the control subsystem and sensor measurements is the perception subsystem invoked again.

Subsumption architecture: These shortcomings have resulted in a number of other architectural propositions. Among these other propositions, the subsumption architecture [3] has received a lot of attention in the 1980s and the 1990s. The key idea of the subsumption architecture was to view a robot as comprising several *behaviors* that are organized hierarchically with the higher levels *subsuming* lower levels. Subsumption in this case means that higher levels are able to combine the collective behaviors of the lower layers to achieve a higher-order behavior that is far more complex compared to the original, much simpler, behaviors. Subsumption of lower level behaviors is triggered by the current state of the robot and the environment to achieve a task in the most efficient manner possible.

Despite the emergence of subsumption and other behavior oriented architectures [4, 5], sense-plan-act remains the dominant computational model, presumably due to its simplicity.

2.2. Robot perception

The first subsystem in the SPA model is the *perception* module. This module receives sensor observations from both proprioceptive and exteroceptive sensors onboard the robot. The perception module is responsible for processing these sensor observations¹ to produce *state estimates*; a representation of the robot (and potentially, environment) configuration.

State: A state is a set of variables that captures the configuration of a robot and/or an environment. In the context of this dissertation, states comprise all variables of interest that are relevant to a specified task. For example, the state of a mobile robot base moving on a 2D planar surface may be uniquely and unambiguously specified by estimating its pose (position and orientation) with respect to a fixed inertial frame.

Complete states and Markov chains: In the context of robot state estimation, it is helpful to define the notion of a *complete state*. Formally, a state x_t is termed complete if future variables of interest (i.e., time step t + 1) are conditionally independent of variables from the past (i.e., time step t - 1) given x_t . A temporal process involving complete states is referred to as a *Markov chain*. We assume, unless otherwise stated, that all states used in this dissertation are complete states and that their evolution produces a Markov chain.

2.2.1. State estimation in robotics

In this section, we briefly review the various components of a state estimation problem as used in modern probabilistic robotics. For a comprehensive treatment of Bayesian inference in robotics, we refer the interested reader to Thrun *et al.* [6].

Preliminaries and notation: Let x_t denote the state (of the robot and/or environment) at time t. At each time step, the robot acts according to the sense-plan-act cycle. It first receives a sensor observation z_t and processes this observation to produce a control action u_{t+1} . The execution of this control action results in a new state x_{t+1} . This process is repeated across time steps², and the dependency structure among these variables is depicted in a dynamic Bayes network in Fig. 2.2. This Bayes network is typically factorized into two distinct conditional densities that admit a recursive definition.

Motion model: The motion (or state transition) model is a conditional density that indicates the probability of arriving at a state x_{t+1} from a state x_t , under an applied control input u_{t+1} . This is denoted $p(x_{t+1} | x_t, u_{t+1})$. Note that the assumption that x_t is a complete

¹We use the term *measurement* and *observation* interchangeably.

²In this dissertation, we assume that all events of interest occur at discrete time intervals. We use the notation t + 1 to indicate the immediate discrete time step after t.



Fig. 2.2. Dynamic Bayes network for a typical state estimation problem in robotics (*cf.* Thrun *et al.* [6]). Here, x_* denotes a state, u_* denotes a control action, and z_* denotes a sensor observation.

state allows us to ignore the influence of all other upstream variables $(u_t, z_t, x_{t-1}, u_{t-1}, z_{t-1})$. We use the notation $x_{0:t}$ to refer to all states x_0, x_1, \dots, x_t .

Observation model: The observation (or measurement) model, denoted $p(z_t | x_t)$, is the conditional probability of obtaining an observation z_t from a state x_t .

Bayesian state estimation problem: A typical state estimation problem involves estimating the state x_t conditioned on the set of all control inputs, past states, and observations. Mathematically, this involves estimating the conditional probability $p(x_t | x_{0:t-1}, z_{1:t-1}, u_{1:t})$. This is also referred to as *filtering*, as only the most recent state is recovered. An estimation problem recovering the entire sequence of states x_0, x_1, \dots, x_t is referred to as *smoothing*.

Recursive Bayesian estimation: A general framework that leverages probabilistic motion and observation models for state estimation may be obtained by applying the Bayes' rule to the graphical model from Fig. 2.2. The recursive Bayes filter (or simply, "Bayes filter') is given by the following equations.

$$p(x_t \mid z_{1:t-1}, u_{1:t}) = \int \underbrace{p(x_t \mid x_{t-1}, u_t)}_{\text{Motion model}} p(x_{t-1} \mid z_{1:t-1}) \, dx_{t-1}$$

$$p(x_t \mid z_{1:t}, u_{1:t}) \propto \underbrace{p(z_t \mid x_t)}_{\text{Observation model}} p(x_t \mid z_{1:t-1}, u_{1:t})$$
(2.2.1)

While recursive Bayesian estimation is computationally infeasible for arbitrary conditional densities, specific choices of motion and observation models enable tractable algorithms.

Gaussian filtering: An extremely popular realization of the Bayesian estimation framework is the family of Gaussian filtering approaches. The conditional likelihoods in this context are assumed to be (multivariate) Gaussian distributions. This family of approaches includes the Kalman filter and variants [7]. Kalman filtering is a popular Gaussian filter for the setting where both the motion and observation models are linear systems with additive white Gaussian noise. For nonlinear motion and observation models, variants such as the extended Kalman filter (EKF), iterated EKF, or the unscented Kalman filter (UKF) [8] may be used. Expressing posterior distributions using a Gaussian density often results in analytical expressions, rendering Gaussian filtering extremely efficient in practice.

Non-parametric filtering: A key weakness of Gaussian filtering approaches is the premise that the posterior distributions are assumed Gaussian. If the modeling assumptions do not hold, the quality of the obtained state estimates tends to be poor. Non-parametric filtering techniques instead attempt to estimate the posterior distribution without having to necessarily produce an analytical expression for the estimator. Two important non-parametric filtering approaches include *histogram filtering* and *particle filtering* [9]. A histogram filter decomposes the state space and estimates the density in each sub-region, resulting in a histogram depicting the posterior. A particle filter, on the other hand, approximates the state space using several random samples drawn from the approximate posterior, which is refined simultaneously with the samples.

2.2.2. Simultaneous localization and mapping (SLAM)

A particular class of state estimation problems that has received tremendous attention over the last four decades is the simultaneous localization and mapping (SLAM) problem. Simply put, SLAM involves *simultaneously* estimating the state of the robot and the environment. The state of the robot is typically represented as a set of one or more kinematic poses that unambiguously specify its configuration. The state of the environment (also known as the map), however, may be specified in several forms; giving rise to a broad spectrum of SLAM approaches. This question of map representation has been the centerpiece of much of SLAM research during the 1980s through the early 2000s.

SLAM – **Problem formulation**: The probabilistic simultaneous localization and mapping problem involves estimating the current state of the robot \mathbf{x}_t at time t and a map of the environment \mathbf{m} , given a sequence of control inputs $\mathbf{u}_{1:t}$ and sensor observations $\mathbf{z}_{1:t}$. Formally, SLAM entails estimating the posterior density $p(\mathbf{x}_t, \mathbf{m} | \mathbf{z}_{1:t}, \mathbf{u}_{1:t})$. A graphical model depicting the SLAM problem is presented in Fig. 2.3

The difficulty in treating SLAM as a regular state estimation problem arises from the fact that both localization and mapping are intertwined sub-problems. The *localization* problem is a state estimation problem that computes $p(\mathbf{x}_t \mid \mathbf{m}, \mathbf{z}_{1:t}, \mathbf{u}_{1:t})$ – the robot's state with respect to a known environment state, control inputs, and sensor observations. The *mapping* problem is a state estimation problem that computes $p(\mathbf{m} \mid \mathbf{x}_t, \mathbf{z}_{1:t}, \mathbf{u}_{1:t})$ – the environment state (the map) with respect to a known robot state, control inputs, and sensor observations. This induces cyclical dependencies, with the solution to each individual estimation sub-problem depending on the solution to the other sub-problem.



Fig. 2.3. Graphical model illustrating the full SLAM problem for three time steps. (*cf.* Thrun *et al.* [6]). Here, x_* denotes a state, u_* denotes a control action, z_* denotes a sensor observation, and *m* denotes the map. Nodes shaded in gray are observations, while the other nodes are variables that are estimated in SLAM.

Anatomy of a SLAM system: SLAM systems can be thought of as comprising two primary components – a *frontend* and a *backend*. The frontend is responsible for converting sensor inputs into observations that can be fed into an observation models for specifying a state estimation problem. The backend is responsible for solving the state estimation problem thus constructed. In a SLAM system, the frontend is usually sensor-dependent; e.g., SLAM systems that use images captured from a camera as input require a different frontend processing scheme compared to systems that use scans acquired from a laser rangefinder. However, the backends may be shared across a number of SLAM systems (sometimes with a little reformulation to handle newer sensor types and data rates). Based on the forumlation used, SLAM backends may be *filtering-based* or *optimization-based*.

Filtering-based SLAM backends: The beginnings of SLAM research can be traced back to the mid-1980s, when several researchers began exploring how the relationships between a robot and its environment could be described. At the time, Kalman filters had already demonstrated great promise in the areas of state estimation. Naturally, the earliest approaches to SLAM leveraged these filtering approaches to estimate the robot and environment states. However, the main challenge in SLAM as opposed to typical filtering problems was that the estimation problem required an ever-increasing state vector with newer observations streaming in at potentially every time step. This led to an increased focus on designing efficient filtering approaches, and several alterntatives such as unscented Kalman filters, information filters,
and particle filters were applied to solve the SLAM problem. Several research initiatives also began tackling the problem of *data association* – determining whether a new observation corresponds to a previously observed entity in the environment. A comprehensive survey of the first 20 years of SLAM research (dating from the mid 1980s through the early 2000s) may be found in Durrant-Whyte *et al.* [10].

Optimization-based SLAM backends: A major paradigm shift in SLAM research occured with the seminal work formulating the estimation problem in SLAM as a graph optimization problem [11]. Gutmann *et al.* [12] presented an incremental solution to this problem in 1999. However, it was not until 2006 that a computationally efficient approach was proposed to solve the full SLAM problem. Dellaert and Kaess [13] presented square-root smoothing and mapping, which introduces the formalism of factor graphs to SLAM backend research. The estimation problem in SLAM is modeled as a factor graph, with each node indicating a variable to be estimated, and edges (called *factors*) indicating a constraint between nodes. These factors are constructed from the sensor observations and data association provided by the frontend. While the original solution to this problem presented in [13] is an offline (batch-mode) solution, an incremental strategy that relied on numerical recipies for incremental, sparse QR factorization was presented by Kaess and Dellaert in 2007 [14]. To date, graph optimization approaches such as iSAM [14] and g20 [15] continue to be the dominant choice for SLAM backends, enabling a large number of real-time and large-scale SLAM systems.

SLAM backends as MAP estimators³: As discussed above, SLAM is typically formulated as a graph optimization problem, over the probabilistic graphical model illustrated in Fig. 2.3. Formally, if \mathcal{X} denotes all state variables of interest (robot states $x_t \forall t$ and map m), the mode of the posterior distribution specified by this graphical model is obtained by maximum a posteriori (MAP) estimation.

$$\mathcal{X}_{MAP} = \underset{\mathcal{X}}{\operatorname{argmax}} \quad p(\mathcal{X} \mid \mathcal{Z}) = \underset{\mathcal{X}}{\operatorname{argmax}} \quad p(\mathcal{Z} \mid \mathcal{X}) \quad p(\mathcal{X})$$
(2.2.2)

If we assume that all measurements $\mathcal{Z} = \{z_1, \dots, z_M\}$ are independent, and that each measurement $z_m, m \in \{1 \dots M\}$ corresponds to a state x_{z_m} , we obtain an efficient factorization of the posterior as follows.

$$\mathcal{X}_{MAP} = \underset{\mathcal{X}}{\operatorname{argmax}} \quad p(\mathcal{X}) \prod_{m=1}^{M} p(z_m \mid \mathcal{X}_{z_m})$$
(2.2.3)

In the case where all observation likelihoods are modelled as Gaussian densities, the above estimation problem can be reduced to a (nonlinear) least squares problem. Let us assume that each conditional observation likelihood $p(z_m | \mathcal{X}_{z_m})$ is a Gaussian with covariance matrix Ω_m and has a mean equal to the sensor observation z_m . Further, we assume that h() is a

³This exposition is drawn largely from Cadena *et al.* [16].

measurement model that maps sensor observations to states. The probability density function (PDF) of the observation likelihood can be written as

$$p(z_m | \mathcal{X}_{z_m}) \propto \exp\left(-h(\mathcal{X}_{z_m}) - z_m\right)^T \Omega_m^{-1} \left(h(\mathcal{X}_{z_m}) - z_m\right) = \exp\left(-\|h(\mathcal{X}_{z_m}) - z_m\|_{\Omega_m}^2\right) \quad (2.2.4)$$

Rewriting the MAP estimation as a negative log-likelihood minimization, we obtain a nonlinear least squares problem.

$$\mathcal{X}_{MAP} = \underset{\mathcal{X}}{\operatorname{argmin}} \quad -\log\left(p(\mathcal{X})\prod_{m=1}^{M}p(z_m \mid \mathcal{X}_{z_m})\right)$$

$$\mathcal{X}_{MAP} = \underset{\mathcal{X}}{\operatorname{argmin}} \quad \sum_{m=1}^{M}\|h(\mathcal{X}_{z_m}) - z_m\|_{\Omega_m}^2$$
(2.2.5)

We review optimization algorithms for nonlinear least-squares problems like these in Sec. 2.6.

SLAM frontends: The primary objective of a SLAM frontend is to setup the MAP estimation problem that is solved by the backend. This entails processing raw sensor observations, associating them with map elements across time, and converting them into a form that is more amenable to nonlinear least-squares optimization. For instance, a vision-based SLAM frontend needs to take in raw images, process them to produce map attributes, track these map attributes across time, and convert them into appropriate constraints (factors) that capture the relationship between the 3D scene captured in the images. Depending on the choice of sensor observations, a number of frontend possibilities exist for a SLAM system designer. In this dissertation, we primarily focus on *visual SLAM*, where images captured by cameras are the primary source of observations made on the operating environment. In particular, we use specific kinds of cameras such as the Microsoft Kinect and Intel RealSense that produce per-pixel depth measurements in addition to color intensities. We review visual SLAM in section 2.5.2.

We note that there exists a large body of literature on leveraging alternate forms of sensing, such as aural (sound) sensors, tactile (touch) sensors, and olfactory (smell) sensors, that are outside the scope of the articles presented here.

2.3. Robot planning

Given perceptual inputs (state estimates), this next subsystem of the SPA model plans a sequence of robot actions that achieve a task specification. Depending on the granularity of this plan, planning techniques are classified as either *symbolic* (task, or high-level) planners or *motion* (low-level) planners. In this dissertation, our focus is solely on symbolic planning. This section introduces symbolic planning problems and how they are specified and solved in modern robotic systems. Towards the end, we briefly discuss motion planning approaches and efforts that attempt to integrate both symbolic and motion planners.

2.3.1. Symbolic planning

Many robotic tasks involve the interplay of high-level *concepts* or *facts* that are hard (in some cases, impossible) to specify mathematically. For example, if a robot is assigned the task of pouring coffee into a cup, it needs to know about the existence of a cup, that a cup is something you *can pour* coffee into, and that the cup *is full* when it holds a certain volume of coffee. To deliver this cup to a person, a robot needs to know other things like whether it *can pick* up the cup, and whether it *can hand over* the cup to the person. The core entities in such a planning problem are *symbols* (e.g., the robot, cup, coffee). Symbols often have *properties* (*predicates*) that indicate how they interact with other symbols (e.g., *is full, can pour, can hand over*). Symbolic planning problems—also referred to as task planning problems—entail prescribing a sequence of high-level (symbolic state) transitions that achieve a set of goal conditions. Such planning problems have existed in much of the classical AI literature, dating back to the 1960s. We now introduce a few concepts associated with symbolic planning, which will enable us to formally specify a planning problem⁴.

- **Object**: An object is an atomic entity in a planning problem that can be acted upon. We use the term *ground* objects to refer to specific objects (e.g., a specific apple on a table), and the term *lifted* objects to refer to object categories as a whole (e.g., the concept "apple").
- Agent: An agent is an entity that acts in the environment, tasked with the goal of following instructions generated by a planning algorithm.
- **Property**: A property is a function defined over one or more objects. A property may be unary (e.g., *weight* of an apple), binary (e.g., an apple *is on the* table), or n-ary.
- **Predicate**: The term predicate refers to a subclass of a properties that are booleanvalued. In this work, we restrict ourselves to unary and binary predicates.
- State: A state is an assignment of values to all possible properties over ground objects.
- Action: An action is an operation that changes the state of one or more objects or agents. Actions may be grounded or object-parameterized. A ground action is an action applied to a ground object instance. An object-parameterized (or lifted) action is bound to an object only when the action is evaluated by instantiation with a ground object instance.
- Transition model: A transition model is a mapping from a state s and an action a to a subsequent state s'. Optionally, transition models may also store a *cost* c for each transition, denoted as c(s, a, s').

 $^{^{4}}$ In this dissertation, we will use the term *planning* to mean "symbolic planning", commonly referred to as "task planning".

Symbolic planning problem: A symbolic planning problem Π is a tuple $\langle \mathcal{O}, \mathcal{P}, \mathcal{A}, \mathcal{T}, \mathcal{C}, \mathcal{I}, \mathcal{G} \rangle$, where \mathcal{O} is a set of objects, \mathcal{P} is a set of properties defined over one or more of these objects, \mathcal{A} is a set of lifted actions, \mathcal{T} is a state transition function, \mathcal{C} is a transition cost function, \mathcal{I} is the initial state of the environment, and \mathcal{G} is the desired or goal state of the environment.

T 1 1 0 1		1 1.	1 .		· •	1 •	1 1
Table 2.1	Taxonomy of	symbolic i	alanning	hy varving each	component in a i	alanning	problem
TUDIC 2.1.	ranomonity or	symbolic p	Jamming	by varying cach	component m a	Jamms	problem

Component varied	Classes of planning problems
state transitions (\mathcal{T})	deterministic, nondeterministic, probabilistic
observability $(\mathcal{I}, \mathcal{O}, \mathcal{P})$	full, partial, none
horizon (\mathcal{T})	finite, infinite
goal specification $(\mathcal{G}, \mathcal{C})$	satisficing, optimizing

Taxonomy of planning problems: Depending on the specific choices and constraints imposed on each component of a planning problem Π , we describe a taxonomy of symbolic planning. State transitions in a planning problem may be deterministic, nondeterministic, or probabilistic. Additionally, the state of the environment and the agent may be fully observable, partially observable, or in a hypothetical extreme, unobservable. Based on the planning horizon, problems are classified as finite or infinite. Depending on the goal specification, planning problems may be specified as *satisficing* or *optimizing*. Satisficing planning problems necessitate the computation of a solution that satisfies a set of conditions without regards to the cost of executing the computed plan. However, optimizing planning problems require the computation of the least cost plan that achieves a goal state. Typically, optimizing planning problems are much harder than satisficing planning problems due to this added complexity of finding minimum-cost solutions. This taxonomy is summarized in Table 2.1.

Planning domain definition language (PDDL): A standard language used to encode symbolic planning tasks is the planning domain definition language (PDDL) [17]. PDDL has been widely influential in providing a unified mechanism to specify planning problems, and has resulted in dramatic progress in the field of classical (symbolic) planning since the mid-1990s [18]. A PDDL planning problem is a special case of the symbolic planning problem defined above, where the set of properties \mathcal{P} is constrained to contain unary or binary predicates⁵. PDDL specifies a planning problem in two stages. First, a *domain file* describes the set of object types, (unary and binary) predicates, and actions. Each action has a *precondition* and an *effect*. A precondition is a binary condition (usually conjunction or disjunction of literals) that must be satisfied before an action may be executed. An effect is a consequence of the action; it describes the eventual value of all predicates that are impacted

⁵We note that recent versions of PDDL support properties with higher arity and initial support for continuous parameters. In this dissertation, we restrict our discussion to PDDL1.2 and earlier

by an action. Many planning systems partition the effect into an *add list* and a *delete list*, where the add list contains positive literals and the delete list contains negative literals. PDDL preconditions can include universal quantification, and effects can include nested conditions. This information included in the domain file specifies the problem domain. The actual problem instance is encoded in a separate file, called the *problem file*. The problem file links to the domain file to retrieve the list of object types, properties, and actions. This file then specifies the actual object instances available to the planner, and the initial and goal conditions. The burden of enumerating all possible transitions and costs is thus shifted onto the PDDL parser and compiler, which allows for the specification of a wide variety of planning problems. PDDL has since been extended to handle probabilistic effects [19], temporal domains [20, 21], and blackbox samplers [22].

Another prominent specification language for planning problems is STRIPS (Stanford research institute problem solver) [23], which is a strict subset of PDDL. STRIPS assumes a *closed-world*, which implies that all predicates that are not specified in the problem are assumed to be *false*. PDDL, on the other hand, assumes unspecified predicates as *unknown*. This allows PDDL to specify a richer set of planning problems.

2.3.2. Classical planning techniques

A large body of work on symbolic planning focuses on designing efficient variants of graph-based search. We refer to this class of approaches—dating from the mid-1980 through the late 2000s—as *classical* planning techniques. These techniques are primarily rule-based or algorithmic, and do not leverage a specific set of prior experiences to plan more efficiently. This is in contrast to learning-based planners, which we discuss subsequently.

To solve the large, complex planning problems that arise in everyday tasks (such as cleaning or organizing a home, cooking a recipe), pure graph search strategies are computationally infeasible. This is because, to guarantee *completeness*⁶, search techniques will often need to visit millions of graph nodes until a solution is found. Furthermore, it may not be possible to fully expand out a state-transition graph to begin with, as planning problems involve state spaces with large branching factors and/or stochastic transitions. This has resulted in a bifurcation of classical planning techniques based on the operational constraints over the goal specification.

Satisficing planners are the dominant class of approaches that have found widespread use in time-constrained robotic applications. These planners attempt to find a feasible solution, as opposed to *optimal* planners which attempt expensive searches to obtain the best possible solution (if one exists). Satisficing planners usually rely on several kinds of heuristics to discard

 $^{^{6}}$ A complete planner is one that is guaranteed to return a solution if one exists for the problem (or report failure otherwise).

large regions of the state-space [24, 25, 26], while optimal planners rely on propositional logic and constraint satisfaction solvers to generate optimal solutions [27, 28, 29, 30].

Heuristic-based planners: A popular class of satisficing planners leverage a declarative specification of the planning problem to devise *admissible* heuristics – cost-to-goal estimators that never overestimate. These admissible heuristics are employed to constrain the search algorithm to promising paths in the planning state graph. Heuristics may be computed via abstracted planning [31], detecting landmarks [32], identifying critical paths in the planning search graph [33], or ignoring delete lists [34, 25, 26]. Some of the most prominent classical planners adopt the latter strategy of ignoring delete lists because of its simplicity. Important variants of this class of planners include planning as heuristic search (HSP) [34], fast-forward planning (FF-plan) [25], fast-downward planning [26] and variants. To date, FF-plan and FD-plan remain the most popular planners for use with robotic task planning [35, 36].

There are several design choices involved in a heuristic planner. The first—and admittedly most important—one is the choice of heuristic. Apart from this, the graph search strategy, node selection mechanism, exploration-exploitation tradeoff, and the choice of data structures used to store frontier nodes and delete lists; are all important factors that influence planning performance. A few popular search strategies include the A* planner and variants, best-first search strategies, and hill-climbing methods.

HSP: HSP [34] was one of the first planners that rendered state-space search practical for large planning problems. HSP ignores delete lists and introduces a heuristic that approximates the cost of an optimal solution in the relaxed-planning graph thus obtained. While the heuristic function is effective for planning, it is computationally expensive, and HSP as a strategy does not guarantee completeness.

Fast forward planning: The FF-plan system [25] is a widely adopted system, proposing several key improvements over HSP. The heuristic used in FF-plan leverages GraphPlan [37] which is capable of polynomial time planning over reduced planning graphs (where delete lists have been ignored). An enforced hill-climbing strategy is used for search, which guarantees completeness provided the graph does not contain any dead ends (irreversible effects that do not allow for backtracking). While FF-plan is much faster, its heuristic is often inferior to HSP, resulting in longer plans with increased costs.

Fast downward planning: The FD-plan system [26] extends FF-plan by combining three independent search strategies and deriving heuristics leveraging graph hierarchies. The three search strategies are complementary, and use multiple heuristics. Several variants of FD-plan have been proposed over the years, and this currently remains the de facto classical planner for satisficing planning scenarios. While FD-plan takes significantly longer time to compute heuristics, solutions produced by FD-plan tend to be much shorter than FF-plan.

MCTS and regression planning: Other prominent classical symbolic planners include Monte Carlo tree search (MCTS) [38] and regression planning [39]. MCTS leverages a large number of random sample paths (rollouts) over the state space to identify promising actions, while regression planning searches *backwards* for a path from the goal state to the start state.

2.3.3. Learning-based planning

While satisficing symbolic planners have achieved remarkable results over real-world planning domains, these techniques primarily rely on heuristics and search strategies crafted by domain experts for efficient planning. Enabling planners to automatically discover task-specific heuristics and/or search strategies should therefore improve planning performance. This is the primary motivation that underlies several learning-based planning techniques.

Approaches to learning-based symbolic planning may be broadly classified into one of the following paradigms. The first category of approaches [36, 40] learns projective abstractions that reduce the original planning problem to a (typically) smaller, easier to solve subproblem. These subproblems are then solved by leveraging classical symbolic planners at a fraction of the computational costs associated with solving the full problem. The second category of approaches [41, 42, 43, 44, 45] instead deploys a function approximator (such as a neural network) as a differentiable planning module, optimized to find the best possible plans for the original planning problem.

Learning projective abstractions for symbolic planning: This class of approaches stems from the idea that in several planning problems of interest [46], there exist a large fraction of objects that are extraneous to the planning problem; i.e., such objects do not form a part of the solution and may be safely ignored. These notions of contextual irrelevance and object importance are leveraged in two recent approaches – CAMPS [40] and PLOI [36]. CAMPS leverages the notion of context-specific independence [47] to project out all contextually irrelevant variables to produce smaller and easier subproblems. PLOI [36]. on the other hand, trains a neural network that estimates an *importance score* for each object in the planning problem. Objects with importance scores above a threshold are used to define a reduced planning problem, which is solved using classical symbolic planners such as FD-plan [26]. To account for erroneous importance score that lead to incorrectly reduced planning problems, PLOI incorporates an iterative replanning strategy – if a reduced planning problem is deemed unsolvable, the importance score threshold is lowered using a multiplicative decay factor, which results in more objects being retained in the reduced problem. In the limit, the threshold approaches 0 and all objects in the original planning problem are retained the reduction, ensuring completeness. PLOI results in state-of-the-art performance over several challenging symbolic domains. However, in large problem instances, PLOI also tends to incur substantial replanning costs, which we address in our work. We note that projective abstractions have also been used in other planning approaches with similar performance gains [48, 49, 50, 51, 52].

Learning neural symbolic planning strategies: An alternative class of learningbased symbolic planners leverage deep neural networks to predict state transitions over a planning graph or to learn heuristics and value functions over the search space. Such techniques often draw inspiration from successful classical planning counterparts such as MCTS and regression planning and swap out key handcrafted components in favor of flexible learnable modules. MCTSNets [45] replaces the various components of MCTS using neural networks that evaluate the quality of a state, simulate a rollout from that state, and preform value propagation respectively. All of these modules are supervised by leveraging a database of expert demonstrations that are used to generate a reward signal that identifies and promotes promising actions. This approach has been applied to long-horizon planning problems, albeit outside the areas of robotics. Regression planning networks (RPNs) [44], on the other hand, model their approach based on classical regression planning. RPNs begin from the goal state and propose a set of subgoals conditioned on the current observation. This conditioning allows for learning-based approaches to lean on prior experience and speeds up search. RPNs achieve remarkably complex tasks in small-scale planning problems, but their applicability to large problem instances remains to be seen.

2.3.4. Low-level planning

Motion planning: Unlike symbolic planning, which focuses on determining the sequence of symbolic/logical actions to achieve a goal, motion planning must take into account the kinematic constraints imposed by the robot's mechanical design and the geometric structure of the environment. The earliest motion planning techniques were geometric techniques that computed solutions by parameterizing the environment using a set of analytical primitives. However, the current best approaches to motion planning are sampling-based [53, 54] which rely on efficient sampling strategies over the robot's configuration space to plan faster. An extensive survey of motion planning techniques can be found in Lavalle [55].

Task and motion planning: A noteworthy line of research that is not directly related to this dissertaion is that of (simultaneous) task and motion planning (TAMP). This class of approaches jointly solve the symbolic and motion planning problems, enabling synergies and communication between the typically disjoint planning modules [56, 57].

2.4. Robot control

The third SPA subsystem is the *control* subsystem, whose primary responsibility is to ensure that the motion plans computed by the planning subsystem are accurately executed, despite the presence of actuation noise and other disturbances (unmodelled effects) in the environment. The development of robot control strategies shares a strong parallel in state estimation techniques (as estimation and control are often employed in an interleaved pipelines even outside of the SPA framework). Based on the information that a controller assumes about the system to be controlled, control techniques are bifurcated into *model-based* and *model-free* approaches.

Model-based control: Model-based approaches involve building a model (or assume one is provided) describing the state-transition dynamics of the system. We define $f_{\theta} : \mathcal{X} \times \mathcal{U} \mapsto \mathcal{X}$ as the transition model, where \mathcal{X} is the state space (i.e., the set of possible states), \mathcal{U} is the action space (i.e., the set of possible actions), and θ are the model parameters. This model is subsequently used in a control strategy that generates a sequence of actions that track a motion plan computed by the motion planning module. A popular class of model-based control is the linear quadratic regulator (LQR) [58], based on the Kalman filter for linear Gaussian models. Another class of popular approaches, which we discuss below in greater detail, is model-predictive control (MPC). Model-based approaches tend to struggle when system dynamics become extremely hard to model reliably. This has resulted in a vast body of literature on robust controller design and optimal control that deals with designing controllers for environments with hard-to-model disturbance factors [59].

Model-free control: Model-free approaches, on the other hand, directly compute a sequence of actions without leveraging an explicit model of state transition dynamics. Such approaches are conceptually simpler and can be optimized end-to-end on a given task, bypassing the model-building stage entirely. Modern model-free reinforcement learning approaches to control have enabled the learning of control policies from very high-dimensional observations such as raw image sequences [60, 61]. However, these approaches suffer from issues of poor sample complexity and generalization. Furthermore, optimality guarantees are hard to obtain for model-free approaches. Conceptually simple model-free controllers deployed in modern robots include the proportional-integral-derivative (PID) (and variants) [62].

2.4.1. Model-predictive control (MPC)

Model-predictive control (MPC)—also called receding horizon control (RHC)—is an extremely popular class of model-based control techniques. In MPC, a predictive model of state transitions is employed in an online optimization problem in a receding-horizon fashion. Formally, the optimization problem solved by an MPC technique is specified as follows [63].

$$\underset{\mathbf{x}_{1:T}\in\mathcal{X}, \mathbf{u}_{1:T}}{\operatorname{argmin}} \quad \sum_{t=1}^{T} C_t(\mathbf{x}_t, \mathbf{u}_t) \text{ subject to } \mathbf{x}_{t+1} = f_{\theta}(\mathbf{x}_t, \mathbf{u}_t), \mathbf{x}_1 = \mathbf{x}_{\text{init}}$$
(2.4.1)

In other words, MPC aims to find a sequence of states $\mathbf{x}_{1:T}$ and control actions $\mathbf{u}_{1:T}$ that minimize a task-specific objective (or cost) function $\sum_t C_t(\mathbf{x}_t, \mathbf{u}_t)$, while ensuring that the sequence of state transitions is feasible. In practice, MPC optimizes this cost over a time horizon T and executes the most promising control action. At the next time-step, instead of executing the next control action in the sequence, MPC solves the optimization using the most current state estimate; to achieve robustness to noisy dynamics models.

2.4.2. Differentiable physics for model-based control

Differentiable simulation is a new frontier that leverages *differentiable programming* to compute *gradients* of the simulator inputs and parameters w.r.t. the outputs. This enables efficient gradient-based optimization to estimate the parameters of a model-based controller by minimizing an objective function defined over the simulator outputs.

Differentiable simulation has shown impressive results in learning simulation of deformable objects [64, 65, 66], fluid dynamics [67], and simulating molecular dynamics [68]. However, existing approaches leveraging differentiability require expensive state-space supervision, which is infeasible for systems involving extreme deformation such as soft robots or cloth.

2.5. Computer vision

The primary sensing modality employed in this dissertation is vision. Much like humans rely on visual perception for several innate capabilities that enable them to plan and execute day-to-day tasks, the central goal of computer/machine vision is to enable machines to do the same, by inferring meaningful attributes of the scene given a set of projections of that scene captured by imaging devices. Computer vision includes a broad set of capabilities, generally pertaining to *recognizing* objects and other meaningful entities in a scene, *reconstructing* aspects of data that are lost in the imaging process (e.g., 3D structure), and *reorganizing* these scene entities and data into meaningful percepts that are passed on to downstream modules [69]. In this sub-section, we survey two key computer vision components that our work builds on, namely, structure-from-motion (SfM) and visual SLAM.

2.5.1. Structure-from-Motion

Structure-from-Motion (SfM) is the task of recovering the underlying 3D structure of a scene by leveraging steriopsis induced by a moving camera. For a long period in the history of computer vision (until the past decade), 3D computer vision was synonymous with SfM. Approaches to SfM may be classified on the basis of the representation of the scene or depending on the nature of the algorithm.

Batch vs online SfM: Early approaches to SfM include *offiline/batch* mode techniqes, which compute scene structure after all input images have been observed. This allows such approaches to optimally choose the sequence of images that would produce the best result, and also leverage simultaneous scene observations from all available viewpoints. *Online/incremental* SfM, on the other hand, aims to provide an up-to-date structure and motion estimate as each image arrives. In this paradigm, the SfM technique has little control

over its stability (as it has no choice but to process the subsequent image), however this enables several interactive applications (such as robotic manipulation and navigation) which require a live scene model.

Feature-based (sparse) vs Dense SfM: Depending on the nature of the scene representation, SfM techniques may either be *sparse* or *dense*. In sparse SfM, 3D estimates are recovered only for subset of distinct pixels across all images. On the other hand, in dense SfM, 3D estimates of every pixel in every image are recovered. The choice of scene representation crucially relies on the downstream task; for tasks involving robot navigation on flat ground, sparse maps have been demonstrated to be sufficient; for tasks involving object manipulation or scene digitization, dense representations are crucial.

Subsystems of an SfM pipeline: Most modern SfM systems comprise two components – a *frontend* and a *backend*. The frontend component is responsible for extracting initial estimates of scene structure and camera motion (e.g., by extracting and comparing sparse features across images). The backend component is responsible for refining these initial estimates by leveraging global consistency constraints. Modern SfM backends cast the problem of recovering structure and motion estimates into a sparse graph optimization framework, akin to graph SLAM systems. This class of problems, commonly referred to as *bundle adjustment* (BA), has received a great deal of attention in the 1990s and 2000s, and has resulted in several stable implementations, to the extent that these are now treated as plug-and-play components in certain computer vision applications.

2.5.2. Visual SLAM

Visual SLAM is the variant of SLAM that relies primarily on visual observations. Visual SLAM is a subset of incremental SfM and can produce either sparse (feature-based) or dense scene representations (maps). A subtle distinguishing factor between visual SLAM and incremental SfM is that visual SLAM typically assumes all images to be captured by the same imaging device, while in the case of incremental SfM, each incoming image may be captured by a different camera. In this subsection, we briefly review prominent approaches to visual SLAM spanning the last 4 decades⁷.

Taxonomy of visual SLAM systems based on map representations: A central question surrounding the design of visual SLAM systems is one of map representation. Depending on the choice of map representation, visual SLAM systems are broadly categorized as *feature-based* and *dense* SLAM approaches. Feature-based SLAM systems process incoming image streams to identify distinctive points (features) across images, and represent each of these features as 3D entities. These features are chosen in a manner that makes them most amenable to data association (tracking) and reconstruction (mapping). Common

⁷From this point on, we use the term SLAM to mean *visual* SLAM.

choices for map features include points, lines, planes, fiducial markers, or a combination of these. Feature-based SLAM is an extremely popular paradigm owing to its computational efficiency and deployability on resource-constrained settings. However, in environments where features are scarce (e.g., textureless environments, strong camera motion), feature-based approaches are rendered inapplicable. An alternative to feature-based SLAM approaches is dense SLAM, wherein every pixel in every image is stored in the map. While this paradigm is computationally more intensive, dense SLAM produces map representations that are human interpretable and may be consumed by graphics applications. Popular choices for representing dense 3D reconstructions include pointclouds, voxel grids, signed distance functions, surfels, and polygonal meshes.

Visual odometry (VO): The beginnings of classical visual SLAM can be traced back to the seminal work on visual odometry (VO) by Nister *et al.* [70]. The term visual odometry refers to the idea of estimating camera egomotion solely from image sequences. Unlike visual SLAM, the task of VO does not involve maintaining an explicit map of the environment (however, modern VO systems tend to maintain an internal map to aid egomotion estimation). The beginnings of VO can be traced to early work by Moravec in the 1980s [71] on vision-based obstacle avoidance and navigation for mobile robots.

Filtering-based SLAM: One of the first successful demonstrations of reconstructing 3D maps from image sequences was the DROID system by Harris and Pike [72]. In this work, a set of distinct feature points were extracted from each incoming image and fed into an EKF for state estimation. A key limitation of this approach, however, was that each estimation problem was solved independently without regards to egomotion estimation. MonoSLAM [73] was the first to demonstrate stable real-time mapping from a mobile monocular camera. While MonoSLAM had its roots in Davison's dissertation in 1999 [74], the key insight to real-time performance was to construct a joint estimation problem involving the egomotion of the camera, as well as feature points in the environment. However, the tight coupling between egomotion estimation (tracking) and 3D feature reconstruction (mapping) renders the system very sensitive to errors induced by incorrect data association or motion blur induced by freeform camera motion.

Decoupled tracking and mapping: A breakthrough in model-based visual SLAM was the PTAM (parallel tracking and mapping) approach by Klein and Murray [75]. The key idea in PTAM was to decouple the tracking and mapping subproblems in visual SLAM, allowing the processes to run in parallel on independent threads. This allows the tracking—a fast process—to run in the foreground at frame rate (i.e., in real-time), while the mapping process can run in the background and maintain a slightly outdated map. This paradigm of decoupling the tracking and mapping processes is retained by several state-of-the-art VO and SLAM systems over the years.

Visual place recognition: A key component of a long-term SLAM system is *loop* closure – the ability to recognize redundant scene attributes across time and leverage the associated observations to reduce estimation errors. In the context of visual SLAM, this capability is achieved by visual place recognition (VPR) modules. A popular approach to appearance-based mapping was FABMAP [76], which learned a bag-of-words model over a vocabulary of distinctive point features extracted from an image sequence. This formed the basis for several classical place recognition systems that are employed in state-of-the-art SLAM pipelines. Recently, learning based approaches—notably NetVLAD [77] and variants—have greatly improved the robustness of place recognition techniques to appearance changes across time. For comprehensive reviews on visual place recognition and discussions on emerging directions respectively, we refer to the surveys by Lowry et al. [78] and Garg et al. [79].

Modern feature-based SLAM systems: State-of-the-art feature-based SLAM systems such as ORB-SLAM [80, 81] leverage advances on all of the aforementioned fronts and combine them with smart keyframe selection strategies and robust backend optimizers to provide performant real-time SLAM capabilities.

Dense SLAM: The field of dense visual odometry began with work by Steinbrucker *et al.* [82], who formulated 3D egomotion estimation as an image alignment problem. Instead of finding distinctive features within an image, egomotion is estimated by computing a perspective warp that maps each pixel in a source image to a corresponding pixel in a target image by directly minimizing the difference in their per-pixel intensities. This was later developed into a complete RGB-D odometry system by Kerl *et al.* [83] and extended to perform SLAM. Another approach, DTAM (dense tracking and mapping) [84], enabled real-time dense SLAM from images captured by monocular cameras by formulating a photometric error over a cost volume.

RGB-D SLAM: The advent of real-time depth capture technologies such as Microsoft Kinect and access to consumer-grade GPU compute dramatically impacted the landscape of dense SLAM research. Novel techniques to fuse depth observations from multiple views in real-time on a GPU were proposed. KinectFusion [85] leveraged volumetric signed distance functions to fuse incoming observations, followed by an alignment scheme to compute egomotion estimates. The entire map was stored and manipulated in GPU memory, which eliminated the need for explicit place recognition and pose-graph optimization; but limited the resolution and/or extent of the maps that could be built. Kintinuous [86] extended KinectFusion [85] by the combination of a novel moving-volume GPU buffer that followed the camera, and a meshing scheme that converted the out-of-buffer map elements into a triangle mesh amenable to pose-graph optimization. Point-based fusion [87] proposed a surfel representation that eliminated the need for voxel-grid maps. This was extended to incorporate graph optimization with a novel map deformation constraint in ElasticFusion [88]. Semi-dense and semi-direct SLAM: Some SLAM systems attempt to leverage the strengths of both feature-based and dense SLAM systems. Approaches such as LSD-SLAM [89] and DSO [90] design photometric bundle adjustment estimation routines for monocular videos. SVO [91] leverages a probabilistic mapping scheme to enable real-time and robust operation.

Deep learning for SLAM: Deep convolutional neural networks have demonstrated tremendous potential for image-based recognition tasks. Traditionally, maps obtained from a SLAM system only capture the geometric structure of a scene. Integrating deep neural networks into SLAM pipelines enables maps to store richer semantic information, such as the objects contained within a scene, their functional attributes, etc. SLAM ++ [92] was an early dense SLAM system that leveraged KinectFusion to build object-based maps of an environment. McCormac *et al.* [93] extend ElasticFusion [88] by integrating it with an image segmentation module to obtain consistent 3D semantic labels. Multiple other SLAM systems have integrated deep neural networks for object detection into SLAM pipelines [94, 95, 96]. Another prominent application of deep learning for SLAM has been in the areas of depth estimation from a single image. Approaches such as SfMLearner [97] have demonstrated the role of projective geometric constraints in recovering dense depth maps from a pair of images. Depth estimates thus learned have directly been plugged into SLAM pipelines [98]. Approaches such as CodeSLAM [99, 100] have attempted a deeper integration between trainable neural network components and traditional dense SLAM by optimizing compact codes in a graph optimization framework.

2.6. Nonlinear least squares optimization

As seen in the preceding sections, several tasks of interest in the robotics and computer vision communities are formulated as nonlinear least squares problems. A nonlinear least squares problem optimizing for the parameters \mathbf{x} of a differentiable function f(.) given observations \mathbf{y} has the following sum-of-squared-residuals form.

$$\min_{\mathbf{x}} \mathcal{C}(\mathbf{x}) = \sum_{i} \mathbf{r}_{i}^{T}(\mathbf{x}) \mathbf{r}_{i}(\mathbf{x}) = \sum_{i} \|f_{i}(\mathbf{x}) - \mathbf{y}_{i}\|_{2}^{2}$$
(2.6.1)

Each residual $\mathbf{r}_i(\mathbf{x})$ is a nonlinear function of the parameters \mathbf{x} . Example nonlinear leastsquares problems of note include pointcloud alignment, mesh deformation, image warping, trajectory optimization, and structure-from-motion. While these problems can be solved using first-order optimization techniques like gradient descent, specialized solvers leveraging approximate curvature information have been developed over the last several decades. Of these, the Gauss-Newton or Levenberg-Marquardt are the most widely applied, with the former being preferred for convergence speed and the latter being preferred in applications with larger noise ranges or poor initial guesses.

2.6.1. Gradient descent

A conceptually simple approach to minimize differentiable objective functions is to gradually update the parameter vectors being optimized along the negative direction of the gradient. Given the objective function $C(\mathbf{x})$ and an initial guess \mathbf{x}_0 , gradient descent applies the following update rule

$$\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t - \eta \nabla_{\mathbf{x}} \mathcal{C}(\mathbf{x}) = \mathbf{x}_0 - \eta \frac{\partial \mathcal{C}(\mathbf{x})}{\partial \mathbf{x}} \Big|_{\mathbf{x} = \mathbf{x}_t}$$
(2.6.2)

Under this scheme, the direction of each update is determined by the gradient $\nabla_{\mathbf{x}} \mathcal{C}(\mathbf{x})$, and the magnitude of the update is determined by the step size parameter η . Gradient-based optimization schemes are the dominant tools used for loss minimization in several disciplines across science and engineering, primarily due to their simplicity.

2.6.2. The Gauss-Newton (GN) method

The Gauss-Newton method is an iterative optimization method involving two steps (lineraization, followed by a Newton update). First, the cost function $C(\mathbf{x})$ linearized by using a first order Taylor series expansion around the current solution guess (see Eq. below).

$$\mathbf{r}_{i}(\mathbf{x})|_{\mathbf{x}=\mathbf{x}_{0}} = \mathbf{r}_{i}(\mathbf{x}_{0}) + \mathbf{J}_{i}(\mathbf{x})|_{\mathbf{x}=\mathbf{x}_{0}}\delta\mathbf{x}$$
$$= \mathbf{r}_{i}(\mathbf{x}_{0}) + \frac{\partial r_{i}(\mathbf{x})}{\partial\mathbf{x}}\Big|_{\mathbf{x}=\mathbf{x}_{0}}\delta\mathbf{x}$$
(2.6.3)

This results in a linear least squares system (locally linear at the linearization point \mathbf{x}_0). Each block in the least squares system thus obtained has normal equations of the form

$$\mathbf{J}_i^T \mathbf{J}_i \delta \mathbf{x} = \mathbf{J}_i^T \mathbf{r}_i \tag{2.6.4}$$

The resulting linear least squares system is solved to compute the updated parameter vector, which now becomes the *linearization* point for the next iteration. This process is repeated until convergence, or a predefined tolerance criterion is achieved. Note how, different from gradient descent updates, there is no requirement of a *step size* parameter η . The magnitude of the update is determined by $(\mathbf{J}_i^T \mathbf{J}_i)^{-1}$; an approximation of the local curvature at the linearization point. This allows Gauss-Newton optimization procedures to converge orders of magnitude faster than gradient descent when suitably initialized.

2.6.3. The Levenberg-Marquardt (LM) method

The above (Gauss-Newton) approximation has two important ramifications. First, there are no checks in place to ensure that subsequent iterates do not diverge from the initial solution. In practice, divergence is observed when the initial guess is poor (i.e., outside the *basin of convergence*). Second, the algorithm is numerically unstable when the approximate

Hessian $(\mathbf{J}^T \mathbf{J})$ is near-singular. The Levenberg-Marquardt technique [101, 102] addresses both these shortcomings. It is a *trust-region method*, in that it specifies a region around the current linearization point (the *trust-radius*) where the Taylor-series expansion (and thereby the Hessian-approximation) holds.

For a given linearization point \mathbf{x}_i , one LM iteration proceeds as follows:

- (1) Compute the approximate Hessian as $\mathbf{J}_i^T \mathbf{J}_i + \lambda \mathbf{I}$, where λ is a damping coefficient (usually initialized to a tiny constant), for numerical conditioning (λ indicates the trust-radius).
- (2) Using this approximate Hessian, perform a Newton step to compute a *lookahead* subsequent iterate $\mathbf{\hat{x}}_{i+1}$.
- (3) Evaluate the cost function at this lookahead iterate $\hat{c}_{i+1} = \mathcal{C}(\hat{\mathbf{x}}_{i+1})$, and compare it with the current cost $c_i = \mathcal{C}(\mathbf{x}_i)$.
- (4) If the lookahead cost \hat{c}_{i+1} increases (compared to c_i), do not update the parameter vector (\mathbf{x}_i) . Increase the damping factor using a suitable damping strategy [103].
- (5) Otherwise, i.e., if the lookahead cost c_i reduces, update the parameter vector \mathbf{x}_i to $\mathbf{\hat{x}}_{i+1}$ and decrease the damping factor.

This variant of the approximate Hessian lends LM the representational power to switch between GN-like $(\lambda \to 0)$ behavior and GD-like behavior $(\lambda \to \infty)$. The most commonly used damping strategy is the multiplicative damping strategy, which damps by using the rule $\lambda \leftarrow \lambda \times 2$ and undamps by using the rule $\lambda \leftarrow \frac{\lambda}{2}$. Modern approaches to nonlinear least-squares leverage LM solvers due to their flexibility and non-divergence guarantees.

2.7. Machine learning background

Modern machine learning (ML) techniques—primarily deep learning (DL) methods—have unlocked a new level of performance in several domains such as image, text, and speech recognition. The term *deep learning* commonly refers to the use of deep neural networks (DNNs, or simply, NNs) (i.e., neural networks comprising several stacked layers) and gradient based optimization (backpropagation) for learning representations. DNNs have proved to be a successful paradigm for learning representations, particularly in the supervised setting (where labeled data exists). While neural networks offer an elegant framework to process large volumes of such data, the most successful ML techniques for processing and manipulating data mandate the availability of large (and ideally, carefully-labelled) training datasets. In the absence of large amounts of data, representation learning is typically enabled by exploiting inductive biases specific to the class of problems under consideration.

In this section, we review some essential deep learning concepts used in this dissertation. For a thorough reference on the subject, we refer the reader to the deep learning textbook by Goodfellow *et al.* [104].

2.7.1. Machine learning problem

In modern machine learning problems, the primary task is to leverage observed data (the *training* set) to specify a function that estimates (predicts) values of interest over unobserved data (the *test* set). Formally, assume a training dataset $\mathcal{D} \triangleq \{(x_i, y_i)\}_{i=1...|\mathcal{D}|}$ with $x_i \in \mathcal{X}$ where \mathcal{X} is the input space and $y_i \in \mathcal{Y}$ where \mathcal{Y} is the output space. A machine learning model is a function $f_{\theta} : \mathcal{X} \mapsto \mathcal{Y}$ from the input to the output space. The quality (accuracy) of this function is computed by measuring the discrepancy \mathcal{L} between the function outputs $f(x_i)$ and the corresponding target values y_i . This discrepancy measure is often referred to as *empirical risk*, and the function \mathcal{L} is referred to as a **loss function**. A machine learning problem involves estimating the parameters θ of the model f_{θ} , typically by minimizing the empirical risk.

$$R_{emp}(f_{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f_{\theta}(x_i), y_i)$$
(2.7.1)

This above setting is commonly referred to as **supervised learning**. Alternative paradigms where labelled data is not assumed available include *semi-supervised* or *un-supervised* learning. For the case where labeled data is not directly available but the quality of a model prediction may be evaluated, learning paradigms such as *reinforcement learning* are applicable. The techniques developed in this dissertation stem from the above empirical risk minimization setup, so we limit our discussion to the supervised learning setting.

2.7.2. Deep learning: Neural networks

Deep learning commonly refers to the use of deep neural networks (i.e., neural networks comprising several stacked layers) and gradient based optimization (backpropagation) for learning representations. This dissertation only employs a set of deep neural networks referred to as *feedforward* neural networks. We summarize common classes of feedforward neural networks below.

Feedforward neural networks: This is the most commonly used class of modern neural networks. A feedforward neural network f_{θ} with parameters θ is a composition of $N \geq 1$ arbitrary (usually, differentiable) functions f_1, f_2, \dots, f_N that map an input $\mathbf{x} \in \text{dom}(f_1)$ to an output $\mathbf{y} \in \text{range}(f_N)$ via the function composition $f_{\theta} = f_N \circ f_{N-1} \circ \cdots \circ f_2 \circ f_1$.

Depending on the type of functions $f_i, i \in \{1, \dots, N\}$, there exist several types of feedforward networks. The most important types of feedforward networks are:

- Multilayer perceptrons (MLPs), where each function f_i is an affine transform of the form $\mathbf{y} = f_i(\mathbf{x}; \theta) = \mathbf{W}_i \mathbf{x} + \mathbf{b}_i$
- Convolutional neural networks (CNNs), where each function f_i is a discrete transposed convolution with a kernel K, i.e., $f_i(\mathbf{x}) = \mathbf{x} * K$, where * is the transposed convolution operator

• Graph neural networks (GNNs), where the inputs \mathbf{x} (and optionally, outputs \mathbf{y}) are graph structured and each function f_i is usually a permutation-invariant computation defined over the graph (i.e., permuting the indices of nodes and edges does not affect the output of f_i).

2.7.3. Backpropagation

The loss function in Eq. 2.7.1 is commonly optimized by gradient descent, to gradually update the parameters θ of the model f_{θ} (i.e., the neural network). The empirical risk function R_{emp} is minimized by using the following gradient-descent update rule:

$$\theta_{i+1} \leftarrow \theta_i - \eta \nabla_\theta R_{emp}(f_\theta) \tag{2.7.2}$$

Backpropagation: In feedforward neural networks, the set of learnable parameters θ are distributed across several hidden layers. To enable updates to all of these parameters, most of which do not explicitly appear in the empirical risk computation, the widely adopted solution is *backpropagation*. At its core, backpropagation is simply the application of the chain rule of multivariate calculus to gradient-based optimization. Assume that the parameter set θ is a union of disjoint mutually exclusive parameter sets $\theta_1, \theta_2, \dots, \theta_N$ such that θ_i is the set of parameters of the function f_i , and the feedforward neural network specified by f_{θ} is $f_{\theta_N} \circ f_{\theta_{N-1}} \circ \cdots \circ f_{\theta_2} \circ f_{\theta_1}$. Without loss of generality, we assume the loss function $R_{emp}(f_{\theta})$ is subdifferentiable over the parameter space θ . This accommodates several classes of operations that are typical of empirical risk computation, such as pooling operations, sorting, and so on. Backpropagation first computes gradient updates for the outermost parameter set θ_N , as $\theta_N \leftarrow \theta_N - \eta \frac{\partial R_{emp}(f_{\theta})}{\partial \theta_N}$. Subsequently, the inner layers of parameters are gradually expanded, one layer at a time, until the chain rule traces back to the parameter set θ_1 . We now specify gradient descent update rules for inner (hidden) layer parameters θ_{N-1} through θ_1 . We use \mathbf{x}_i to denote the output of the feedforward network at the *i*th layer, i.e., $\mathbf{x}_i = f_{\theta_i}(f_{\theta_{i-1}}(\cdots f_{\theta_1}(\mathbf{x}))).$

$$R_{emp}(f_{\theta}(\mathbf{x})) = R_{emp}(f_{\theta_N}(f_{\theta_{N-1}}(\cdots f_{\theta_1}(\mathbf{x}))))$$

$$\theta_N \leftarrow \theta_N - \eta \frac{\partial R_{emp}(f_{\theta_N}(f_{\theta_{N-1}}(\cdots f_{\theta_1}(\mathbf{x}))))}{\partial \theta_N}$$

$$\theta_{N-1} \leftarrow \theta_{N-1} - \eta \frac{\partial R_{emp}(f_{\theta_N}(\mathbf{x}_{N-1}))}{\partial \mathbf{x}_{N-1}} \frac{\partial \mathbf{x}_{N-1}}{\partial \theta_{N-1}}$$
(2.7.3)

$$\theta_1 \leftarrow \theta_1 - \eta \frac{\partial R_{emp}(f_{\theta_i}(f_{\theta_{i-1}}(\cdots f_{\theta_1}(\mathbf{x}))))}{\partial f_{\theta_1}(\mathbf{x})} \frac{\partial f_{\theta_1}(\mathbf{x})}{\partial \theta_1}$$

In practice, gradient computation for several common operations can be automated, enabling easy implementation while also reducing scope for error. We discuss common approaches to algorithmic differentiation in the next section.

2.8. Differentiable programming

The ability to compute derivatives of functions is an essential tool in several disciplines of science and engineering. In the context of this dissertation, several problems in robotics, computer vision, graphics, and machine learning are cast into the framework of gradient-based optimization. However, computing gradients by hand is both cumbersome and is prone to human error-factors. Differentiable programming (also sometimes referred to as algorithmic or automatic differentiation) is a set of tools that enables automated computation of the derivates of a function (a program) by repeated application of the chain rule of calculus.

2.8.1. Computational graphs and explicit autodifferentiation

We review the idea of a *computational graph*, which is central to modern differentiable programming, and by extension, machine learning. Formally, a computational graph is a directed acyclic graph (DAG) where each node corresponds to an *elementary computation*, and (directed) edges indicate program control flow. Elementary computations include arithmetic, trigonometric, and other analytical differentiable functions, as well as other subdifferentiable functions such as pooling, ranking, and more. In a differentiable programming framework, these elementary computations have manually specified derivatives. A wide range of higher-order functions, including large neural networks with branches and conditionals, can be constructed by composing such elementary computations using operators for which manual derivative specifications exist. Popular modern frameworks for large-scale differentiable programming include PyTorch [105], TensorFlow [106], and JAX [107].

An example of the function 3(xy + z) is shown in Fig. 4.2. By definition, computational graphs are hierarchical, and composable. Any function that can be represented by a computational graph is amenable to algorithmic differentiation by application of the chain rule.

The program or function whose derivatives are being automatically computed is referred to as the forward or *primal* program. The intermediate computations are called *primals*.

Forward mode autodifferentiation: In the forward mode (also referred to as the *tangent* mode), *primals* and their derivatives are all computed in a single execution of the primal program. This scheme steps through all branches, unrolls control loops, inlines all functions, and generates additional variables to hold derivatives (referred to as *tangents*) with respect to each leaf (input) node. The complexity of this program scales linearly with increasing dimensionality of the input.

Reverse mode autodifferentiation: In the reverse mode (also referred to as the *adjoint* mode), primals and their derivatives are computed in two program executions, each in a different direction. The first execution runs the primal program, while logging all intermediate variables and the sequence of operations. Once this halts, the second execution begins in the reverse direction, tracing the computational graph backwards and computing the chain rule *backwards*. Reverse mode autodifferentiation is the mode employed when implementing backpropagation in a differentiable programming framework.

Forward vs reverse mode autodifferentiation: Both the forward and reverse modes have complementary strengths and weaknesses. The reverse mode does not add too much overhead on the actual program execution, whereas the forward mode execution slows down linearly with increasing input dimensionality. On the other hand, reverse mode autodifferentiation imposes an additional overhead – the need to cache intermediate values of primals, which is cumbersome as the output dimensionality increases. For deep neural networks, reverse mode is the preferred autodifferentiation mode as we derivative computation with respect to thousands, if not millions, of parameters.

Explicit audodifferentiation: Differentible programming frameworks enable the flexibile specification of arbitray computation flows (branches, conditionals, recursive functions); scenarios where an analytical expression for the derivative may not exist. This is enabled by flattening out branches by pruning other branches that are not taken, unrolling any loops, inlining functions, and reparameterizing recursion as iteration. We refer to this as *explicit* autodifferentiation and is the preferred technique for imperative programs.

2.8.2. The adjoint method and implicit autodifferentiation

There exist several classes of functions for which explicit autodifferentiation is cumbersome, or even infeasible. For example, consider a function that specifies a constrained optimization problem. An imperative program would merely unroll, inline, and flatten the program to build a computational graph. However, gradients through this graph do not necessarily satisfy the constraints imposed on the gradient due to the nature of the function. Another key issue with explicit autodifferentiation is that unrolling extremely long loops leads to vanishing or exploding gradients. We will now review key ideas in implicit autodifferentiation, primarily the discrete-time adjoint method.

Consider a function specified *implicitly* (i.e., as a constraint set) $\mathbf{g}(\mathbf{x}, \theta) = 0$, with θ being a function ⁸ of \mathbf{x} . Let $\theta^*(\mathbf{x})$ be a point in the constraint set, such that $\mathbf{g}(\mathbf{x}, \theta^*(\mathbf{x})) = 0$. By

⁸If θ were to be independent of **x**, an explicit differentiation scheme could have been applied instead.

implicit differentiation, one may compute the gradient of the function $\mathbf{g}()$ w.r.t. \mathbf{x} as follows:

$$\frac{\partial \mathbf{g}(\mathbf{x}, \theta^*(\mathbf{x}))}{\partial \mathbf{x}} = 0 \implies \frac{\partial \mathbf{g}(\mathbf{x}, \theta^*)}{\partial \mathbf{x}} + \frac{\partial \mathbf{g}(\mathbf{x}, \theta^*)}{\partial \theta^*} \frac{\partial \mathbf{g}(\mathbf{x}, \theta^*(x))}{\partial \mathbf{x}} = 0$$
$$\implies \frac{\partial \mathbf{g}(\mathbf{x}, \theta^*(x))}{\partial \mathbf{x}} = -\left(\frac{\partial \mathbf{g}(\mathbf{x}, \theta^*)}{\partial \theta^*}\right)^{-1} \frac{\partial \mathbf{g}(\mathbf{x}, \theta^*)}{\partial \mathbf{x}} \qquad (2.8.1)$$

This is a remarkable result, in that the resulting gradient depends solely on the input and the output, and does not require unrolling of the (typically iterative) function \mathbf{g} . For an in-depth review of implicit differentiation and its connections with modern deep learning, we refer to the excellent tutorial by Kolter *et al.* [108].

2.8.3. Autodifferentiation implementation: Tracing and program transformation

Tracing (Taping): The most common implementation of autodifferentiation used by frameworks such as Tensorflow [106] is *tracing* (sometimes referred to as *taping*). When the forward program is executed, tracing records intermediate values into a *tape* (a cache) linearly. In a subsequent replay of this trace, derivative programs are compiled.

Program transformation: While the partial derivatives above can be computed by graph-based automatic differentiation frameworks [105, 106, 107], there has been renewed interest in program transformation. In program—or source—transformation, adjoints of each elementary operation (*kernel*) are computed at runtime to generate derivative programs.

2.9. Computer graphics

This dissertation also builds atop recent advances in the computer graphics community along the lines of differentiable rendering and simulation.

2.9.1. Realistic image synthesis (Rendering)

In computer graphics, *rendering* refers to the process of converting a *scene description* into an 2D image of a scene. These scene descriptions typically include the form and placement of 3D objects in an environment, the material properties associated with these objects (e.g., wood, plastic, metal), the location and geometry of light sources in the environment, and the properties of a (virtual) camera used to capture a (virtual) image of the scene.

The complexity of this forward simulation problem can vary, depending on the assumptions made by the underlying rendering algorithm. For example, a simple lighting and shading model can lead to higher-performance simulations, albeit at the cost of accuracy – the resulting images may lack many important real-world shading cues, such as shadows or complex interreflection effects. More complicated lighting and shading models exist, including so-called "physically-based" models that aim to fully simulate the light transport and image formation process, typically relying on large-scale Monte Carlo simulations to resolve a complex high-dimensional integration problem.

2.9.2. Differentiable rendering

The simpler shading models readily admit differentiable implementations, where image gradients can be computed with respect to variations in the scene description parameters [109, 110, 111, 112]. Despite the limitations of the underlying image formation models here, these simple *differentiable renderers* have demonstrated the ability to solve many interesting inverse problems in computer vision. Moreover, the integration of differentiable renderers into larger end-to-end ML architectures is proving to be a very fruitful and powerful application of their capabilities: many applications – from larger-scale shape-from-shading computer vision problems to simulation-to-real policy training for robots in reinforcement learning – are all benefiting from the inclusion of differentiable models of shading. Furthermore, recent works in the computer graphics literature are exploring the design of efficient Monte Carlo-based differentiable physically-based renderers, allowing for even more accurate models of image formation to be integrated in end-to-end architectures [113, 114].

2.10. Context and open challenges

To successfully execute meaningful real-world tasks, each subsystem of the sense-plan-act computation model needs to solve extremely challenging sub-problems. These challenges are further exacerbated by the compounding errors through each subsystem and across each time step. This renders straightforward 'drop-in' solutions—that merely 'replace' these components with modern neural networks—infeasible. In this dissertation, we discuss alternative options to such *end-to-end* replacement, arguing in favour of a more modular and structured paradigm. We present first steps towards rethinking the SPA subsystem in the context of modern machine learning methods by tackling the following open challenges.

Deep learning for visual SLAM: While visual SLAM has progressed leaps and bounds in the years leading up to 2012⁹, deep learning is yet to significantly impact the visual SLAM research landscape. Some notable applications of deep learning to visual SLAM include feature extraction and matching modules like SuperPoint [115] and SuperGlue [116] respectively, which are increasingly being adopted as drop-in feature extractors and matchers. However, the remainder of the visual SLAM pipeline remains largely unchanged. In this dissertaion, we present an alternative approach to thinking about visual SLAM as a large, differentiable function comprising several subsystems. This view enables replacing relevant parts of visual SLAM pipelines with modern machine learning techniques, while retaining components of

 $^{^{9}}$ The year 2012 was when modern deep learning began to dominate the areas of visual recognition.

classical system that need not be re-learnt. This paves the way towards learning flexible, interpretable map representations tailored to downstream task performance signals.

Compact representations for symbolic planning: Existing symbolic planners fail to produce effective plans on long-horizon tasks, where some learning-based approaches to planning have achieved state-of-the-art results by pruning the state-space of search-based planners. However, such approaches require an additional learning phase on each problem domain for performant planning. We leverage the inherent hierarchy in a 3D scene to devise fast pruning procedures that enable classical planners to match, and outperform, state-of-the-art learning-based planners.

Gradient flow across large computation graphs: In the work presented in the subsequent sections, we extensively leverage ideas from the differentiable programming community to enable specifying our knowledge about the world in the form of *differentiable world programs*. We demonstrate that it is possible to build very large and structured differentiable programs while preserving the quality of gradients by reparameterizing non-differentiable operations or by leveraging program transformation.

State-less visuomotor control: State-of-the-art physically-based differentiable rendering systems have shown impressive results in extracting 3D scene geometry. However, they do not generalize to scenes with dynamic/interactive objects. We augment these differentiable rendering engines with differentiable physics engines that provide a principled model of world dynamics. This novel combination enables extremely challenging tasks, such as visuomotor control based on a single image, or system identification from a small set of frames in a video. The availability of an accurate and efficient differentiable simulator as a module in modern machine learning experimentation frameworks will also immediately facilitate its incorporation in larger end-to-end learning architectures that can leverage such capabilities.

Chapter 3

Prologue to Article 1

This article was published and presented at the IEEE International Conference on Robotics and Automation (ICRA) 2020. This was joint work with Ganesh Iyer and Liam Paull. At the time, Ganesh Iyer was a masters student at Carnegie Mellon University, USA.

Contribution statement: The core idea of gradSLAM was conceived by the author of this dissertation. This was refined over regular meetings with Liam Paull and Ganesh Iyer. The author implemented the bulk of the gradSLAM system. Ganesh Iyer implemented the differentiable PointFusion framework and carried out experiments on a real video sequence. The author and Liam Paull jointly prepared the manuscript.

After this manuscript was submitted to ICRA, Soroush Saryazdi led the refurbishing of the codebase into what eventually became the gradSLAM PyTorch library. The author carried out regular, extensive code reviews and testing to ensure correctness and reproducibility. Other major contributors to the gradSLAM library include Nikhil Varma Keetha, Sree Sai Harsha, and Abhishek Jain. This project also greatly benefitted by support from Prof. Derek Nowrouzezahrai and Prof. James Forbes.

Note: What follows is a slightly abridged version of the ICRA publication, with additional qualitative results that we had to leave out of the conference paper due to a hard space limit of six pages.

Chapter 4

Article 1: gradSLAM: Dense SLAM meets automatic differentiation

Abstract

Blending representation learning approaches with simultaneous localization and mapping (SLAM) systems is an open question, because of their highly modular and complex nature. Functionally, SLAM is an operation that transforms raw sensor inputs into a distribution over the state(s) of the robot and the environment. If this *transformation* (SLAM) were expressible as a differentiable function, we could leverage task-based error signals to learn representations that optimize task performance. However, several components of a typical dense SLAM system are non-differentiable. In this work, we propose ∇ SLAM (*grad*SLAM), a methodology for posing SLAM systems as *differentiable* computational graphs, which unifies gradient-based learning and SLAM. We propose differentiable trust-region optimizers, surface measurement and fusion schemes, and raycasting, without sacrificing accuracy. This amalgamation of dense SLAM with computational graphs enables us to backprop all the way from 3D maps to 2D pixels, opening up new possibilities in gradient-based learning for SLAM¹.

4.1. Introduction

Simultaneous localization and mapping (SLAM) has—for decades—been a central problem in robot perception and state estimation. A large portion of the SLAM literature has focused either directly or indirectly on the question of map representation. This fundamental choice dramatically impacts the choice of processing blocks in the SLAM pipeline, as well as all other downstream tasks that depend on the outpus of the SLAM system. Of late, gradient-based learning approaches have transformed the outlook of several domains (Eg. image recognition [117], language modeling [118], speech recognition [119]). However, such techniques have had

¹Video abstract: https://youtu.be/2ygtSJTmo08



Fig. 4.1. ∇ SLAM (gradSLAM) is a *fully differentiable* dense simultaneous localization and mapping (SLAM) system. The central idea of ∇ SLAM is to construct a computational graph representing every operation in a dense SLAM system. We propose differentiable alternatives to several non-differentiable components of traditional dense SLAM systems, such as optimization, odometry estimation, raycasting, and map fusion. This creates a pathway for gradient-flow from 3D map elements to sensor observations (e.g., *pixels*). We implement differentiable variants of three dense SLAM systems that operate on voxels, surfels, and pointclouds respectively. ∇ SLAM thus is a novel paradigm to integrate representation learning approaches with classical SLAM.

limited success in the context of SLAM, primarily since many of the elements in the standard SLAM pipeline are not differentiable. A fully differentiable SLAM system would enable task-driven representation learning since the error signals indicating task performance could be back-propagated all the way through the SLAM system, to the raw sensor observations.

This is particularly true for *dense 3D maps* generated from RGB-D cameras, where there has been a lack of consensus on the right representation (pointclouds, meshes, surfels, etc.). Several methods have demonstrated a capability for producing dense 3D maps from sequences of RGB or RGB-D frames [85, 86, 87]. However, none of these methods are able to solve the *inverse mapping* problem, i.e., answer the question: "How much does a specific pixel-measurement contribute to the resulting 3D map"? Formally, we desire an the expression that relates a pixel in an image (or in general, a sensor measurement s) to a 3D map \mathcal{M} of the environment. We propose to solve this through the development of a differentiable mapping function $\mathcal{M} = \mathcal{G}_{SLAM}(s)$. Then the gradient of that mapping $\nabla_s \mathcal{M}$ can intuitively tell us that perturbing the sensor measurement s by an infinitesimal δs causes the map \mathcal{M} to change by $\nabla_s \mathcal{G}_{SLAM}(s) \delta s$.

Central to our goal of realizing a fully differentiable SLAM system are *computational* graphs, which underlie most gradient-based learning techniques. We make the observations that, if an entire SLAM system can be decomposed into elementary operations, all of which are differentiable, we could compose these elementary operations² to preserve differentiability. However, modern *dense* SLAM systems are quite sophisticated, with several non-differentiable subsystems (optimizers, raycasting, surface mapping), that make such a construct challenging.

 $^{^2\}mathrm{Again},$ using differentiable composition operators.

We propose ∇ SLAM (gradSLAM), a differentiable computational graph view of SLAM. We show how all non-differentiable functions in SLAM can be realised as smooth mappings. First, we propose a differentiable trust region optimizer for nonlinear least squares systems. Building on it, we present differentiable strategies of mapping, raycasting, and global measurement fusion.

The ∇ SLAM framework is very general, and can be extended most dense SLAM systems for differentiability. In Sec. 4.4, we provide three examples of SLAM systems that can be realized as differentiable computation graphs: implicit-surface mapping (Kinectfusion [85]), surfel-based mapping (PointFusion [87]), and iterative closest point (ICP) mapping (ICP-SLAM). We show that the differentiable approaches maintain similar performance to their non-differentiable counterparts, with the added advantage that they allow gradients to flow through them.

To foster further research on differentiable SLAM systems and their applications to spatially-grounded learning, ∇ SLAM is available as an open-source PyTorch framework. Our project page and code can be accessed at https://gradslam.github.io.

4.2. Related Work

Several works in recent years have applied recent machine learning advances to SLAM or have reformulated a subset of *components* of the full SLAM system in a differentiable manner.

4.2.1. Learning-based SLAM approaches

There is a large body of work in deep learning-based SLAM systems. For example, CodeSLAM [99] and SceneCode [120] attempt to represent scenes using compact *codes* that represent 2.5D depth maps. DeepTAM [121] trains a tracking network and a mapping network, which learn to reconstruct a voxel representation from a pair of images. CNN-SLAM [98] extends LSD-SLAM [89], a popular monocular SLAM system, to use single-image depth predictions from a convnet. Another recent trend has been to try to formulate the SLAM problem over higher level features such as objects, which may be detected with learned detectors [122][123][124]. DeBrandandere et al. [125] perform lane detection by backpropagating least squares residuals into a frontend module. Recent work has also formulated the passive [126] and active localization problems [127, 128] in an end-to-end differentiable manner. While all of these approaches try to leverage differentiability in submodules of SLAM systems (eg. odometry, optimization, etc.), there is no single framework that models an entire SLAM pipeline as a differentiable graph.

4.2.2. Differentiable visual odometry

The beginnings of differentiable visual odometry can be traced back to the seminal Lucas-Kanade iterative matching algorithm [129]. Kerl *et al.* [83]³ apply the Lucas-Kanade algorithm to perform real-time dense visual odometry. Their system is differentiable, and has been extensively used for self-supervised depth and motion estimation [130, 97, 131]. Coupled with the success of Spatial Transformer Netowrks (STNs) [132], several libraries (gvnn [133], kornia [134]) have since implemented these techniques as differentiable *layers*, for use in neural networks.

However, extending differentiability beyond the two-view case (*frame-frame alignment*) is not straightforward. Global consistency necessitates fusing measurements from live frames into a global model (*model-frame alignment*), which is not trivially differentiable.

4.2.3. Differentiable optimization

Some approaches have recently proposed to learn the optimization of nonlinear least squares objective functions. This is motivated by the fact that similar cost functions have similar loss landscapes, and learning methods can help converge faster, or potentially to better minima.

In BA-Net [135], the authors learn to predict the damping coefficient of the Levenberg-Marquardt optimizer, while in LS-Net [136], the authors entirely replace the Levenberg-Marquard optimizer by an LSTM netowrk [137] that predicts update steps. In GN-Net [138], a differentiable version of the Gauss-Newton loss is used to show better robustness to weather conditions. RegNet [139] employs a learning-based optimization approach based on photometric error for image-to-image pose registration. However, all the aforementioned approaches require the training of additional neural nets and this requirement imposes severe limitations on the generalizability. OptNet [140] introduces differentiable optimization layers for quadratic programs, that do not involve learnable parameters.

Concurrently, Grefenstette *et al.* [141] propose to unroll optimizers as computational graphs, which allows for computation of arbitrarily higher order gradients. Our proposed differentiable Levenberg-Marquardt optimizer is similar in spirit, with the addition of gating functions to result in better gradient flows.

In summary, to the best of our knowledge, there is no *single* approach that models the entire SLAM pipeline as a differentiable model, and it is this motivation that underlies ∇ SLAM.

³The formulation first appeared in Steinbrüker *et al.* [82].

4.3. ∇ SLAM

In this section we will overview our proposed method for ∇ SLAM and also detail the individual differentiable sub-components.

4.3.1. Preliminaries: Computational graphs

In gradient-based learning architectures, all functions and approximators are conventionally represented as *computational graphs*. Formally, a computation graph is a directed acyclic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where each node $v \in \mathcal{V}$ holds an operand or an operator, and each (directed) edge $e \in \mathcal{E}$ indicates the control flow in the graph. Further, each node in the graph also specifies computation rules for the gradient of the outputs of the node with respect to the inputs to the node. Computational graphs can be nested and composed



Fig. 4.2. A computational graph. Nodes in red represent variables. Nodes in blue represent operations on variables. Edges represent data flow. This graph computes the function 3(xy + z). Dashed lines indicate (local, i.e., per-node) gradients in the backward pass.

in about any manner, whilst preserving differentiability. An example computation graph for the function 3(xy + z) is shown in Fig. 4.2.

In a standard SLAM pipeline there are several subsystems/components that are not differentiable (i.e., for a few forward computations in the graph, gradients are unspecifiable). For example, in the context of dense 3D SLAM [85][87], nonlinear least squares modules, raycasting routines, and discretizations are non-diffrentiable. Further, for several operations such as index selection / sampling, gradients exist, but are zero *almost everywhere*, which result in extremely sparse gradient flows.

4.3.2. Method Overview

The objective of ∇ SLAM is to make every computation in SLAM exactly realised as a composition of differentiable functions⁴. Broadly, the sequence of operations in dense SLAM systems can be termed as *odometry estimation* (frame-to-frame alignment), *map building* (model-to-frame alignment/local optimization), and *global optimization*. An overview of the approach is shown in 4.1.

First, we provide a description of the precise issues that render nearly all of the aforementioned modules non-differentiable, and propose differentiable counterparts for each module. Finally, we show that the proposed differentiable variants allow the realization of several

⁴Wherever exact differentiable realizations are not possible, we desire *as-exact-as-possible* realizations.

classic dense mapping algorithms (*KinectFusion* [85], *PointFusion* [87], ICP-SLAM) in the ∇ SLAM framework.⁵

4.3.3. ∇ LM: A differentiable nonlinear least squares solver

Most state-of-the-art SLAM solutions optimize nonlinear least squares objectives to obtain local/globally consistent estimates of the robot state and the map. Such objectives are of the form $\frac{1}{2} \sum \mathbf{r}(\mathbf{x})^2$, where $\mathbf{r}(\mathbf{x})$ is a nonlinear function of residuals. Example application scenarios that induce this nonlinear



Fig. 4.3. Computational graph for ∇LM

least squares form include visual odometry, depth measurement registration (e.g., ICP), and pose-graph optimization among others. Such objective functions are minimized using a succession of linear approximations $(\mathbf{r}(\mathbf{x} + \delta \mathbf{x})|_{\mathbf{x}=\mathbf{x}_0} = \mathbf{r}(\mathbf{x}_0) + \mathbf{J}(\mathbf{x}_0)\delta \mathbf{x})$, using Gauss-Newton (GN) or Levenberg-Marquardt (LM) solvers. GN solvers are extremely sensitive to intialization, numerical precision, and moreover, provide no guarantees on *non-divergent* behavior. Hence most SLAM systems use LM solvers.

Trust-region methods (such as LM) are not differentiable as at each optimization step, they involve recalibration of optimizer parameters, based on a *lookahead* operation over subsequent iterates [142]. Specifically, after a new iterate is computed, LM solvers need to make a *discrete* decision between damping or undamping the linear system. Furthermore, when undamping, the iterate must be restored to its previous value. This discrete *switching* behavior of LM does not allow for gradient flow in the backward pass.

We propose a computationally efficient soft reparametrization of the damping mechanism to enable differentiability in LM solvers. Our key insight is that, if $\mathbf{r_0} = \mathbf{r}(\mathbf{x_0})^{\mathbf{T}}\mathbf{r}(\mathbf{x_0})$



Fig. 4.4. An example curve fitting problem, showing that ∇LM performs near-identical to LM, with the added advantage of being fully differentiable.

is the norm of the error at the current iterate, and $\mathbf{r_1} = \mathbf{r}(\mathbf{x_1})^T \mathbf{r}(\mathbf{x_1})$ is the norm of the error at the *lookahead* iterate, the value of $\mathbf{r_1} - \mathbf{r_0}$ determines whether to damp or to undamp. And, only when we choose to undamp, we revert to the current iterate. We define two smooth

 $^{^{5}}$ That is, realizable as *fully* differentiable computational graphs.



Fig. 4.5. Computation graph for the differentiable mapping module. The uncolored boxes indicate intermediate variables, while the colored boxes indicate processing blocks. Note that the specific choice of the functions for update surface measurement and map fusion depend on the map representation used.

gating functions Q_x and Q_λ based on the generalized logistic function [143] to update the iterate and determine the next damping coefficient.

$$\lambda_{1} = Q_{\lambda}(r_{0}, r_{1}) = \lambda_{min} + \frac{\lambda_{max} - \lambda_{min}}{1 + De^{-\sigma(r_{1} - r_{0})}}$$

$$Q_{x}(r_{0}, r_{1}) = x_{0} + \frac{\delta x_{0}}{1 + e^{-(r_{1} - r_{0})}}$$
(4.3.1)

where D and σ are tunable parameters that control the falloff [143]. Also $[\lambda_{min}, \lambda_{max}]$ is the range of values the damping function can assume. Notice that this smooth parameterization of the LM update allows the optimizer to be expressed as a fully differentiable computational graph (Fig. 4.3).

It must be noted that this scheme can be modified to accommodate other kinds of gating functions, such as hyperbolic curves. We however, choose the above gating functions, as they provide sufficient flexibility. A thorough treatment of the impact of the choice of gating functions on performance is left for future work.

4.3.4. Differentiable mapping

Another non-smooth operation in dense SLAM is map construction (*surface measurement*). For example, consider a *global* map \mathcal{M} being built in the reference frame of the first imagesensor measurement I_0 . When a new frame I_k arrives at time k, dense SLAM methods need to align the surface measurement being made in the live frame, with the map \mathcal{M} . Notwithstanding the specific choice of map representation (i.e., pointclouds, signed-distances, surfels), a generic *surface alignment* process comprises the following steps.

- (1) The map \mathcal{M} is intersection-tested with the live frame, to determine the *active set* \mathcal{M}_a of map elements, and the active set of image pixels \mathcal{P}_a . The remaining map elements are *clipped*.
- (2) Active image pixels \mathcal{P}_a are checked for measurement validity (e.g., missing depth values / blurry pixels, etc.). This results in a *valid active set* of image pixels \mathcal{P}_{valid}

- (3) The set of pixels in \mathcal{P}_{valid} is backprojected to 3D and compared with the map. At this stage, it must be discerned whether these pixels measure existing elements in \mathcal{M}_a , or if they measure a new set of elements that need to be added to the global map.
- (4) Once the above decision is made, these surface measurements are *fused* into the global map. The choice of the fusion mechanism is dependent on the underlying representation of each map element (points, surfels, TSDF, etc.).

The above process involves a number of differentiable yet non-smooth operations (clipping, indexing, thresholding, new/old decision, active/inactive decision, etc.). Although the above sequence of operations can be represented as a computation graph, it will not necessarily serve our purpose here since, even though (local) derivatives can be defined for operations such as clipping, indexing, thresholding, and discrete decisions, these derivatives exist only at that single point. The overall function represented by the computation graph will have undefined gradients "almost everywhere" (akin to step functions). We propose to mitigate this issue by making the functions locally *smooth*. Concretely, we propose the following corrective measures.

- (1) The surface measurement made by each valid pixel p in the live frame (i.e., $p \in \mathcal{P}_{valid}$) is not a function of p alone. Rather, it is the function of the pixel p and its (active/inactive) neighbours nbd(p), as determined by a kernel K(p,nbd(p)).
- (2) When a surface measurement is transformed to the global frame, rather than using a hard (one-one) association between a surface measurement and a map element, we use a soft association to multiple map elements, in accordance with the sensor characteristics.
- (3) Every surface measurement is, by default, assumed to represent a new map element, which is passed to a *differentiable fusion* step (*cf.* Sec 4.3.5).

The kernel K(p, nbd(p)) can be a discrete approximation (e.g., constant within a pixel) or can vary at the subpixel level, based on the choice of the falloff function. For faster computation and coarse gradients, we use a bilinear interpolation kernel. While bilinear interpolation is a sensible approximation for image pixels, this is often a poor choice for use in 3D *soft* associations. For forming 3D associations, we leverage characteristics of RGB-D sensors in defining the soft falloff functions. Specifically, we compute, for each point P in the live surface measurement, a set of closest candidate points in a region $exp\left(-\frac{r(P)^2}{2\sigma^2}\right)$, where r(P) is the radial depth of the point from the camera ray, and σ affects the falloff region.⁶

4.3.5. Differentiable map fusion

The aforementioned differentiable mapping strategy, while providing us with a smooth observation model, also causes an undesirable effect: the number of map elements increases

 $[\]overline{^{6}$ This is a well-known falloff function, usually with Kinect-style depth sensors [87, 144, 145].

in proportion with exploration time. However, map elements should ideally increase with proportion to the *explored volume of occupied space*, rather than with exploration time. Conventional dense mapping techniques (e.g., KinectFusion [85], PointFusion [87]) employ this through *fusion* of redundant observations of the same map element. As a consequence, the recovered map has a more manageable size, but more importantly, the reconstruction quality improves greatly. While most fusion strategies are differentiable (eg. [85, 87]), they impose falloff thresholds that cause an abrupt change in gradient flow at the truncation point. We use a logistic falloff function, similar to Eq. 4.3.1, to ease gradient flow through these truncation points.

4.3.6. Differentiable ray backprojection

Some dense SLAM systems [85, 86] perform global pose estimation by raycasting a map to a live frame. Such an operation inherently involves non-differentiable steps. First, from each pixel in the image, a ray from the camera is backprojected, and its intersection with the first map element along the direction of the ray is determined. This involves marching along the ray until a map element is found, or until we exit the bounds of reconstruction. Usual (non-differentiable) versions of ray marching use max-min acceleration schemes [146] or rely on the existence of volumetric signed distance functions [85]. Several attempts have been made to make the raycasting operation differentiable. Scene representation networks [147] proposes to predict ray marching steps using an LSTM. In other works such as DRC [148] and WS-GAN [149], the authors pool over all voxels



Fig. 4.6. Ray differentials: Inset shows the computation graph of the ray value computation. The dashed rectangle is not differentiable, and its derivatives are approximated as shown in Eq 4.3.2

along a ray to compute the *potential* of a ray. In this work, we make one enhancement to the ray pooling operation. We pool over all voxels along a ray, but have a Gaussian falloff defined around the depth measurement of the image pixel through which the ray passes. Further, we use finite differences to compute the derivative of the ray potential with respect to the pixel neighbourhood. We use the finite differences based ray differentials defined in Igehy *et al.* [150]. If p_c is the image pixel that the ray R_c pierces, and $\mathcal{V}_c = \{v_c\}$ is the set of all voxels it

pierces, then the aggregated value of the ray is denoted v_c (with respect to an aggregation function $\Phi(\psi(v_c) \ \forall v_c \in \mathcal{V}_c)$). The aggregation function simply multiplies each value $\psi(v_c)$ with the density of the Gaussian fallof at v_c , and normalizes them. Similarly v_l , v_r , v_u , and v_b are the aggregated values of rays emanating from the pixels to the left, right, above, and below p_c respectively. Then, the partial derivative $\frac{\partial v_c}{\partial c}$ can be approximated as

$$\frac{\partial v_c}{\partial p_c} = \begin{pmatrix} (v_r - v_l)/2\\ (v_u - v_d)/2 \end{pmatrix}$$
(4.3.2)

An illustration of the ray differential computation scheme can be found in Fig. 4.6.

4.4. Case Studies: KinectFusion, PointFusion, and *ICP-SLAM*

To demonstrate the applicability of the ∇ SLAM framework, we leverage the differentiable computation graphs specified in Sec 4.3 and compose them to realise three practical SLAM solutions. In particular, we implement differentiable versions of the *KinectFusion* [85] algorithm that constructs TSDF-based volumetric maps, the *PointFusion* [87] algorithm that constructs surfel maps, and a pointcloud-only SLAM framework that we call *ICP-SLAM*.

4.4.1. KinectFusion

Recall that KinectFusion [85] alternates between *tracking* and *mapping* phases. In the tracking phase, the entire up-to-date TSDF volume is raycast onto the live frame, to enable a point-to-plane ICP that aligns the live frame to the raycast model. Subsequently, in the mapping phase, surface measurements from the current frame are *fused* into the volume, using the TSDF fusion method proposed in [85]. The surface measurement is given as (cf.[85])

$$sdf(p) = trunc(\|K^{-1}x\|_{2}^{-1}\|t-p\|_{2} - depth(x))$$
$$trunc(sdf) = min(1, \frac{sdf}{\mu})(sign(sdf)) \quad iff \ sdf \ \ge -\mu$$

$$(4.4.1)$$

Here, p is the location of a voxel in the camera frame, and $x = \lfloor \pi(Kp) \rfloor$ is the live frame pixel to which p projects to. μ is a parameter that determines the threshold beyond which a surface measurement is invalid. However, we note that the *floor* operator is non-differentiable "almost everywhere". Also, the truncation operator, while differentiable within a distance of μ from the surface, is abruptly truncated, which hinders gradient flow . Instead, we again use a generalized logistic function [143] to create a smooth truncation, which provides better-behaved gradients at the truncation boundary. The other steps involved here, such as raycasting, ICP, etc. are already differentiable in the ∇ SLAM framework (*cf.*Sec 4.3). Fusion
of surface measurements is performed using the same approach as in [85] (weighted averaging of TSDFs).

4.4.2. PointFusion

As a second example, we implement PointFusion [87], which incrementally fuses surface measurements to obtain a global surfel map. Surfel maps compare favourably to volumetric maps due to their reduced memory usage.⁷ We closely follow our differentiable mapping formulation (*cf.*Sec 4.3.4) and use surfels as map elements. We adopt the fusion rules from [87] to perform map fusion.

4.4.3. ICP-SLAM

As a baseline example, we implement a simple pointcloud based SLAM technique, which uses ICP to incrementally register pointclouds to a global pointcloud set. In particular, we implement two systems. The first one aligns every pair of consecutive incoming frames, to obtain an odometry estimate (also referred to as *frame-to-frame alignment* or ICP-Odometry). The second variant performs what we call *frame-to-model alignment* (ICP-SLAM). That is, each incoming frame is aligned (using ICP) with a pointcloud containing the entire set of points observed thus far.

$T_{max} = 10$ iters		Expor	iential			S_{i}	ine			Si	nc	
	GD	GN	LM	∇LM	GD	\mathbf{GN}	LM	∇LM	GD	GN	$\mathbf{L}\mathbf{M}$	∇LM
$ a_{pred} - a_{gt} ^2$	0.422	0.483	0.483	0.483	0.379	0.341	0.342	0.342	2.929	0.304	0.304	0.304
$ t_{pred} - t_{gt} ^2$	0.606	0.50	0.550	0.550	0.222	0.359	0.360	0.360	3.024	0.304	0.304	0.040
$\ w_{pred} - w_{gt}\ ^2$	1.268	0.667	0.075	0.075	1.215	0.080	0.084	0.085	0.462	10 ⁻⁷	0.023	10^{-4}
$ f(x)_{pred} - f(x)_{gt} ^2$	0.716	0.160	0.163	0.160	0.666	0.148	0.152	0.148	0.700	$5 imes 10^{-8}$	0.005	4×10^{-5}
$T_{max} = 50$ iters												
$ a_{pred} - a_{gt} ^2$	0.365	0.275	0.231	0.275	0.486	0.429	0.434	0.434	3.329	0.380	0.380	0.380
$ t_{pred} - t_{gt} ^2$	0.263	0.219	0.231	0.218	0.519	0.455	0.459	0.460	2.739	0.380	0.380	0.380
$ w_{pred} - w_{gt} ^2$	1.220	0.205	0.007	0.369	1.327	0.273	0.376	0.383	0.383	$2 imes 10^{-7}$	0.202	4×10^{-5}
$ f(x)_{pred} - f(x)_{gt} ^2$	0.669	0.083	0.004	0.078	0.673	0.153	0.153	0.151	0.795	$2 imes 10^{-7}$	0.005	3×10^{-5}
$T_{max} = 100$ iters												
$ a_{pred} - a_{gt} ^2$	0.431	0.475	0.480	0.487	0.486	0.429	0.434	0.434	2.903	0.196	0.196	0.196
$ t_{pred} - t_{gt} ^2$	0.466	0.311	0.378	0.323	0.519	0.455	0.459	0.460	2.847	0.196	0.196	0.196
$\ w_{pred} - w_{gt}\ ^2$	1.140	0.364	0.066	0.065	1.327	0.273	0.376	0.382	0.601	10^7	0.026	9×10^{-5}
$ f(x)_{pred} - f(x)_{gt} ^2$	0.662	0.243	0.162	0.230	0.673	0.153	0.153	0.151	0.707	$6 imes 10^{-8}$	0.005	4×10^{-5}

Table 4.1. ∇ **LM** performs quite similarly to its non-differentiable counterpart, on a variety of non-linear functions, and at various stages of optimization. Here, **GD**, **GN**, and **LM** refer to gradient descent, Gauss-Newton, and Levenberg-Marquardt optimizers respectively.

 $[\]overline{^{7}\text{On the flipside}}$, surfel-based algorithms are harder to parallelize compared to volumetric fusion.

4.5. Experiments and results

4.5.1. Differentiable optimization

In Sec 4.3.3, we introduced two generalized logistic functions Q_{λ} and Q_x to compute the damping functions as well as the subsequent iterates. We conduct multiple experiments to verify the impact of this approximation on the performance (convergence speed, quality of solution) of nonlinear least squares solvers.

We first design a test suite of nonlinear curve fitting problems (inspiration from [136]), to measure the performance of ∇ LM to its non-differentiable counterpart. We consider three nonlinear functions, *viz. exponential*, *sine*, and *sinc*, each with three parameters *a*, *t*, and *w*.

$$f(x) = a \exp\left(-\frac{(x-t)^2}{2w^2}\right)$$

$$f(x) = \sin(ax + tx + w)$$

$$f(x) = \operatorname{sinc}(ax + tx + w)$$
(4.5.1)

For each of these functions, we uniformly sample the parameters $p = \{a, t, w\}$ to create a suite of ground-truth curves, and uniformly sample an initial guess $p_0 = \{a_0, t_0, w_0\}$ in the interval [-6, 6]. We sample 100 problem instances for each of the three functions. We run a variety of optimizers (such as gradient descent (GD), Gauss-Newton (GN), LM, and ∇ LM) for a maximum of 10, 50, and 100 iterations. We compute the mean squared error in *parameter space* (independently for each parameter a, t, w) as well as in *function space* (i.e., $||f(x)_{pred} - f(x)_{gt}||^2$. Note that these two errors are not necessarily linearly related, as the interaction between the parameters and the function variables are highly nonlinear. The results are presented in Table ??. It can be seen that ∇ LM performs near-identically to LM.



Fig. 4.7. ∇ LM performs comparably to LM optimizers. In this figure, we show example curve fitting problems from the test suite.

4.5.2. Comparitive analysis of case studies

In Sec 4.4, we implemented KinectFusion [85], PointFusion [87], and ICP-SLAM as differentiable computational graphs. Here, we present an analysis of how each of the approaches compare to their non-differentiable counterparts. Table 4.2 shows the trajectory tracking performance of the non-differentiable and differentiable (∇) versions of ICP-Odometry, ICP-SLAM, and PointFusion. We observe no virtual change in performance when utilizing the differentiable mapping modules and ∇ LM for optimization. This is computed over split subsets of the living_room_traj0 sequence.

We also evaluate the reconstruction quality of ∇ -KinectFusion with that of Kintinuous [86]. On a subsection of the living_room_traj0 sequence of the ICL-NUIM [151] benchmark, the surface reconstruction quality of Kintinuous is 18.625, while that of differentiable KinectFusion is 21.301 (better). However, this quantity is misleading, as Kintinuous only retains a subset of high confidence points in the extracted mesh, while our differentiable KinectFusion outputs (see Fig. 4.8) contain a few noisy artifacts, due to our smooth truncation functions.



Fig. 4.8. Qualitative results: On the living room lr kt0 sequence of the ICL-NUIM dataset [151]. The reconstructions are near-identical to their non-differentiable counterparts. However, distinct from classic SLAM approaches, these reconstructions allow for gradients to flow from a 3D map element all the way to the entire set of pixel-space measurements of that element.

4.5.3. Qualitative results

 ∇ SLAM works out of the box on multiple other RGB-D datasets. Specifically, we present qualitative results of running our differentiable SLAM systems on RGB-D sequences from the TUM RGB-D dataset [152], ScanNet [153], as well as on an in-house sequence captured from an Intel RealSense D435 camera.

Fig. 4.9- 4.11 show qualitative results obtained by running ∇ SLAM on a variety of sequences from the TUM RGB-D benchmark (Fig. 4.9), ScanNet (Fig. [153]), and an inhouse sequence (Fig. 4.11). These differentiable SLAM systems all execute fully on the GPU, and are capable of computing gradients with respect to *any* intermediate variable (Eg. camera poses, pixel intensities/depths, optimization parameters, camera intrinsics, etc.).

Method	ATE	RPE
$\begin{array}{c} \text{ICP-Odometry (non-differentiable)} \\ \nabla \text{ICP-Odometry} \end{array}$	0.029 0.01664	0.0318 0.0237
ICP-SLAM (non-differentiable) ∇ ICP-SLAM	0.0282 0.01660	0.0294 0.0204
PointFusion (non-differentiable) ∇ PointFusion	0.0071 0.0072	0.0099 0.0101
$\begin{array}{c} {\rm KinectFusion} \ ({\rm non-differentiable}) \\ {\bf \nabla} {\rm KinectFusion} \end{array}$	0.013 0.016	0.019 0.021

Table 4.2. Performance of ∇ SLAM. The differentiable counterparts perform nearly similar to their non-differentiable counterparts (ATE: Absolute Trajectory Error, RPE: Relative Pose Error).



Fig. 4.9. Reconstruction obtained upon running the differentiable ICP-Odometry pipeline on a subsection of the rgbd_dataset_freiburg1_xyz sequence.

4.5.4. Analysis of Gradients

The computational graph approach of ∇ SLAM allows us to recover meaningful gradients of 2D (or 2.5D) measurements with respect to a 3D surface reconstruction. In Fig. 4.12, the top row shows an RGB-D image differentiably transformed—using ∇ SLAM—into a (noisy) TSDF surface measurement, and then compared to a more precise global TSDF map. Elementwise comparision of aligned volumes gives us a reconstruction error, whose gradients are backpropagated through to the input depthmap using the computational graph maintained by ∇ SLAM (and visualized in the depth image space). In the second row, we intentionally introduce an occluder that masks out a small (40 × 40) region in the RGB-D image, thereby introducing a reconstruction artifact. Computing the volumetric error between the global and local occluded TSDF volumes and inspecting the gradients with respect to the input indicates the per pixel contribution of the occluding surface to the volumetric



Fig. 4.10. Qualitative results on sequences from the ScanNet [153] dataset. Owing to GPU memory constraints, we use each of the differentiable SLAM systems (∇ KinectFusion, ∇ PointFusion, and ∇ ICP-SLAM) to reconstruct parts of the scene. We also show outputs from BundleFusion [154] for reference.

error. Thus, $\nabla SLAM$ provides a rich interpretation of the computed gradients: they denote the contribution of each pixel towards the eventual 3D reconstruction.

4.5.5. Application: RGB and depth completion

In Fig. 4.13, we similarly introduce such occluders (top row) and pixel noise (bottom row) in one of the depth maps of a sequence and reconstruct the scene using ∇ PointFusion. We then calculate the chamfer distance between the noisy and true surfel maps and backpropogate the error with respect to each pixel. The minimized loss leads to the targeted recovery of the noisy and occluded regions. We additionally show an RGB-D image completion task (from uniform noise)in Fig. 4.14.



Fig. 4.11. In-house sequence collected from an Intel RealSense D435 camera. The reconstruction (right) is obtained by running ∇ PointFusion. Note that we do not perform any noise removal. Differentiable noise filtering is left for future work.



Fig. 4.12. Analysis of gradients: ∇ SLAM enables gradients to flow all the way back to the input images. *Top*: An RGB-D image pair (depth not shown) is passed through ∇ SLAM, and reconstruction error is computed using a precise fused map. Backpropagation passes these gradients all the way back to the depth map (blue map). *Bottom*: An explicit occluder added to the center of the RGB-D pair. This occluder distorts the construction by creating a gaping hole through it. But, using the backpropagated gradients, one can identify the set of image/depthmap pixels that result in a particular area to be reconstructed imperfectly.

4.6. Conclusion

We introduce ∇ SLAM, a differentiable computational graph framework that enables gradient-based learning for a large set of localization and mapping based tasks, by providing explicit gradients with respect to the input image and depth maps. We demonstrate a diverse set of case studies, and showcase how the gradients propogate throughout the tracking, mapping, and fusion stages. Future efforts will enable ∇ SLAM to be directly plugged into and optimized in conjunction with downstream tasks. ∇ SLAM can also enable a variety of self-supervised learning applications, as any gradient-based learning architecture can now be equipped with a sense of *spatial understanding*.



Fig. 4.13. End-to-end gradient propagation: (Top): A chunk of a depth map is *chopped*. The resultant sequence is reconstructed using ∇ PointFusion and the pointcloud is compared to a *clean* one reconstructed using the unmodified depth map. The Chamfer distance between these two pointclouds is used to define a reconstruction error between the two clouds, which is backpropagated through to the input depth map and updated by gradient descent. (*Bottom*): Similar to the Fig. 4.12, we show that ∇ SLAM can *fill-in* holes in the depthmap by leveraging multi-view gradient information.



Fig. 4.14. RGB-D completion using end-to-end gradient propagation: Three RGB-D images and a *noise image* are passed through ∇ PointFusion, and compared to a clean reconstruction obtained from four RGB-D images. The reconstruction loss is used to optimize the *noise image* by gradient descent. We can recover most of the artifacts from the raw RGB and depth images. Note that finer features are hard to recover from a random initialization, as the overall *SLAM function* is only locally differentiable.

Chapter 5

Prologue to Article 2

This article was published and presented at the Conference on Robot Learning (CoRL), 2021.

Authors: Christopher Agia^{*}, Krishna Murthy Jatavallabhula^{*}, Mohamed Khodeir, Ondrej Miksik, Vibhav Vineet, Mustafa Mukadam, Liam Paull, Florian Shkurti

Contribution statement: This work was jointly led by Christopher Agia and the author. The ideas were primarily conceived by the author and Christopher Agia over regular meetings. Christopher Agia focused on developing the dataset and the benchmark, while the author focused on developing the SCRUB and SEEK algorithms. Critical aspects of both the benchmark and the contributed algorithms were pair programmed by Christopher Agia and the author. Mohamed Khodeir was influential in expanding our planning suite to include several optimal planners, and in generating visualizations and plots for the manuscript. All the authors contributed towards manuscript preparation and review.

Note: This paper is presented as-is, with minor cosmetic changes to adhere to the Universite de Montreal thesis template.

Article 2: Taskography: Evaluating robot task planning over large 3D scene graphs

Abstract

3D scene graphs (3DSGs) are an emerging description; unifying symbolic, topological, and metric scene representations. However, typical 3DSGs contain hundreds of objects and symbols even for small environments; rendering task planning on the *full* graph impractical. We construct **Taskography**, the first large-scale robotic task planning benchmark over 3DSGs. While most benchmarking efforts in this area focus on vision-based planning, we systematically study *symbolic* planning, to decouple planning performance from visual representation learning. We observe that, among existing methods, neither classical nor learning-based planners are capable of real-time planning over full 3DSGs. Enabling realtime planning demands progress on both (a) sparsifying 3DSGs for tractable planning and (b) designing planners that better exploit 3DSG hierarchies. Towards the former goal, we propose SCRUB, a task-conditioned 3DSG sparsification method; enabling classical planners to match and in some cases surpass state-of-the-art learning-based planners. Towards the latter goal, we propose SEEK, a procedure enabling learning-based planners to exploit 3DSG structure, reducing the number of replanning queries required by current best approaches by an order of magnitude. We will open-source all code and baselines to spur further research along the intersections of robot task planning, learning and 3DSGs.

6.1. Introduction

Real-world robotic task planning problems in large environments require reasoning over tens of thousands of object-action pairs. Faced with long-horizon tasks and an abundance of choices, state-of-the-art task planners struggle with an efficiency-reliability trade-off in grounding actions towards the goal. Hence, designing actionable scene abstractions suitable for a range of robotic tasks has drawn long-standing attention from the robotics and computer vision communities [155, 156, 157, 158, 159, 160].

A promising approach for building symbolic abstractions from raw perception data are 3D scene graphs (3DSGs, see Fig. 6.1) [161, 162, 163] – hierarchical representations of a scene that capture metric, semantic, and relational information, such as affordances, properties, and relationships among scene entities. While 3DSGs have to date been applied to simpler planning problems like goal-directed navigation [160, 164], active object search [165], and node classification [166], their amenability to more complex robotic task planning problems has yet to be thoroughly evaluated.

To investigate the joint application of 3DSGs and modern task planners to complex robotics tasks we propose **Taskography**: the first large-scale benchmark comprising a number of challenging task planning domains designed for 3DSGs. Analyzing planning times and costs on a diversity of domains in TASKOGRAPHY reveals that neither classical nor learning-based planners are capable of real-time planning over full 3DSGs, however, that they become so only when 3DSGs are sparsified.

Many real-world problems only require reasoning over a small subset of scene objects. E.g., the task "*fetch a mug from the kitchen*" primarily involves reasoning about scene elements associated with mugs or kitchens, rendering a vast majority of the remaining environment contextually irrelevant. Most planners aren't able to exploit such implicitly defined task contexts, instead spending most of their computation time reasoning about extraneous scene attributes and actions [36] (see Fig. 6.5).

We argue that performant task planning over 3DSGs demands progress on two fronts: (a) sparsifying 3DSGs to make planning problems tractable, and (b) designing task planners that exploit the spatial hierarchies encapsulated in 3DSGs. To address (a), we present scrub—a planner-agnostic strategy guaranteed to produce a minimal *sufficient* object set for grounded planning problems. That is, planning on this reduced subset of scene entities suffices to solve the planning problem defined over the full 3DSG. Classical planning over state spaces (3DSGs) augmented by SCRUB outperforms state-of-the-art learning-based planners on the majority of tasks on our benchmark, without requiring any prior learning, establishing a strong baseline for future work in robotic task planning. To address (b), we present seek: a procedure tailored to 3DSGs, which supplements learning-based incremental planners by imposing 3DSG structure, ensuring all objects in the *sufficient* set are reachable from the start state. In our experiments, augmenting state-of-the-art planners with SEEK results in computational savings and an order of magnitude fewer replanning iterations.

In summary, we make the following contributions:

- TASKOGRAPHY: a large-scale benchmark to evaluate robotic task planning over 3DSGs,
- SCRUB: a planner-agnostic strategy to adapt 3DSGs for performant planning,

• SEEK: a procedure that enables learning-based planners to better exploit 3DSGs

We will open-source all code and baselines in TASKOGRAPHY-API, enabling the construction of new task planning domains, and benchmarking the performance of newer learning-based planners.

6.2. Related work

Early research in **symbolic planning** was centered around *optimal* planning [27, 28, 29, 30, 38]; planners producing solutions that preserve cost or plan length optimality. These methods are computationally expensive and thereby untenable to even moderately sized problems. This spurred work on *satisficing* planners that forgo optimal solutions for cheaper, feasible plans. Notable paradigms include regression planning [24], tree search [167], and heuristic search [34, 25, 26, 168, 169]. Whilst the many successes of heuristic planners [170, 171], computing low-cost informative heuristics is deterred by many extraneous objects [40, 36]; an inauspicious characteristic of large 3DSGs.

Robot task planning techniques have focused on constructing more effective representations to plan upon [172, 173, 174]. There are also approaches that integrate task and motion planning [56, 175, 176]–further demonstrated in hierarchical task space [177]–but which fall outside the scope of our work. Several approaches exploit task hierarchies for robot task planning [178] and control [179, 180, 181]. Different from these, our work focuses on exploiting abstractions in *spatial structure* encapsulated in 3DSGs, not to be conflated with hierarchical planners that exploit *task structure* [182].

State-of-the-art learning-based planners have demonstrated promising performance in small-to-moderate problem sizes. However, techniques such as relational policy learning [183], relational heuristic learning [184], action grounding [185], program guided symbolic planning [186, 187, 188, 189, 190], and regression planning networks [44] fail in large problem instances with branching factors and operators of the order considered (see Fig. 6.2) in the TASKOGRAPHY benchmark. Moreover, several planners that learn to search [45, 191, 192, 193] depend on hard-to-obtain dense rewards or do not scale with domain complexity [194, 195, 196].

The simplification of planning problems via **pruning strategies** to enable efficient search has been explored in both propositional [197, 198, 185] and numeric [199] planning contexts. Among these, PLOI [36] is a particularly performant learning-based approach that leverages object-centric relational reasoning [200, 201, 202, 203] to score and prune *extraneous objects* to the task. While PLOI outperforms existing classical planners on the TASKOGRAPHY benchmark, it incurs a large number of replanning steps owing to inaccurate neural network predictions; and inability to exploit 3DSG hierarchies. Our proposed SEEK procedure decreases replanning steps by two orders of magnitude.

Planning benchmarks in the symbolic planning communities have featured a variety of tasks with time complexities ranging from polynomial (e.g., shortest-path) to NP-hard problems (e.g., traveling salesman). There also exists a handful of environments [204, 35, 205, 206, 207] for benchmarking learned action policies from language directives and egocentric visual observations, task and motion planning [208], or the modelling of physical interactions [209, 210]. Another recent benchmark [211] only supports navigation and block-stacking tasks. However, there isn't currently a large-scale benchmark tailored to robotic task planning in 3DSGs with several hundreds of objects.

6.3. Background

Task planning. A task planning problem Π is a tuple $\langle \mathcal{O}, \mathcal{P}, \mathcal{A}, \mathcal{T}, \mathcal{C}, \mathcal{I}, \mathcal{G} \rangle$. As a running example, consider the task find an apple, slice it, and place it on the counter. \mathcal{O} is the set of all ground objects (instances) in the problem. \mathcal{P} is a set of properties, each defined over one or more objects; weight(apple) = 70 grams. **Predicates** are subclasses of properties in that they are boolean-valued; canPlace(apple, refrigerator) = True. \mathcal{A} is a finite set of lifted actions operating over object tuples; slice(apple), place(apple, counter). \mathcal{T} is a transition model and \mathcal{C} denotes state transition costs. \mathcal{I} and \mathcal{G} are initial and goal states. A state is an assignment of values to all possible properties grounded over objects. For the running example, a goal state may be specified as on(apple, counter)=True and sliced(apple)=True. Planning problems may be grounded-slice *this* apple, or lifted-slice *an* apple.

3D scene graphs (3DSGs). A 3DSG [161, 162] is a hierarchical multigraph G = (V, E) with $k \in \{1 \cdots K\}$ levels, where $V^k \in V$ denotes the set of vertices at level k. Edges originating from a vertex $v \in V^k$ may only terminate in $V^{k-1} \cup V^k \cup V^{k+1}$ (i.e., edges connect nodes within one level of each other). Each 3DSG in our work comprises at least 5 levels with increasing spatial precision as we move down the hierarchy: the topmost level in the hierarchy is a root node representing a scene. This node branches out to indicate the various *floors* in the building, which in turn branches out to denote various *rooms* in a floor, and subsequently *places* within a room. A place is a collection of *objects*, which may themselves contain other *objects* (to allow for container types such as cabinets and refrigerators).¹ At each level, edges indicate various types of relations among nodes (e.g., at the room level, an edge indicates multiple affordance relations). Each node also stores semantic attributes such as node type, functionality, affordances, etc., following [161].

¹The lowest level in [162] is a metric-semantic mesh. However since our focus is on symbolic planning, we only require scene graph levels that contain *objects*.



Fig. 6.1. A *state* in a planning problem specified over a 3DSG. Nodes are scene entities and store unary predicates. Edges indicate binary predicates (relations). A goal is a conjunction of unary and binary literals. We only show a subset of relations for brevity. E.g., if the robot executes an action that moves it to another room, the robotInRoom relation shown in this figure will be set to False for the room on the lower left.

6.4. Taskography

We propose TASKOGRAPHY: the first large scale benchmark to evaluate symbolic planning over 3DSGs. Currently, TASKOGRAPHY comprises 20 challenging robotic task planning domains totaling 3734 tasks. Different from current benchmarks for embodied AI that focus primarily on egocentric *visual* reasoning [35, 212, 213, 204, 210, 206]; TASKOGRAPHY is designed to evaluate *symbolic* reasoning over 3DSGs. To emulate the complexity of real-world task planning problems, TASKOGRAPHY builds atop the Gibson [214] dataset comprising real-world scans of large building interiors (averaging 2-3 floors per building; 7 rooms per floor), and their corresponding 3DSGs [161].

Augmenting 3DSGs with plannable attributes. A prerequisite for planning over 3DSGs—absent in existing work [163, 161, 162]—is a database of *plannable attributes*: predicates, actions, and transition models. To support task planning, we augment each 3DSG in Gibson [214] (tiny and medium splits) with several layers of additional unary and binary predicates. For each 3DSG node, we obtain class labels, object dimensions and pose from [161]. We annotate object affordances by building a knowledge base of lifted object-action pairs and recursively applying it to every 3DSG node, while accounting for

exceptions (objects that are concealed or contained within others). We also detect *door* objects in the 3DSG and use this to add additional edges describing room connectivity. We annotate objects with all possible properties in our planning domains (e.g., "*is this object typically a receptacle?*"). Our rich property set (*plannable attributes*) is chosen to support a wide range of realistic-robotic tasks geared towards large (building-scale) 3DSGs.



Fig. 6.2. The Taskography benchmark comprises large-scale planning problems defined over buildings from the Gibson dataset [214]. (Left) Representative buildings from Gibson [214]. (Middle/Right) We feature a variety of problem classes ranging in scale and complexity as illustrated by the domain statistics.

Benchmark statistics. Each of the 20 TASKOGRAPHY domains specifies a class of planning problems that resemble real-world use cases (and theoretically complex extensions) that a robot would encounter in office, house, or building scale environments. These domains range from grounded planning domains to lifted planning domains, domains with no extraneous objects to domains where most objects are extraneous, and domains for which polynomial time solutions exist to NP-hard problems. The simplest domains in the benchmark have 1000 state variables and an average branching factor of 5; for hard domains, these are 4000 and 60 respectively (see Fig. 6.2).

Taskography-API. Our project page (https://taskography.github.io) will host code and data used in this work. In TASKOGRAPHY-API, an open-source python package, we provide access to 18 classical and learning-based symbolic planners, templates to implement novel domains, and methods to generate problem instances of varying complexities and train/evaluate learning-based planners.

Planners considered. TASKOGRAPHY supports a comprehensive set of planners to facilitate standardized evaluation on novel domains. The following planners are available at the time of writing.

- Optimal planners: Fast Downward (FD) with the opt-lmcut heuristic [26], Sat-Plan [29], Delfi [30], DecStar-optimal [169], and Monte Carlo tree search.
- Satisficing planners: Fast Forward (FF), FF with axioms (FF-X) [25], Fast Downward (FD) with the lama-cut heuristic [26], DecStar-satisficing [169], Cerberus [168], Best First Width Search (BFWS) [215], and regression planning.

• Learning-based planners: Relational policy learning [183], Planning with learned object importance (PLOI) [36] (and variants – see Sec. 6.6).

General assumptions. To facilitate evaluation of all of these classes of planners, the first edition of our benchmark only considers *fully observable* tasks and *discrete* state and action spaces. All goal states are specified as *conjunctions* of literals. While we make no distinction between deterministic or stochastic transitions, all current experiments assume a *closed world*, i.e., all possible lifted actions and effects are known apriori.

6.4.1. Robot planning domains: Case studies

The full TASKOGRAPHY benchmark comprises 20 domains. We discuss the four task categories from which all domains are constructed that we believe to be interesting to a broad robotics audience.

Domain 1. Rearrangement(k): Based on the recently proposed rearrangement challenge [216], this task requires a robot randomly spawned to rearrange a set of k objects of interest into k corresponding receptacles. The robot often needs to execute multiple other actions along the way, such as opening/closing doors, navigating to goals, planning the sequence of objects to visit, etc.

Domain 2. Courier (n, k): A robot that couriers objects is equipped with a knapsack of maximum payload capacity of n units. The robot needs to locate and courier k objects (of varying weights $w \in \{1, 2, 3\}$ units) to k distinct delivery points. The knapsack can be used to stow and retrieve items in random-access fashion; effectively embedding a combinatorial optimization problem into the task. Stow and retrieve actions increase branching, necessitating far deeper searches.

We also provide *lifted* variants of these tasks. Here, goals are specified over desired object-receptacle class relations (e.g., "put a cup on a table") as opposed to over object instances (e.g., "put this cup on the table"). These tasks introduce ambiguity in both the search of classical task-planners and learning-based techniques, which must now distinguish object instances of relevant classes.

Domain 3. Lifted Rearrangement (k): A lifted version of the rearrangement domain where the goals are specified at an object category level, as opposed to an instance level.

Domain 4. Lifted Courier (n, k): A lifted version of the courier domain where the goals are specified at an object category level, as opposed to an instance level.

To promote compatability with a range of planning systems [171, 217], we represent all tasks in PDDL format [218, 46]. We also include mechanisms for translating tasks into alternative problem definition languages that are essential for some of our supported planners [29]. **Table 6.1. Taskography** benchmark results on select grounded and lifted *Rearrangement* (**Rearr**) and *Courier* (**Cour**) 3DSG domains. Planning times are reported in seconds and do not incorporate planner-specific domain translation times (factored into planning timeouts). A '-' indicates planning timeouts or failures (10 minutes for optimal planners, 30 seconds for all others). Results are aggregated over 10 random seeds. Optimal task planning is infeasible in larger problem instances or for more complex domains, while most satisficing planners are unable to achieve real-time performance. PLOI [36], a recent learning-based planner consistently performs the best across all domains.

		Rea	urr(1) T	ìny	Rea	urr(2) T	iny	Rearr(10) Me	dium	Cour(7	, 10) M	edium	Lifted	Rearr(5)) Tiny	Lifted	Cour(5,	5) Tiny
	Planner	Len.	Time	Fail	Len.	Time	Fail	Len.	Time	Fail	Len.	Time	Fail	Len.	Time	Fail	Len.	Time	Fail
	FD-seq-opt-lmcut	15.77	24.81	0.04	25.80	104.47	0.55	-	-	1.00	-	-	1.00	-	-	1.00	-	-	1.00
ыl	SatPlan	14.77	10.35	0.45	26.67	3.27	0.67	-	-	1.00	-	-	1.00	-	-	1.00	-	-	1.00
Ë.	Delfi	15.13	0.36	0.16	29.10	27.77	0.29	-	-	1.00	-	-	1.00	-	-	1.00	-	-	1.00
ob	DecStar-opt-fb	-	-	1.00	-	-	1.00	-	-	1.00	-	-	1.00	-	-	1.00	-	-	1.00
-	MCTS	-	-	1.00	-	-	1.00	-	-	1.00	-	-	1.00	-	-	1.00	-	-	1.00
	FF	16.71	<u>0.19</u>	0.00	34.44	0.55	0.00	159.04	5.30	0.09	128.41	6.62	0.24	62.86	3.40	0.47	57.74	4.03	0.44
50	FF-X	16.71	0.25	0.00	34.44	0.58	0.00	159.80	5.02	<u>0.08</u>	<u>128.19</u>	6.72	0.24	67.88	3.48	0.89	61.19	7.56	0.77
ici.	FD-lama-first	15.19	2.96	0.33	38.47	3.25	0.18	208.28	6.35	0.49	156.34	4.92	0.29	66.81	3.20	0.49	61.13	3.34	0.56
isf	Cerberus-sat	11.50	12.00	0.85	-	-	1.00	-	-	1.00	-	-	1.00	-	-	1.00	-	-	1.00
sat	Cerberus-agl	14.77	5.13	0.45	33.00	7.30	0.49	176.60	8.91	0.72	125.73	12.99	0.83	60.50	7.62	0.60	59.19	7.05	0.77
	DecStar-agl-fb	14.72	2.62	0.55	34.96	2.58	0.58	211.16	7.20	0.82	132.60	4.50	0.58	66.30	3.02	0.71	58.75	4.46	0.71
	BFWS	15.56	0.90	0.22	32.16	0.37	0.18	151.17	0.41	0.23	152.71	<u>1.13</u>	0.21	56.90	0.94	0.41	61.92	2.30	0.43
	Regression-plan	-	-	1.00	-	-	1.00	-	-	1.00	-	-	1.00	-	-	1.00	-	-	1.00
Н	Relational policy [183]	-	-	1.00	-	-	1.00	-	-	1.00	-	-	1.00	-	-	1.00	-	-	1.00
lea	PLOI [36]	16.45	0.00^{*}	0.00	37.04	0.00^{*}	0.00	213.43	0.17	0.00	161.90	0.34	0.00	78.68	0.22	0.24	71.71	0.26	0.26

6.4.2. Benchmarking classical and learned planners on Taskography

We present the empirical results on the TASKOGRAPHY benchmark across several classes of task-planners in Table. 6.1. (Please consult supplementary material for a number of additional results).

Evaluation protocol. We treat the evaluation of optimal planners separately to the remaining methods. Optimal planners are not intended to be fast unlike satisficing and learning-based variants. Rather, they compute a solution of minimum length (not necessarily unique) to a given problem. Optimal planners are hence allotted 10 minutes to solve each problem, while satisficing and learning-based planners are allotted 30 seconds. For learning-based methods, we evaluate results over 10 random seeds for statistical significance. We report standard deviations in the supplementary material. All domains comprise 40 training problems. The domains tagged *Tiny* and *Medium* comprise 55 and 182 test problems respectively, unless otherwise specified.

Optimal planners work only on the simplest of domains. Despite the reasonable performance of optimal planners on the Rearrangment(1) domain, they are unable to efficiently scale with increasing task complexity and fail to solve a single task on the Rearrangment (k) and Courier (n, k) domains for k > 2. In particular, the Rearrangement(1) domain is a superset of the grounded hierarchical path planning (HPP) task as described by [160]. Because the HPP task does not consider state changes to the scene graph (i.e., directly equating the 3DSG to the planning graph for search), efficient shortest path planning is

	Rear	r(10) T	iny	Rearr(10) Medium				
Planner	Len.	Time	Fail	Len.	Time	Fail		
FF	162.61	7.04	0.07	<u>159.04</u>	<u>5.30</u>	0.09		
FD (satisficing)	205.89	7.68	0.51	208.28	6.35	0.49		
DecStar-agl-fb	193.00	6.78	0.85	211.16	7.20	0.82		
BFWS	160.93	0.57	<u>0.18</u>	151.17	0.41	<u>0.23</u>		

Table 6.2. Interestingly, task complexity does not correlate strongly with scene complexity. It is instead determined by the number of operators, and avg. branch factor.

tractable. However, increasingly complex robot tasks requires more than the mere ability to path plan in 3DSGs.

Planning performance degrades with domain complexity, not scene complexity. We observe an increase in the number of planning failures and timeouts as satisficing planners are applied to larger Rearrangement(k) domains (Table 6.2). Interestingly, larger scenes do not appear to directly correlate with task complexity, as the performance metrics remain largely consistent between the tiny and medium splits of the same domain (Table. 6.2).

Satisficing planners fail in domains requiring long-horizon reasoning. In the Courier(n, k) domains, satisficing planners tend to produce shorter length solutions by leveraging the knapsack's capacity to stow objects on the way to various delivery points. However, the planners often display shortsighted behaviours by stowing objects early in the search, depleting knapsack slots that could potentially help further along the task. This yields dead-end configurations and excessive backtracking, and thus, an increase in timeouts is observed.

Planners that do not exploit forward heuristics fail due to large branching factors. Due to the large branching factor of our domains, common strategies such as Monte-Carlo Tree Search (MCTS) and MC Regression Planning are unable to solve any task within a reasonable time constraint. For instance, a Rearrangement(10) task has an average branching factor of 6.5 for MCTS. Since a reward is only obtained at the end (typical planners take 200 steps to get there), MCTS degenerates to a slow breadth-first search.

Learning based planners that prune the state space excel on all domains. We also evaluate two prevalent learning-to-plan methods based on generalized relational planning [183] and planning with learned abstractions [36]. While the relational policy stuggles to generalize in our domains (long-horizon, sparse rewards), PLOI demonstrates an impressive ability to detect and prune contextually irrelvant parts of the 3DSGs. However, it also requires a significant number of replanning steps (see figure to the right) as it often retains objects within a graph without ensuring that all properties and ancestors required to access the object are also preserved.



Fig. 6.3. Learning-based planners like PLOI outperform all other planners on the benchmark, but still incur significant overhead (number of replanning steps).

Discussion. Our evaluation of existing performant planners on the TASKOGRAPHY benchmark consistently reveals two important trends across all domains.

- Pruning a 3DSG is essential for real-time performance, more so on challenging domains.
- While learning-based planners excel across all domains, they require a large number of replanning steps.

These imply that efficient utilization of 3DSGs in real-time robotic task planning requires *both* adapting 3DSGs to better suit existing planners, and enabling performant (learning-based) planners to better exploit 3DSG hierarchies. The remainder of our work addresses these issues.

6.5. SCRUB: Principled sparsification of 3DSGs for efficient planning

As discussed above, learning-based planners leverage a wealth of prior knowledge acquired during a training phase to significantly prune extraneous scene graph entities. We argue that, if equipped with the right sparsification machinery, classical planners can compete with, or outperform learning methods. We develop SCRUB, a principled 3DSG sparsification scheme that prunes a 3DSG G (w.r.t. planning problem $\Pi_G = \langle \mathcal{O}, \mathcal{P}, \mathcal{A}, \mathcal{T}, \mathcal{C}, \mathcal{I}, \mathcal{G} \rangle$) by removing vertices and edges extraneous to the task, resulting in a sparsified 3DSG \hat{G} (and planning problem $\hat{\Pi}_{\hat{G}} = \langle \hat{\mathcal{O}}, \hat{\mathcal{P}}, \hat{\mathcal{A}}, \hat{\mathcal{T}}, \hat{\mathcal{C}}, \hat{\mathcal{I}}, \mathcal{G} \rangle$)

Algorithm 1 SCRUB algorithm

1:	function SCRUB(3DSG G, Planning problem $\Pi = \langle \mathcal{O}, \mathcal{P}, \mathcal{A}, \mathcal{T}, \mathcal{C}, \mathcal{I}, \mathcal{G} \rangle$)
2:	$\hat{\mathcal{O}} = \{\}$ \triangleright Init. sufficient object set
3:	$g = \text{OBJECTS}(\mathcal{G}.\text{literals}) \cup \{robot\}$ \triangleright Init. set of objects in the goal literal set
4:	while not empty g do
5:	$\hat{\mathcal{O}} \leftarrow \hat{\mathcal{O}} \cup g$ \triangleright Add to $\hat{\mathcal{O}}$ all objects newly discovered in g
6:	$p \leftarrow \text{all binary predicates relating a newly added object (i.e. o \in g - \hat{\mathcal{O}}) to its$
	ancestors in G
7:	$g \leftarrow \text{Objects}(p)$
8:	if all objects \mathcal{O} visited then
9:	Break
10:	end if
11:	end while
12:	$\hat{G} \leftarrow G$ \triangleright Initialize sparsified scenegraph
13:	CONNECTROOMS \triangleright All-pairs shortest paths
14:	Remove all nodes from \hat{G} that are not in $\hat{\mathcal{O}}$
15:	Prune literals that are no longer valid in the sparsified graph return Sparsified 3DSG
	\hat{G}
16:	end function

Definition 1. A valid 3DSG sparsification of G for a planning problem Π_G to \hat{G} (and corresponding planning problem $\hat{\Pi}_{\hat{G}}$) is a computable function SCRUB(Π_G) = $\hat{\Pi}_{\hat{G}}$ such that, a plan p solves Π_g iff it solves $\hat{\Pi}_{\hat{G}}$.

A satisficing plan for Π_G may thus be obtained by simply solving the (much easier to solve) sparsified problem $\hat{\Pi}_{\hat{G}}$. Savings in planning time depend on the complexity of the sparsified subgraph \hat{G} . SCRUB presents a simple strategy which is guaranteed to be minimal for grounded planning problems and satisficing for lifted planning problems.

For exposition, we consider grounded planning problems; see appendix for how SCRUB is adapted to lifted planning problems or stochastic transitions. SCRUB begins with an initially empty sufficient object set $\hat{\mathcal{O}}$. Satisfying the goal minimally requires all ground objects in the goal to be included in the sufficient object set $\hat{\mathcal{O}}$ (else goal objects are unreachable). In addition, the robot itself must be part of the sufficient set. Let p be the set of all binary predicates which include any of these objects. And let g be the set of all objects contained in p. In general, this will be a superset of the objects we started with. We iteratively repeat this process, each time adding the new objects in g to our sufficient set $\hat{\mathcal{O}}$.

The process terminates either when either the set g has no new objects (indicating convergence), or until all the objects in the scene graph are visited at least once (indicating the input graph already defines a minimal object set). We initialize the nodes of \hat{G} with objects in $\hat{\mathcal{O}}$, and copy over all edges $(u, v) \in G$ for which both $u, v \in \hat{\mathcal{O}}$. SCRUB terminates in time linear in the number of the predicates or nodes (whichever is larger).



Fig. 6.4. Best performing planners with and without SCRUB.



Fig. 6.5. SCRUB greatly prunes operators and states of planning problems.

Proposition 1. SCRUB is complete and results in a minimal scene subgraph for all grounded planning problems over the scenegraph domain. (Please refer to supp. material for proof)

6.5.1. Impact of SCRUB on modern task planners

In this section, we investigate the effect that a 3DSG reduction scheme like SCRUB may have on the performance of modern task planners. We experiment with the four domains shown in Fig. 6.4 and evaluate the impact of scrub on planning performance and on domain structure.

SCRUB enables classical planners to obtain performance at least as good as state-of-the-art planners. In Fig. 6.4, we see that SCRUB drastically reduces planning time for FF, FD-lama-first, and BFWS to a few milliseconds on Rearrangement(10), and upper-bounds times at 5 seconds on Courier(10, 10). We see this enables BFWS, FD, and FF to outperform PLOI (lower plan lengths for similar plan times). The grounded domains each have 182 test problems, and the lifted domains each have 70 test problems.

SCRUB greatly reduces the number of operators and states. To asses the impact of SCRUB, we compute statistics (number of operators, number of state variables) in Fig. 6.5.

Planner	% Success	Length	Time
FD (satisficing)	51.43	66.81	3.20
FD (satisficing) + scrub	72.86	73.09	1.61
FD (optimal)	-	-	-
FD (optimal) + scrub	72.86	68.33	2.26

Table 6.3. Planner statistics evaluated over 70 test problems on *Lifted Rearrangement*(5).

We see that SCRUB prunes *more than two-thirds* of the operators and state variables for grounded planning problems, and about a third in the case of lifted planning problems.

SCRUB enables optimal planners to run on lifted domains. Table 6.3 reports results of running the satisficing and optimal variants of FD with and without SCRUB, on the *Lifted Rearrangement(5)* domain. While FD (optimal) did not converge even with a timeout of 24 hours, FD (optimal) + SCRUB solved about 72% of the tasks under a 30-second timeout, taking 2 seconds per task on average.

6.6. SEEK: A procedure for efficient learning-based planning

While SCRUB results in a 3DSG reduction that is guaranteed to find a satisficing plan if one exists—its conservative approach hurts performance in challenging lifted planning problems as shown in Fig. 6.4. For such problems, learning-based graph-pruning strategies like PLOI [36] outperform classical planners over SCRUBBED 3DSGs. However, as can be seen in Sec. 6.4.2, even PLOI [36] incurs a significant number of replanning iterations.

We posit that several replanning iterations may be avoided by exploiting the 3DSG hierarchy. Pruning strategies like PLOI first score all objects, and retain a minimal set by thresholding. A simple threshold does little to ensure that all retained objects are reachable from the root of the scene graph. To alleviate this issue, we propose SEEK: a procedure that ensures we obtain a connected graph, with the objective of reducing the number of replanning steps needed.

SEEK requires as input the 3DSG, the planning problem Π , and an object scoring mechanism f_{θ} . This scoring mechanism is typically a graph neural network (akin to [36]) that, given the current state, scores each object with an *importance* value in [0, 1]. We first run the scorer and only retain objects above a threshold score t. We follow an identical approach to PLOI [36] and at each step geometrically decay the threshold by γ , such that at iteration i, the threshold is $t_i = \gamma t_{i-1}$, with $t_0, \gamma \in [0, 1)$. For each retained object o, we recursively traverse up the 3DSG, adding all ancestors of o to the sufficient object set. This procedure ensures that all objects are reachable from their respective room nodes. While

Table 6.4. SEEK significantly reduces the number of replanning steps required by state-ofthe-art learning-based planners. For each planner, we report average *wall time* (including translation time).

Planner	Rearrangement (2) - Medium				Courier (10, 3) - Medium				Lift	Lifted Rearrangement (5) - Medium				I	Lifted Courier (5, 5) - Medium					
1 laintei	%Succ.	Len.	%Used	Time	#Replan	%Succ.	Len.	%Used	Time	#Replan	%Succ.	Len.	%Used	Time	#Replan	%Succ.	Len.	%Used	Time	#Replan
Random	0.87	39.81	0.99	9.51	836	0.62	180	0.10	12.11	204	0.63	68.98	0.99	10.93	235	0.67	67.89	0.98	10.81	233
Random + seek	0.86	39.82	0.98	8.55	543	0.60	183.49	0.99	12.33	162	0.59	69.22	0.97	9.52	155	0.63	65.48	0.97	10.97	167
Hierarchical	1	35.76	0.28	0.45	150	1	191.75	0.48	1.16	40	0.80	76.75	0.59	2.60	269	0.73	69.69	0.61	2.73	173
Hierarchical + seek	1	35.76	0.28	0.30	12	1	191.75	0.48	0.97	7	0.80	76.70	0.56	2.20	208	0.77	76.04	0.55	1.59	76
PLOI [36]	1	35.76	0.28	0.44	141	1	191.75	0.48	1.13	41	0.79	78.16	0.59	2.49	258	0.73	69.88	0.62	2.75	169
PLOI + seek	1	35.76	0.28	0.31	14	1	191.75	0.48	0.97	7	0.80	76.61	0.56	2.18	197	0.77	79.19	0.55	1.53	53

SEEK, unlike SCRUB, is not guaranteed to be satisficing, it results in far fewer replanning steps without affecting computation time.

SEEK reduces replanning steps by an order of magnitude. To assess the impact of the SEEK procedure on planning performance, we evaluate performance with respect to other learning-based planners on TASKOGRAPHY in Table 6.4. As a baseline, we evaluate a *random* pruning strategy that uniformly randomly retains or prunes every object. Even for this naive strategy, SEEK offers significant performance improvement. We also evaluate *PLOI* [36] and our adaptation dubbed *hierarchical*, which trains multiple graph neural networks, one for each level of the 3DSG hierarchy. For each variant, SEEK offers a consistent performance improvement by decreasing the number of replanning steps required as seen in Fig. 6.6. SEEK is thus a conceptually simple strategy for use with learning-based planners.

Lifted Courier (5, 5) Rearrangement (2) Hierarchical PLOI + Seek 0 50 100 150 Num Replanning Steps

Fig. 6.6. SEEK reduces replanning steps by an order of magnitude.



Fig. 6.7. SCRUB on grounded domains, SEEK on lifted domains.

SEEK on lifted domains: In general, we note that SCRUB is more performant on grounded domains (due to minimality properties) and SEEK is more performant on lifted domains (where SCRUB typically retains all instances of important object categories, but SEEK is more effective due to its opportunistic retention of instances (Fig. 6.7)).

domains.

6.7. Concluding remarks

SCRUB on

grounded

Limitations. TASKOGRAPHY currently supports only a fraction of the diverse types of planning problems possible on 3DSGs. Geared towards identifying the most promising avenues in learning-based planning, the first release of this benchmark focuses exclusively on offline task planning in fully observable and deterministic domains. Furthermore, low-level motion planning is excluded from our benchmark. Robots operating in the real world will need to reason under partial observability, sensor noise, and resource constraints.

Outlook. TASKOGRAPHY, in conjunction with SCRUB and SEEK aid the robot learning community by (a) providing guidelines and implementations for practitioners choosing a task planner, (b) serving as a benchmark for upcoming learning-based planners, and (c) guiding the design of futuristic spatial representations for robotic task planning. We believe TASKOGRAPHY is a first step towards addressing several of the grand challenges along the road to developing general planning capabilities for autonomous intelligent robots.

Chapter 7

Prologue to Article 3

This article was published and presented at the International Conference on Learning Representations (ICLR), 2021.

Authors: Krishna Murthy Jatavallabhula^{*}, Miles Macklin^{*}, Florian Golemo, Vikram Voleti, Linda Petrini, Martin Weiss, Breandan Considine, Jerome Parent-Levesque, Kevin (Robert) Xie, Kenny Erleben, Liam Paull, Florian Shkurti, Derek Nowrouzezahrai, Sanja Fidler (First two authors contributed equally)

Contribution statement: The idea was conceived after a joint brain storming session involving the author, Martin Weiss, Breandan Considine, Derek Nowrouzezahrai, and Bhairav Mehta. The author led the effort of building prototype differentiable physics and graphics engines. Martin Weiss built early prototypes of a billiards simulator. At around the same time, Miles Macklin independently built an efficient differentiable physics engine. Sensing a strong overlap in research interests, the author and Miles Macklin began collaborating, with Miles Macklin focusing on the differentiable physics engine, and the author focusing on the differentiable graphics engine and the overall systems integration. Florian Golemo and Vikram Voleti were instrumental in setting up the datasets used in our experiments, and in preparing the visualizations and plots for the manuscript. Linda Petrini implemented the neural network baselines for system identification. All authors participated in weekly discussions and helped with preparing the manuscript.

Note: This paper is presented as-is, with minor cosmetic changes to adhere to the Universite de Montreal thesis template.

Chapter 8

Article 3: gradSim: Differentiable simulation for system identification and visuomotor control



Fig. 8.1. ∇ *Sim* is a unified differentiable rendering and multiphysics framework that allows solving a range of control and parameter estimation tasks (rigid bodies, deformable solids, and cloth) directly from images/video.

Abstract

We consider the problem of estimating an object's physical properties such as mass, friction, and elasticity directly from video sequences. Such a system identification problem is fundamentally ill-posed due to the loss of information during image formation. Current solutions require precise 3D labels which are labor-intensive to gather, and infeasible to create for many systems such as deformable solids or cloth. We present ∇Sim , a framework that overcomes the dependence on 3D supervision by leveraging differentiable multiphysics simulation and differentiable rendering to jointly model the evolution of scene dynamics and image formation. This novel combination enables backpropagation from pixels in a video sequence through to the underlying physical attributes that generated them. Moreover, our unified computation graph – spanning from the dynamics and through the rendering process – enables learning in challenging visuomotor control tasks, without relying on state-based (3D) supervision, while obtaining performance competitive to or better than techniques that rely on precise 3D labels.

8.1. Introduction

Accurately predicting the dynamics and physical characteristics of objects from image sequences is a long-standing challenge in computer vision. This end-to-end reasoning task requires a fundamental understanding of *both* the underlying scene dynamics and the imaging process. Imagine watching a short video of a basketball bouncing off the ground and ask: "Can we infer the mass and elasticity of the ball, predict its trajectory, and make informed decisions, e.g., how to pass and shoot?" These seemingly simple questions are extremely challenging to answer even for modern computer vision models. The underlying physical attributes of objects and the system dynamics need to be modeled and estimated, all while accounting for the loss of information during 3D to 2D image formation.

Depending on the assumptions on the scene structre and dynamics, three types of solutions exist: *black, grey,* or *white box. Black box* methods [219, 220, 221, 222] model the state of a dynamical system (such as the basketball's trajectory in time) as a learned embedding of its states or observations. These methods require few prior assumptions about the system itself, but lack interpretability due to entangled variational factors [223] or due to the ambiguities in unsupervised learning [224, 225]. Recently, grey box methods [226] leveraged partial knowledge about the system dynamics to improve performance. In contrast, *white box* methods [227, 228, 64, 229] impose prior knowledge by employing explicit dynamics models, reducing the space of learnable parameters and improving system interpretability.

Most notably in our context, all of these approaches require precise 3D labels – which are labor-intensive to gather, and infeasible to generate for many systems such as deformable solids or cloth.

We eliminate the dependence of white box dynamics methods on 3D supervision by coupling explicit (and differentiable) models of scene dynamics with image formation (rendering)¹.

Explicitly modeling the end-to-end dynamics and image formation underlying video observations is challenging, even with access to the full system state. This problem has been treated in the vision, graphics, and physics communities [230, 231], leading to the development of robust forward simulation models and algorithms. These simulators are not readily usable for solving *inverse* problems, due in part to their non-differentiability. As such, applications of black-box *forward* processes often require surrogate gradient estimators such as finite differences or REINFORCE [232] to enable any learning. Likelihood-free inference

 $^{^{1}}Dynamics$ refers to the laws governing the motion and deformation of objects over time. *Rendering* refers to the interaction of these scene elements – including their material properties – with scene lighting to form image sequences as observed by a virtual camera. *Simulation* refers to a unified treatment of these two processes.



Fig. 8.2. ∇ Sim: Given video observations of an evolving physical system (e), we randomly initialize scene object properties (a) and evolve them over time using a differentiable physics engine (b), which generates states. Our renderer (c) processes states, object vertices and global rendering parameters to produce image frames for computing our loss. We backprop through this computation graph to estimate physical attributes and controls. Existing methods rely solely on differentiable physics engines and require supervision in state-space (f), while ∇ Sim only needs image-space supervision (g).

for black-box forward simulators [233, 234, 235, 236, 237, 238, 239] has led to some improvements here, but remains limited in terms of data efficiency and scalability to high dimensional parameter spaces. Recent progress in *differentiable simulation* further improves the learning dynamics, however we still lack a method for end-to-end differentiation through the entire simulation process (i.e., from video pixels to physical attributes), a prerequisite for effective learning from video frames alone.

We present ∇Sim , a versatile end-to-end differentiable simulator that adopts a holistic, unified view of differentiable dynamics and image formation(*cf.* Fig. 8.1,8.2). Existing differentiable physics engines only model time-varying dynamics and require supervision in *state space* (usually 3D tracking). We additionally model a differentiable image formation process, thus only requiring target information specified in *image space*. This enables us to backpropagate [240] training signals from video pixels all the way to the underlying physical and dynamical attributes of a scene.

Our main contributions are:

- ∇Sim , a differentiable simulator that demonstrates the ability to backprop from video pixels to the underlying physical attributes (*cf.* Fig. 8.2).
- We demonstrate recovering many physical properties exclusively from video observations, including friction, elasticity, deformable material parameters, and visuomotor controls (sans 3D supervision)
- A PyTorch framework facilitating interoperability with existing machine learning modules.

We evaluate ∇Sim 's effectiveness on parameter identification tasks for rigid, deformable and thin-shell bodies, and demonstrate performance that is competitive, or in some cases superior, to current physics-only differentiable simulators. Additionally, we demonstrate the effectiveness of the gradients provided by ∇Sim on challenging visuomotor control tasks involving deformable solids and cloth.

8.2. ∇Sim : A unified differentiable simulation engine

Typically, physics estimation and rendering have been treated as disjoint, mutually exclusive tasks. In this work, we take on a unified view of *simulation* in general, to compose physics estimation and rendering. Formally, simulation is a function $\operatorname{Sim} : \mathbb{R}^P \times [0, 1] \mapsto \mathbb{R}^H \times \mathbb{R}^W; \operatorname{Sim}(\mathbf{p}, t) = \mathcal{I}$. Here $\mathbf{p} \in \mathbb{R}^P$ is a vector representing the simulation state and parameters (objects, their physical properties, their geometries, etc.), t denotes the time of simulation (conveniently reparameterized to be in the interval [0, 1]). Given initial conditions \mathbf{p}_0 , the simulation function produces an image \mathcal{I} of height H and width W at each timestep t. If this function Sim were differentiable, then the gradient of $\operatorname{Sim}(\mathbf{p}, t)$ with respect to the simulation parameters \mathbf{p} provides the change in the output of the simulation from \mathcal{I} to $\mathcal{I} + \nabla \operatorname{Sim}(\mathbf{p}, t) \delta \mathbf{p}$ due to an *infinitesimal perturbation of* \mathbf{p} by $\delta \mathbf{p}$. This construct enables a gradient-based optimizer to estimate physical parameters from video , by defining a *loss* function over the image space $\mathcal{L}(\mathcal{I}, .)$, and descending this loss landscape along a direction parallel to $-\nabla \operatorname{Sim}(.)$. To realise this, we turn to the paradigms of *computational graphs* and *differentiable programming*.

 ∇ Sim comprises two main components: a differentiable physics engine that computes the physical states of the scene at each time instant, and a differentiable renderer that renders the scene to a 2D image. Contrary to existing differentiable physics [241, 242, 243, 244, 227, 245, 246, 64, 229] or differentiable rendering [109, 110, 111, 112] approaches, we adopt a holistic view and construct a computational graph spanning them both.

8.2.1. Differentiable physics engine

Under Lagrangian mechanics, the state of a physical system can be described in terms of generalized coordinates \mathbf{q} , generalized velocities $\dot{\mathbf{q}} = \mathbf{u}$, and design/model parameters θ . For the purpose of exposition, we make no distinction between rigid bodies, or deformable solids, or thin-shell models of cloth, etc. Although the specific choices of coordinates and parameters vary, the simulation procedure is virtually unchanged. We denote the combined state vector by $\mathbf{s}(t) = [\mathbf{q}(t), \mathbf{u}(t)]$.

The dynamic evolution of the system is governed by second order differential equations (ODEs) of the form $\mathbf{M}(\mathbf{s}, \theta) \dot{\mathbf{s}} = \mathbf{f}(\mathbf{s}, \theta)$, where \mathbf{M} is a mass matrix that depends on the state and parameters. The forces on the system may be parameterized by design parameters (e.g. Young's modulus). Solutions to these ODEs may be obtained through black box numerical integration methods, and their derivatives calculated through the continuous adjoint method [247]. However, we instead consider our physics engine as a differentiable operation that provides an implicit relationship between a state vector $\mathbf{s}^- = \mathbf{s}(t)$ at the start of a time step, and the updated state at the end of the time step $\mathbf{s}^+ = \mathbf{s}(t + \Delta t)$. An arbitrary

discrete time integration scheme can be then be abstracted as the function $\mathbf{g}(\mathbf{s}^-, \mathbf{s}^+, \theta) = \mathbf{0}$, relating the initial and final system state and the model parameters θ .

Gradients through this dynamical system can be computed by graph-based autodiff frameworks [105, 106, 107], or by program transformation approaches [64, 248]. Our framework is agnostic to the specifics of the differentiable physics engine, however in Appendices B.1 through B.4 we detail an efficient approach based on the source-code transformation of parallel kernels, similar to DiffTaichi [64]. In addition, we describe extensions to this framework to support mesh-based tetrahedral finite-element models (FEMs) for deformable and thin-shell solids. This is important since we require surface meshes to perform differentiable rasterization as described in the following section.

8.2.2. Differentiable rendering engine

A renderer expects *scene description* inputs and generates color image outputs, all according to a sequence of image formation stages defined by the *forward* graphics pipeline. The scene description includes a complete *geometric* descriptor of scene elements, their associated material/reflectance properties, light source definitions, and virtual camera parameters. The rendering process is not generally differentiable, as *visibility* and *occlusion* events introduce discontinuities. Most interactive renderers, such as those used in real-time applications, employ a *rasterization* process to project 3D geometric primitives onto 2D pixel coordinates, resolving these visibility events with non-differentiable operations.

Our experiments employ two differentiable alternatives to traditional rasterization, SoftRas [111] and DIB-R [112], both of which replace discontinuous triangle mesh edges with smooth sigmoids. This has the effect of blurring triangle edges into semi-transparent boundaries, thereby removing the non-differentiable discontinuity of traditional rasterization. DIB-R distinguishes between *foreground pixels* (associated to the principal object being rendered in the scene) and *background pixels* (for all other objects, if any). The latter are rendered using the same technique as SoftRas while the former are rendered by bilinearly sampling a texture using differentiable UV coordinates.

 ∇Sim performs differentiable physics simulation and rendering at independent and adjustable rates, allowing us to trade computation for accuracy by rendering fewer frames than dynamics updates.

8.3. Experiments

We conducted multiple experiments to test the efficacy of ∇Sim on physical parameter identification from video and visuomotor control, to address the following questions:

• Can we accurately identify physical parameters by backpropagating from video pixels, through the simulator? (Ans: *Yes, very accurately, cf.* Sec. 8.3.1)



Fig. 8.3. Parameter Estimation: For deformable experiments, we optimize the material properties of a beam to match a video of a beam hanging under gravity. In the rigid experiments, we estimate contact parameters (elasticity/friction) and object density to match a video (GT). We visualize entire time sequences (t) with color-coded blends.

- What is the performance gap associated with using ∇Sim (2D supervision) vs. differentiable physics-only engines (3D supervision)? (Ans: ∇Sim is competitive/superior, cf. Tables 8.1, 8.2, 8.3)
- How do loss landscapes differ across differentiable simulators (∇Sim) and their nondifferentiable counterparts? (Ans: Loss landscapes for ∇Sim are smooth, cf. Sec. 8.3.1.3)
- Can we use ∇Sim for visuomotor control tasks? (Ans: Yes, without any 3D supervision, cf. Sec. 8.3.2)
- How sensitive is ∇Sim to modeling assumptions at system level? (Ans: Moderately, cf. Table 8.4)

Each of our experiments comprises an *environment* \mathcal{E} that applies a particular set of physical forces and/or constraints, a (differentiable) loss function \mathcal{L} that implicitly specifies an objective, and an *initial guess* θ_0 of the physical state of the simulation. The goal is to recover optimal physics parameters θ^* that minimize \mathcal{L} , by backpropagating through the simulator.

8.3.1. Physical parameter estimation from video

First, we assess the capabilities of ∇Sim to accurately identify a variety of physical attributes such as mass, friction, and elasticity from image/video observations. To the best of our knowledge, ∇Sim is the first study to **jointly** infer such fine-grained parameters from video observations. We also implement a set of competitive baselines that use strictly more information on the task.

8.3.1.1. Rigid bodies (rigid). Our first environment-rigid-evaluates the accuracy of estimating of physical and material attributes of rigid objects from videos. We curate a dataset of 10000 simulated videos generated from variations of 14 objects, comprising primitive shapes such as boxes, cones, cylinders, as well as non-convex shapes from ShapeNet [250] and DexNet [251]. With uniformly sampled initial dimensions, poses, velocities, and physical properties (density, elasticity, and friction parameters), we apply a *known* impulse to the object and record a video of the resultant trajectory. Inference with ∇Sim is done by guessing an initial

Approach	Mean	Abs.
	abs. err.	Rel.
	(kg)	err.
Average	0.2022	0.1031
Random	0.2653	0.1344
ConvLSTM [220]	0.1347	0.0094
$PyBullet + REINFORCE \ [249]$	0.0928	0.3668
DiffPhysics (3D sup.)	1.35e-9	5.17e-9
∇Sim	2.36e-5	9.01e-5

Table 8.1. Mass estimation: ∇Sim obtains *precise* mass estimates, comparing favourably even with approaches that require 3D supervision (*diffphysics*). We report the mean abolute error and absolute relative errors for all approaches evaluated.

mass (uniformly random in the range $[2, 12]kg/m^3$), unrolling a *differentiable* simulation using this guess, comparing the rendered out video with the true video (pixelwise mean-squared error - MSE), and performing gradient descent updates. We refer the interested reader to the appendix (Sec. B.7) for more details.

Table 8.1 shows the results for predicting the mass of an object from video, with a known impulse applied to it. We use EfficientNet (B0) [252] and resize input frames to 64×64 . Feature maps at a resoluition of $4 \times 4 \times 32$ are concatenated for all frames and fed to an MLP with 4 linear layers, and trained with an MSE loss. We compare ∇Sim with three other baselines: PyBullet + RE-INFORCE [249, 253], diff. physics only (requiring 3D supervision), and a ConvL-STM baseline adopted from [220] but with a stronger backbone. The *DiffPhysics* baseline

	mass	elast	icity	friction		
Approach	m	k_d	k_e	k_f	μ	
Average	1.7713	3.7145	2.3410	4.1157	0.4463	
Random	10.0007	4.18	2.5454	5.0241	0.5558	
ConvLSTM [220]	0.029	0.14	0.14	0.17	0.096	
DiffPhysics (3D sup.)	1.70e-8	0.036	0.0020	0.0007	0.0107	
∇Sim	2.87e-4	0.4	0.0026	0.0017	0.0073	

Table 8.2. Rigid-body parameter estimation: ∇Sim estimates contact parameters (elasticity, friction) to a high degree of accuracy, despite estimating them from video. Diffphys. requires accurate 3D ground-truth at 30 FPS. We report absolute *relative* errors for each approach evaluated.

is a strict subset of ∇Sim , it only inolves the differentiable physics engine. However, it needs precise 3D states as supervision, which is the primary factor for its superior performance. Nevertheless, ∇Sim is able to very precisely estimate mass from video, to a absolute relative error of 9.01e-5, nearly two orders of magnitude better than the ConvLSTM baseline. Two other baselines are also used: the "Average" baseline always predicts the dataset mean and

	Deform	Deformable solid FEM									
	Per-particle mass	particle mass Material properties									
	m	μ	λ	v							
Approach	Rel. MAE	Rel. MAE	Rel. MAE	Rel. MAE							
DiffPhysics (3D Sup.)	0.032	0.0025	0.0024	0.127							
∇Sim	0.048	0.0054	0.0056	0.026							

Table 8.3. Parameter estimation of deformable objects: We estimate per-particle masses and material properties (for solid def. objects) and per-particle velocities for cloth. In the case of cloth, there is a perceivable performance drop in *diffphysics*, as the center of mass of a cloth is often outside the body, which results in ambiguity.

the "*Random*" baseline predicts a random parameter value from the test distribution. All baselines and training details can be found in Sec. B.8 of the appendix.

To investigate whether analytical differentiability is required, our PyBullet + REIN-FORCE baseline applies black-box gradient estimation [232] through a non-differentiable simulator [254], similar to [249]. We find this baseline particularly sensitive to several simulation parameters, and thus worse-performing. In Table 8.2, we jointly estimate friction and elasticity parameters of our compliant contact model from video observations alone. The trend is similar to Table 8.1, and ∇Sim is able to precisely recover the parameters of the simulation. A few examples can be seen in Fig. 8.3.

8.3.1.2. Deformable Bodies (deformable). We conduct a series of experiments to investigate the ability of ∇Sim to recover physical parameters of deformable solids and thin-shell solids (cloth). Our physical model is parameterized by the per-particle mass, and Lamé elasticity parameters, as described in in Appendix B.3.1. Fig. 8.3 illustrates the recovery of the elasticity parameters of a beam hanging under gravity by matching the deformation given by an input video sequence. We found our method is able to accurately recover the parameters of 100 instances of deformable objects (cloth, balls, beams) as reported in Table 8.3 and Fig. 8.3.

8.3.1.3. Smoothness of the loss landscape in ∇Sim . Since ∇Sim is a complex combination of differentiable non-linear components, we analyze the loss landscape to verify the validity of gradients through the system. Fig. 8.4 illustrates the loss landscape when optimizing for the mass of a rigid body when all other physical properties are known.

We examine the image-space mean-squared error (MSE) of a unit-mass cube (1 kg) for a range of initializations (0.1 kg to 5 kg). Notably, the loss landscape of ∇Sim is well-behaved and conducive to momentum-based optimizers. Applying MSE to the first and last frames of the predicted and true videos provides the best gradients. However, for a naive gradient estimator applied to a non-differentiable simulator (PyBullet + REINFORCE), multiple local minima exist resulting in a very narrow region of convergence. This explains ∇Sim 's superior performance in Tables 8.1, 8.2, 8.3.


Fig. 8.4. Loss landscapes when optimizing for physical attributes using ∇Sim . (*Left*) When estimating the mass of a rigid-body with known shape using ∇Sim , despite images being formed by a highly nonlinear process (simulation), the loss landscape is remarkably smooth, for a range of initialization errors. (*Right*) when optimizing for the elasicity parameters of a deformable FEM solid. Both the Lamé parameters λ and μ are set to 1000, where the MSE loss has a unique, dominant minimum. Note that, for fair comparison, the ground-truth for our PyBullet+REINFORCE baseline was generated using the PyBullet engine.

8.3.2. Visuomotor control

To investigate whether the gradients computed by ∇Sim are meaningful for vision-based tasks, we conduct a range of visuomotor control experiments involving the actuation of deformable objects towards a visual target pose (a single image). In all cases, we evaluate against diffphysics, which uses a goal specification and a reward, both defined over the 3D state-space.

8.3.2.1. Deformable solids. The first example (control-walker) involves a 2D walker model. Our goal is to train a neural network (NN) control policy to actuate the walker to reach a target pose on the right-hand side of an image. Our NN consists of one fully connected layer and a tanh activation. The network input is a set of 8 time-varying sinusoidal signals, and the output is a scalar activation value per-tetrahedron. ∇Sim



Fig. 8.5. Visuomotor Control: ∇Sim provides gradients suitable for diverse, complex visuomotor control tasks. For control-fem and control-walker experiments, we train a neural network to actuate a soft body towards a target *image* (GT). For control-cloth, we optimize the cloth's initial velocity to hit a target (GT) (specified as an image), under nonlinear lift/drag forces.

is able to *solve* this environment within three iterations of gradient descent, by minimizing a pixelwise MSE between the last frame of the rendered video and the goal image as shown in Fig. 8.5 (lower left).



Fig. 8.6. Results of various approaches on the **control-fem** environment (6 randomseeds; each randomseed corresponds to a different goal configuration). While *diffphysics* performs well, it assumes strong 3D supervision. In contrast, ∇Sim is able to *solve* the task by using just a *single image* of the target configuration.

In our second test, we formulate a more challenging 3D control problem (control-fem) where the goal is to actuate a soft-body FEM object (a gear) consisting of 1152 tetrahedral elements to move to a target position as shown in Fig. 8.5 (center). We use the same NN architecture as in the 2D walker example, and use the Adam [255] optimizer to minimize a pixelwise MSE loss. We also train a privileged baseline (diffphysics) that uses strong supervision and minimizes the MSE between the target position and the precise 3D location of the center-of-mass (COM) of the FEM model at each time step (i.e. a dense reward). We test both diffphysics and ∇Sim against a naive baseline that generates random activations and plot convergence behaviors in Fig. 8.6.

While *diffphysics* appears to be a strong performer on this task, it is important to note that it uses explicit 3D supervision at each timestep (i.e. 30 FPS). In contrast, ∇Sim uses a *single image* as an implicit target, and yet manages to achieve the goal state, albeit taking a longer number of iterations.

8.3.2.2. Cloth (control-cloth). We design an experiment to control a piece of cloth by optimizing the initial velocity such that it reaches a pre-specified target. In each *episode*, a random cloth is spawned, comprising between 64 and 2048 triangles, and a new start/goal combination is chosen.

In this challenging setup, we notice that *state-based* MPC (*diffphysics*) is often unable to accurately reach the target. We believe this is due to the underdetermined nature of the problem, since, for objects such as cloth, the COM by itself does not uniquely determine

the configuration of the object. Visuomotor control on the other hand, provides a more well-defined problem. An illustration of the task is presented in Fig. 8.5 (column 3), and the convergence of the methods shown in Fig. 8.7.



Fig. 8.7. Results on **control-cloth** environment (5 randomseeds; each controls the dimensions and initial/target poses of the cloth). *diffphysics* converges to a suboptimal solution due to ambiguity in specifying the pose of a cloth via its center-of-mass. ∇Sim solves the environment using a single target image.

8.3.3. Impact of imperfect dynamics and rendering models

Being a white box method, the performance of ∇Sim relies on the choice of dynamics and rendering models employed. An immediate question that arises is "how would the performance of ∇Sim be impacted (if at all) by such modeling choices." We conduct multiple experiments targeted at investigating modelling errors and summarize them in Table 8.4 (left).

We choose a dataset comprising 90 objects equally representing rigid, deformable, and cloth types. By not modeling specific dynamics and rendering phenomena, we create the following 5 variants of our simulator.

- (1) Unmodeled friction: We model all collisions as being frictionless.
- (2) Unmodeled elasticity: We model all collisions as perfectly elastic.
- (3) *Rigid-as-deformable*: All rigid objects in the dataset are modeled as deformable objects.
- (4) *Deformable-as-rigid*: All deformable objects in the dataset are modeled as rigid objects.
- (5) Photorealistic render: We employ a photorealistic renderer—as opposed to ∇Sim 's differentiable rasterizers—in generating the target images.

In all cases, we evaluate the accuracy with which the mass of the target object is estimated from a target video sequence devoid of modeling discrepancies. In general, we observe that imperfect dynamics models (i.e. unmodeled friction and elasticity, or modeling a rigid object as deformable or vice-versa) have a more profound impact on parameter identification compared to imperfect renderers.

8.3.3.1. Unmodeled dynamics phenomenon. From Table 8.4 (left), we observe a noticeable performance drop when dynamics effects go unmodeled. Expectedly, the repurcussions of incorrect object type modeling (Rigid-as-deformable, Deformable-as-rigid) are more severe compared to unmodeled contact parameters (friction, elasticity). Modeling a deformable body as a rigid body results in irrecoverable deformation parameters and has the most severe impact on the recovered parameter set.

8.3.3.2. Unmodeled rendering phenomenon. We also independently investigate the impact of unmodeled rendering effects (assuming perfect dynamics). We independently render ground-truth images and object foreground masks from a photorealistic renderer [230]. We use these photorealistic renderings for ground-truth and perform physical parameter estimation from video. We notice that the performance obtained under this setting is superior compared to ones with dynamics model imperfections.

	Mean Bel. Abs. Err.					
	infodali feori ffibbi Elifi	Tetrahedra (#)	Forward (DP)	Forward (DR)	Backward (DP)	Backward $(DP + DR)$
Unmodeled friction	0.1866	100	9057 Hz	3504 Hz	3721 Hz	3057 Hz
Unmodeled elasticity	0.2281	200	9057 Hz	3478 Hz	3780 Hz	2963 Hz
Rigid-as-deformable	0.3462	400	8751 Hz	3357 Hz	3750 Hz	1360 Hz
	0.0102	1000	4174 Hz	1690 Hz	1644 Hz	1041 Hz
Deformable-as-rigid	0.4974	2000	3967 Hz	1584 Hz	1655 Hz	698 Hz
Photorealistic render	0.1793	5000	3871 Hz	1529 Hz	1553 Hz	424 Hz
Perfect model	0 1071	10000	3721 Hz	1500 Hz	1429 Hz	248 Hz

Table 8.4. (*Left*) **Impact of imperfect models**: The accuracy of physical parameters estimated by ∇Sim is impacted by the choice of dynamics and graphics (rendering) models. We find that the system is more sensitive to the choice of dynamics models than to the rendering engine used. (*Right*) **Timing analysis**: We report runtime in simulation steps / second (Hz). ∇Sim is significantly faster than real-time, even for complex geometries.

8.3.3.3. Impact of shading and texture cues. Although our work does not attempt to bridge the reality gap, we show early prototypes to assess phenomena such as shading/texture. Fig. 8.8 shows the accuracy over time for mass estimation from video. We evaluate three variants of the renderer - "Only color", "Shading", and "Texture". The "Only color" variant renders each mesh element in the same color regardless of the position and orientation of the light source. The "Shading" variant implements a Phong shading model and can model specular and diffuse reflections. The "Texture" variant also applies a non-uniform texture sampled from ShapeNet [250]. We notice that shading and texture cues significantly improve convergence speed. This is expected, as vertex colors often have very little appearance cues inside the object boundaries, leading to poor correspondences between the rendered and



Fig. 8.8. Including shading and texture cues lead to faster convergence. Inset plot has a logarithmic Y-axis.

ground-truth images. Furthermore, textures seem to offer slight improvements in convergence speed over shaded models, as highlighted by the inset (log scale) plot in Fig. 8.8.

8.3.3.4. Timing analysis. Table 8.4 (right) shows simulation rates for the forward and backward passes of each module. We report forward and backward pass rates separately for the differentiable physics (DP) and the differentiable rendering (DR) modules. The time complexity of ∇Sim is a function of the number of tetrahedrons and/or triangles. We illustrate the arguably more complex case of deformable object simulation for varying numbers of tetrahedra (ranging from 100 to 10000). Even in the case of 10000 tetrahedra—enough to contruct complex mesh models of multiple moving objects— ∇Sim enables faster-than-real-time simulation (1500 steps/second).

8.4. Related work

Differentiable physics simulators have seen significant attention and activity, with efforts centered around embedding physics structure into autodifferentiation frameworks. This has enabled differentiation through contact and friction models [241, 242, 243, 244, 227, 245, 246], latent state models [256, 257, 258, 259], volumetric soft bodies [260, 261, 228, 64], as well as particle dynamics [257, 262, 263, 64]. In contrast, ∇Sim addresses a superset of simulation scenarios, by coupling the physics simulator with a differentiable rendering pipeline. It also supports tetrahedral FEM-based hyperelasticity models to simulate deformable solids and thin-shells.

Recent work on **physics-based deep learning** injects structure in the latent space of the dynamics using Lagrangian and Hamiltonian operators [224, 264, 265, 266, 225, 267], by explicitly conserving physical quantities, or with ground truth supervision [268, 269, 220].

Sensor readings have been used to predicting the effects of forces applied to an object in models of learned [270, 271] and intuitive physics [249, 272, 273, 274, 275, 276, 277, 278, 279, 280]. This also includes approaches that learn to model multi-object interactions [219, 220, 221, 281, 222, 282]. In many cases, intuitive physics approaches are limited in their prediction horizon and treatment of complex scenes, as they do not sufficiently accurately model the 3D geometry nor the object properties. System identification based on parameterized physics models [283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294] and inverse simulation [295] are closely related areas.

There is a rich literature on **neural image synthesis**, but we focus on methods that model the 3D scene structure, including voxels [296, 297, 298, 299, 300], meshes [301, 302, 303, 304, 305], and implicit shapes [306, 307, 308, 309, 310, 311, 312]. Generative models condition the rendering process on samples of the 3D geometry [313]. Latent factors determining 3D structure have also been learned in generative models [223, 314]. Additionally, implicit neural representations that leverage differentiable rendering have been proposed [315, 316] for realistic view synthesis. Many of these representations have become easy to manipulate through software frameworks like Kaolin [317], Open3D [318], and PyTorch3D [319].

Differentiable rendering allows for image gradients to be computed w.r.t. the scene geometry, camera, and lighting inputs. Variants based on the rasterization paradigm (NMR [110], OpenDR [109], SoftRas [111]) blur the edges of scene triangles prior to image projection to remove discontinuities in the rendering signal. DIB-R [112] applies this idea to background pixels and proposes an interpolation-based rasterizer for foreground pixels. More sophisticated differentiable renderers can treat physics-based light transport processes [113, 114] by ray tracing, and more readily support higher-order effects such as shadows, secondary light bounces, and global illumination.

8.5. Conclusion

We presented ∇Sim , a versatile differentiable simulator that enables system identification from videos by differentiating through physical processes governing dyanmics and image formation. We demonstrated the benefits of such a holistic approach by estimating physical attributes for time-evolving scenes with complex dynamics and deformations, all from raw video observations. We also demonstrated the applicability of this efficient and accurate estimation scheme on end-to-end visuomotor control tasks. The latter case highlights ∇Sim 's efficient integration with PyTorch, facilitating interoperability with existing machine learning modules. Interesting avenues for future work include extending our differentiable simulation to contact-rich motion, articulated bodies and higher-fidelity physically-based renderers – doing so takes us closer to operating in the real-world.

Conclusion

All models are wrong, but some are useful

George Box

At the core of this dissertation lies the idea that "you need not learn what you already know". Gradient-based learners such as neural networks do not need to re-learn concepts of projective geometry, optimization, search algorithms, or image formation, as a robust and well-founded theory for these elements already exists.

We present simple, powerful, and effective approaches spanning all three subsystems of the *sense-plan-act* computational model for embodied agents. Drawing upon work in diverse fields (computer vision, graphics, robotics, and machine learning), the ideas presented in this dissertation enable multiple challenging applications that have thus far remained out of reach for deep learning methods.

- Differentiable programming for learned perception (SLAM) systems: Realizing the visual SLAM pipeline as a differentiable function opens up new ways of integrating deep learning with SLAM. Where prior approaches have attempted to completely replace SLAM pipelines with deep neural networks, ∇ SLAM enables learning in only components of the SLAM system that warrant it, while retaining useful inductive biases from classical robot vision – results of decades of research.
- Compact representations for symbolic planning: To enable robots to plan longhorizon tasks in large-scale environments, we leverage the function approximation capabilities of deep neural networks to prune a 3D scene graph to only retain scene attributes that are relevant to a downstream task. Inductive biases imposed by 3D scene graphs enable classical planners to perform on-par-with learning-based planners.
- Differentiable simulation for visuomotor control: By bringing in ideas from the computer graphics and machine learning communities, we construct differentiable programs that model complex phenomenon such as multiphysics and light transport. Such differentiable world programs can solve extremely challenging tasks, such as enabling visuomotor control from a single image goal specification.

Contemporary and follow-up work: We are beginning to see a number of other researchers adding to the lines of work presented in this dissertation.

- Sodhi *et al.* [320] and Yi *et al.* [321] present alternate ways of differentiating through factor graph optimization for state estimation applications. Karkus *et al.* [322] demonstrate that differentiable SLAM systems learn representations more amenable to downstream tasks and can be trained end-to-end for embodied navigation tasks.
- RISP [323] builds atop ∇Sim and proposes novel ideas, including the learning of high-dimensional state representations that are invariant to rendering configurations. This enables visuomotor control and system identification from real-world videos.

9.1. Discussion

We briefly reflect on the role of differentiable world programs in enabling the design of intelligent embodied agents. The central question addressed in this dissertation is one that has sparked lively debate among AI researchers for the past decade – "do classical techniques matter in this day and age of deep learning"?

Today, most visual recognition pipelines are solely composed of deep neural networks. Several commercial products and services already leverage deep learning for visual recognition, speech processing, and natural language processing. However, all of these applications mandate the availability of large, manually labelled training datasets. In the last two years, multiple approaches (e.g., GPT-3 [324], CLIP [325]) have demonstrated that—given abundant data and compute—model-free approaches with minimal inductive priors can match and surpass human-level performance on such pattern recognition tasks.

However, embodied agents acting in an environment require far superior and complex capabilities. Task-centric perception, symbolic and low-level planning, and sensorimotor control all rely on complex reasoning and perceptual organization that is seemingly trivial for humans, but impossible for existing AI systems. This is further exacerbated by the (relatively) minute quantities of real-world data available for learning these intricate behaviors. It is extremely difficult to collect large-scale real-world datasets for tasks involving embodied intelligence, owing to the cost of physical interaction and the impossibility of emulating all possible interaction scenarios – particularly ones that are safety-critical. Even if it were possible to gather such data, there are no guarantees that models trained on such data will generalize to other robots and/or sensorimotor configurations. Taking this hypothetical scenario a step further, we will then be faced with the unduly challenging task of producing accurate, task-specific labels for the multi-dimensional data thus obtained.

This suggests that for the forseeable future, intelligent agents will require data and label-efficient training strategies. The most effective way to simplify a learning problem in such a setting is therefore to impart our knowledge of the world when instantiating the model (we refer to this as *specified knowledge*) and allow the model the flexibility to update this knowledge specification (we refer to this as *acquired knowledge*). This dissertation explores the capabilities of differentiable programming in infusing such specified knowledge into modern, learning-based approaches in a prescriptive manner. However, *differentiable world programs* are not without their own set of shortcomings, which we briefly discuss below.

9.2. Known limitations and future directions

We identify the following four primary limitations of differentiable world programs that remain to be addressed in future work.

- Capturing uncertainty: In most settings discussed in this dissertation, gradientbased optimization relies crucially on an initial guess. The value of the initial guess can severely impact the optimization process, leading to a high variance in the resultant estimates. An extremely promising avenue for future work is to leverage automated gradient-based probabilistic inference [326, 327, 328] to estimate posterior distributions over parameter vectors, as opposed to point estimates that do not capture uncertainty. The complementary strengths of differentiable and probabilistic programming approaches are thus ripe for exploration.
- Incorporating non-differentiable components: There are several classes of computations for which gradients may not exist or are degenerate (e.g., a discrete action chosen by a symbolic planner is non-differentiable). Crucially, the existence of even a small number of such computations may severely impact gradient flow through large computation graphs. Designing gradient estimators for such operations, or reparameterizing them by leveraging neural networks as differentiable decision machines will widen the set of applications differentiable programs are currently applied to.
- Robustness to model misspecifications: A major challenge in model-based learning paradigms occurs due to unmodelled attributes or effects. For example, it is infeasible to model atmospheric effects such as turbulence and wear-and-tear within a differentiable simulator, while ensuring gradient flow through all parameters of interest. This suggests that a flexible blend of model-based and model-free approaches is desirable. We are beginning to see flavours of this paradigm in recent work [226, 329], but much remains to be done to tightly integrate these seemingly disparate and complementary paradigms.
- Discovering symbols, properties, and inductive biases: To achieve true, indefinite autonomy, embodied agents will need to adapt to their ever-changing environment; discovering inductive biases from noisy, low-dimensional observations. Aspects to be discovered include newer symbols (i.e., objects, world phenomena), their properties, and possibly their governing equations and the uncertainty surrounding them. This is a promising direction for longer-term research and relies crucially on efficient solutions to the above three open problems.

Final remarks

This work bridges the vision, robotics, graphics, and learning communities. Each community has advanced greatly in recent years both in terms of performance and feasibility: robots have become affordable, learning methods have enabled huge strides in vision and graphics; but by and large, these fields have evolved in isolation. Building autonomous and truly intelligent agents will thus require insights from all these communities. This dissertation fundamentally questions the way we think about building autonomous embodied agents, and presents initial steps towards blending classical approaches with modern machine learning.

References

- R. J Firby. Adaptive Execution in Dynamic Domains. PhD thesis, Departament of Computer Science, Yale University, 1989. 31
- [2] Erann Gat, R. Peter Bonnasso, Robin Murphy, and Aaai Press. On three-layer architectures. In Artificial Intelligence and Mobile Robots, pages 195–210. AAAI Press, 1997. 31
- [3] Rodney Brooks. A robust layered control system for a mobile robot. *IEEE journal on robotics and automation*, 2(1):14–23, 1986. 32
- [4] David P Miller, Rajiv S Desai, Erann Gat, Robert Ivlev, and John Loch. Reactive navigation through rough terrain: Experimental results. In AAAI, pages 823–828. Citeseer, 1992. 33
- [5] Erann Gat, Rajiv Desai, Robert Ivlev, John Loch, and David P Miller. Behavior control for robotic exploration of planetary surfaces. *IEEE Transactions on Robotics and Automation*, 10(4):490–503, 1994.
 33
- Sebastian Thrun, Wolfram Burgard, and Dieter Fox. Probabilistic robotics. MIT Press, Cambridge, Mass., 2005. 19, 33, 34, 36
- [7] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. Transactions of the ASME-Journal of Basic Engineering, 82(Series D):35–45, 1960. 34
- [8] Eric A Wan and Rudolph Van Der Merwe. The unscented kalman filter for nonlinear estimation. In Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No. 00EX373), pages 153–158. IEEE, 2000. 35
- [9] Pierre Del Moral. Nonlinear filtering: Interacting particle resolution. Comptes Rendus de l'Académie des Sciences-Series I-Mathematics, 325(6):653-658, 1997. 35
- [10] H. F. Durrant-Whyte and T. Bailey. Simultaneous Localisation and Mapping (SLAM): Part I. IEEE Robotics and Automation Magazine, 13(2):99–110, 2006. 37
- [11] Feng Lu and Evangelos Milios. Globally consistent range scan alignment for environment mapping. Autonomous robots, 4(4):333–349, 1997. 37
- [12] J-S Gutmann and Kurt Konolige. Incremental mapping of large cyclic environments. In Proceedings 1999 IEEE International Symposium on Computational Intelligence in Robotics and Automation. CIRA'99 (Cat. No. 99EX375), pages 318–325. IEEE, 1999. 37
- [13] Frank Dellaert and Michael Kaess. Square root sam: Simultaneous localization and mapping via square root information smoothing. The International Journal of Robotics Research, 25(12):1181–1203, 2006.
 37
- [14] Michael Kaess, Ananth Ranganathan, and Frank Dellaert. isam: Fast incremental smoothing and mapping with efficient data association. In *Proceedings 2007 IEEE International Conference on Robotics* and Automation, pages 1670–1677. IEEE, 2007. 37

- [15] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g 2 o: A general framework for graph optimization. In 2011 IEEE International Conference on Robotics and Automation, pages 3607–3613. IEEE, 2011. 37
- [16] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J.J. Leonard. Past, present, and future of simultaneous localization and mapping: Towards the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016. 37
- [17] Constructions Aeronautiques, Adele Howe, Craig Knoblock, ISI Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, David Wilkins SRI, Anthony Barrett, Dave Christianson, et al. Pddl| the planning domain definition language. Technical report, Technical Report, 1998. 40
- [18] Derek Long, Henry Kautz, Bart Selman, Blai Bonet, Hector Geffner, Jana Koehler, Michael Brenner, Jörg Hoffmann, Frank Rittinger, Corin R Anderson, et al. The aips-98 planning competition. AI magazine, 21(2):13–13, 2000. 40
- [19] Håkan LS Younes and Michael L Littman. Ppddl1. 0: An extension to pddl for expressing planning domains with probabilistic effects. *Techn. Rep. CMU-CS-04-162*, 2:99, 2004. 41
- [20] Stefan Edelkamp and Jörg Hoffmann. Pddl2. 2: The language for the classical part of the 4th international planning competition. Technical report, Technical Report 195, University of Freiburg, 2004. 41
- [21] Maria Fox and Derek Long. Pddl+: Modeling continuous time dependent effects. In *Proceedings of the* 3rd International NASA Workshop on Planning and Scheduling for Space, volume 4, page 34, 2002. 41
- [22] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 440–448, 2020. 41
- [23] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. Artificial intelligence, 2(3-4):189–208, 1971. 41
- [24] Blai Bonet and Hector Geffner. Planning as heuristic search: New results. In European Conference on Planning, pages 360–372. Springer, 1999. 42, 85
- [25] Jörg Hoffmann and Bernhard Nebel. The ff planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research, 14:253–302, 2001. 42, 85, 88
- [26] Malte Helmert. The fast downward planning system. Journal of Artificial Intelligence Research, 26:191– 246, 2006. 42, 43, 85, 88
- [27] Henry Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In Proceedings of the National Conference on Artificial Intelligence, pages 1194–1201. Citeseer, 1996. 42, 85
- [28] Henry Kautz and Bart Selman. Unifying sat-based and graph-based planning. In *IJCAI*, volume 99, pages 318–325, 1999. 42, 85
- [29] Henry Kautz, Bart Selman, and Jörg Hoffmann. Satplan: Planning as satisfiability. In 5th international planning competition, volume 20, page 156, 2006. 42, 85, 88, 89
- [30] Michael Katz, Shirin Sohrabi, Horst Samulowitz, and Silvan Sievers. Delfi: Online planner selection for cost-optimal planning. *IPC-9 planner abstracts*, pages 57–64, 2018. 42, 85, 88
- [31] Qiang Yang. Intelligent planning: a decomposition and abstraction based approach. Springer Science & Business Media, 2012. 42
- [32] Jörg Hoffmann, Julie Porteous, and Laura Sebastia. Ordered landmarks in planning. Journal of Artificial Intelligence Research, 22:215–278, 2004. 42
- [33] James E Kelley Jr. Critical-path planning and scheduling: Mathematical basis. Operations research, 9(3):296–320, 1961. 42

- [34] Blai Bonet and Héctor Geffner. Planning as heuristic search. Artificial Intelligence, 129(1-2):5–33, 2001.
 42, 85
- [35] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, 2020. 42, 86, 87
- [36] Tom Silver, Rohan Chitnis, Aidan Curtis, Joshua Tenenbaum, Tomas Lozano-Perez, and Leslie Pack Kaelbling. Planning with learned object importance in large problem instances using graph neural networks. AAAI International conference on Artificial Intelligence, 2020. 15, 42, 43, 84, 85, 89, 90, 91, 95, 96, 146, 147, 148
- [37] Avrim L Blum and Merrick L Furst. Fast planning through planning graph analysis. Artificial intelligence, 90(1-2):281–300, 1997. 42
- [38] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012. 42, 85
- [39] Richard Waldinger. Achieving several goals simultaneously. In *Readings in artificial intelligence*, pages 250–271. Elsevier, 1981. 42
- [40] Rohan Chitnis, Tom Silver, Beomjoon Kim, Leslie Pack Kaelbling, and Tomas Lozano-Perez. Camps: Learning context-specific abstractions for efficient planning in factored mdps. arXiv preprint arXiv:2007.13202, 2020. 43, 85
- [41] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016. 43
- [42] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017. 43
- [43] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. arXiv preprint arXiv:1712.01815, 2017. 43
- [44] Danfei Xu, Roberto Martín-Martín, De-An Huang, Yuke Zhu, Silvio Savarese, and Li Fei-Fei. Regression planning networks. arXiv preprint arXiv:1909.13072, 2019. 43, 44, 85
- [45] Arthur Guez, Théophane Weber, Ioannis Antonoglou, Karen Simonyan, Oriol Vinyals, Daan Wierstra, Rémi Munos, and David Silver. Learning to search with mctsnets. In *International Conference on Machine Learning*, pages 1822–1831. PMLR, 2018. 43, 44, 85
- [46] Tom Silver and Rohan Chitnis. Pddlgym: Gym environments from pddl problems. arXiv preprint arXiv:2002.06432, 2020. 43, 89
- [47] Craig Boutilier, Nir Friedman, Moises Goldszmidt, and Daphne Koller. Context-specific independence in bayesian networks. arXiv preprint arXiv:1302.3562, 2013. 43
- [48] Jiri Baum, Ann E Nicholson, and Trevor I Dix. Proximity-based non-uniform abstractions for approximate planning. Journal of Artificial Intelligence Research, 43:477–522, 2012. 43
- [49] Craig Boutilier. Correlated action effects in decision theoretic regression. In UAI, pages 30–37, 1997. 43
- [50] Natalia Hernandez Gardiol. Relational envelope-based planning. 2007. 43
- [51] Lihong Li, Thomas J Walsh, and Michael L Littman. Towards a unified theory of state abstraction for mdps. ISAIM, 4:5, 2006. 43

- [52] George Dimitri Konidaris and Andrew G Barto. Efficient skill learning using abstraction selection. In IJCAI, volume 9, pages 1107–1112. Citeseer, 2009. 43
- [53] Steven M LaValle et al. Rapidly-exploring random trees: A new tool for path planning. 1998. 44
- [54] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. The international journal of robotics research, 30(7):846–894, 2011. 44
- [55] Steven M LaValle. Planning algorithms. Cambridge university press, 2006. 44
- [56] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Integrated task and motion planning in belief space. The International Journal of Robotics Research, 32(9-10):1194–1227, 2013. 44, 85
- [57] Rohan Chitnis, Dylan Hadfield-Menell, Abhishek Gupta, Siddharth Srivastava, Edward Groshev, Christopher Lin, and Pieter Abbeel. Guided search for task and motion plans using learned heuristics. In 2016 IEEE International Conference on Robotics and Automation (ICRA), pages 447–454. IEEE, 2016. 44
- [58] Gregory C Chow et al. Analysis and control of dynamic economic systems. Wiley, 1975. 45
- [59] Frank L Lewis, Draguna Vrabie, and Vassilis L Syrmos. Optimal control. John Wiley & Sons, 2012. 45
- [60] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602, 2013. 45
- [61] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In International conference on machine learning, pages 1928–1937. PMLR, 2016. 45
- [62] Bela G Liptak. Instrument engineers' handbook, volume two: Process control and optimization. CRC press, 2018. 45
- [63] B Amos, I Rodriguez, J Sacks, B Boots, and Z Kolter. Differentiable mpc for end-to-end planning and control. Neural Information Processing Systems (Neurips), 2018. 45
- [64] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. DiffTaichi: Differentiable programming for physical simulation. *International Conference on Learning Representations (ICLR)*, 2020. 46, 102, 104, 105, 113, 149, 150, 152, 154
- [65] Eric Heiden, David Millard, Erwin Coumans, Yizhou Sheng, and Gaurav S Sukhatme. Neuralsim: Augmenting differentiable simulators with neural networks. In 2020 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2021. 46
- [66] Mengyuan Yan, Gen Li, Yilin Zhu, and Jeannette Bohg. Learning topological motion primitives for knot planning. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2020. 46
- [67] Nils Thuerey. Differentiable fluid simulations for deep learning. In APS Division of Fluid Dynamics Meeting Abstracts, pages H17–006, 2019. 46
- [68] Samuel S Schoenholz and Ekin D Cubuk. JAX-MD: End-to-end differentiable, hardware accelerated, molecular dynamics in pure python, 2019. 46
- [69] The three r's of computer vision: Recognition, reconstruction and reorganization. Pattern Recognition Letters, 72:4–14, 2016. Special Issue on ICPR 2014 Awarded Papers. 46
- [70] David Nistér, Oleg Naroditsky, and James Bergen. Visual odometry. In Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004., volume 1, pages I–I. Ieee, 2004. 48
- [71] Hans Peter Moravec. Obstacle avoidance and navigation in the real world by a seeing robot rover. PhD thesis, Stanford University, 1980. 48

- [72] Christopher G Harris and JM Pike. 3d positional integration from image sequences. Image and Vision Computing, 6(2):87–90, 1988. 48
- [73] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1052–1067, 2007.
 48
- [74] Andrew John Davison. Mobile robot navigation using active vision. Advances in Scientific Philosophy Essays in Honour of, 1999. 48
- [75] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In 2007 6th IEEE and ACM international symposium on mixed and augmented reality, pages 225–234. IEEE, 2007.
 48
- [76] Mark Joseph Cummins and Paul M Newman. Fab-map: Appearance-based place recognition and mapping using a learned visual vocabulary model. In *ICML*, 2010. 49
- [77] Relja Arandjelovic, Petr Gronat, Akihiko Torii, Tomas Pajdla, and Josef Sivic. Netvlad: Cnn architecture for weakly supervised place recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5297–5307, 2016. 49
- [78] Stephanie Lowry, Niko Sünderhauf, Paul Newman, John J Leonard, David Cox, Peter Corke, and Michael J Milford. Visual place recognition: A survey. *IEEE Transactions on Robotics*, 32(1):1–19, 2015. 49
- [79] Sourav Garg, Tobias Fischer, and Michael Milford. Where is your place, visual place recognition? arXiv preprint arXiv:2103.06443, 2021. 49
- [80] Montiel J. M. M. Mur-Artal, Raúl and Juan D. Tardós. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015. 49
- [81] Raúl Mur-Artal and Juan D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017. 49
- [82] Frank Steinbrücker, Jürgen Sturm, and Daniel Cremers. Real-time visual odometry from dense rgb-d images. In ICCV Workshops, 2011. 49, 66
- [83] Christian Kerl, Jürgen Sturm, and Daniel Cremers. Robust odometry estimation for rgb-d cameras. In ICRA, 2013. 49, 66
- [84] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. Dtam: Dense tracking and mapping in real-time. In 2011 international conference on computer vision, pages 2320–2327. IEEE, 2011. 49
- [85] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew W Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *ISMAR*, 2011. 49, 64, 65, 67, 68, 71, 72, 73, 75
- [86] Thomas Whelan, Michael Kaess, Hordur Johannsson, Maurice Fallon, John J Leonard, and John McDonald. Real-time large-scale dense rgb-d slam with volumetric fusion. *The International Journal of Robotics Research*, 34(4-5):598–626, 2015. 49, 64, 71, 75
- [87] Maik Keller, Damien Lefloch, Martin Lambers, Shahram Izadi, Tim Weyrich, and Andreas Kolb. Real-time 3d reconstruction in dynamic scenes using point-based fusion. In 2013, 2013. 49, 64, 65, 67, 68, 70, 71, 72, 73, 75
- [88] Thomas Whelan, Stefan Leutenegger, R Salas-Moreno, Ben Glocker, and Andrew Davison. Elasticfusion: Dense slam without a pose graph. Robotics: Science and Systems, 2015. 49, 50
- [89] J. Engel, T. Schöps, and D. Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In ECCV, 2014. 50, 65

- [90] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. IEEE transactions on pattern analysis and machine intelligence, 40(3):611–625, 2017. 50
- [91] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In 2014 IEEE international conference on robotics and automation (ICRA), pages 15–22. IEEE, 2014. 50
- [92] Renato F Salas-Moreno, Richard A Newcombe, Hauke Strasdat, Paul HJ Kelly, and Andrew J Davison. Slam++: Simultaneous localisation and mapping at the level of objects. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1352–1359, 2013. 50
- [93] John McCormac, Ankur Handa, Andrew Davison, and Stefan Leutenegger. Semanticfusion: Dense 3d semantic mapping with convolutional neural networks. In 2017 IEEE International Conference on Robotics and automation (ICRA), pages 4628–4635. IEEE, 2017. 50
- [94] Parv Parkhiya, Rishabh Khawad, J. Krishna Murthy, K. Madhava Krishna, and Brojeshwar Bhowmick. Constructing category-specific models for monocular object slam. *IEEE International Conference on Robotics and Automation (ICRA)*, 2018. 50
- [95] Sudeep Pillai and John Leonard. Monocular slam supported object recognition. Robotics Science and Systems (R-SS), 2015. 50
- [96] Beipeng Mu, Shih-Yuan Liu, Liam Paull, John Leonard, and Jonathan P How. Slam with objects using a nonparametric pose graph. *IEEE/RSJ International Conference onnIntelligent Robots and Systems* (*IROS*), 2016. 50
- [97] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. Unsupervised learning of depth and ego-motion from video. In CVPR, 2017. 50, 66
- [98] K. Tateno, F. Tombari, I. Laina, and N. Navab. Cnn-slam: Real-time dense monocular slam with learned depth prediction. In CVPR, 2017. 50, 65
- [99] Michael Bloesch, Jan Czarnowski, Ronald Clark, Stefan Leutenegger, and Andrew J Davison. Codeslam—learning a compact, optimisable representation for dense visual slam. In CVPR, 2018. 50, 65
- [100] Jan Czarnowski, Tristan Laidlow, Ronald Clark, and Andrew J Davison. Deepfactors: Real-time probabilistic dense monocular slam. *IEEE Robotics and Automation Letters*, 5(2):721–728, 2020. 50
- [101] Kenneth Levenberg. A method for the solution of certain non linear problems in least squares. In Quart. Appl. Math, 1944. 52
- [102] Donald W Marquardt. An algorithm for least-squares estimation of nonlinear parameters. Journal of the society for Industrial and Applied Mathematics, 1963. 52
- [103] Michael Lampton. Damping-undamping strategies for the levenberg-marquardt nonlinear least-squares method. Computers in Physics, 1997. 52
- [104] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016. http://www. deeplearningbook.org. 52
- [105] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch, 2017. 55, 57, 105, 149
- [106] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay

Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 55, 57, 105, 149

- [107] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. JAX: composable transformations of Python+NumPy programs, 2018. 55, 57, 105, 149
- [108] Zico Kolter, David Duvenaud, and Matt Johnson. Deep implicit layers: Neural odes, deep equilibrium models, and beyond. http://implicit-layers-tutorial.org/, 2021. Accessed: 28 January, 2022. 57
- [109] Matthew M. Loper and Michael J. Black. Opendr: An approximate differentiable renderer. In Proceedings of the European Conference on Computer Vision (ECCV), 2014. 58, 104, 114
- [110] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In Proceedings of Computer Vision and Pattern Recognition (CVPR), 2018. 58, 104, 114
- [111] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for imagebased 3d reasoning. *Proceedings of International Conference on Computer Vision (ICCV)*, 2019. 58, 104, 105, 114, 152
- [112] Wenzheng Chen, Jun Gao, Huan Ling, Edward Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. Learning to predict 3d objects with an interpolation-based differentiable renderer. *Neural Information Processing Systems (Neurips)*, 2019. 58, 104, 105, 114, 152
- [113] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable monte carlo ray tracing through edge sampling. SIGGRAPH Asia, 37(6):222:1–222:11, 2018. 58, 114
- [114] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. Mitsuba 2: A retargetable forward and inverse renderer. *Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 38(6), 2019. 58, 114
- [115] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint: Self-supervised interest point detection and description. In *Proceedings of the IEEE conference on computer vision and pattern* recognition workshops, pages 224–236, 2018. 58
- [116] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superglue: Learning feature matching with graph neural networks. In *Proceedings of the IEEE/CVF conference on computer* vision and pattern recognition, pages 4938–4947, 2020. 58
- [117] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In NeurIPS, 2012. 63
- [118] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. 63
- [119] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Brian Kingsbury, et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 2012. 63
- [120] Shuaifeng Zhi, Michael Bloesch, Stefan Leutenegger, and Andrew J Davison. Scenecode: Monocular dense semantic reconstruction using learned encoded scene representations. In CVPR, 2019. 65
- [121] Huizhong Zhou, Benjamin Ummenhofer, and Thomas Brox. Deeptam: Deep tracking and mapping. In ECCV, 2018. 65
- [122] S. Yang and S. Scherer. Cubeslam: Monocular 3-d object slam. IEEE Transactions on Robotics, 35(4):925–938, Aug 2019. 65
- [123] B. Mu, S. Liu, L. Paull, J. Leonard, and J. P. How. Slam with objects using a nonparametric pose graph. In 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 4602–4609, Oct 2016. 65

- [124] P. Parkhiya, R. Khawad, J. K. Murthy, B. Bhowmick, and K. M. Krishna. Constructing category-specific models for monocular object-slam. In 2018 IEEE International Conference on Robotics and Automation (ICRA), pages 1–9, May 2018. 65
- [125] Bert De Brabandere, Wouter Van Gansbeke, Davy Neven, Marc Proesmans, and Luc Van Gool. End-to-end lane detection through differentiable least-squares fitting. CoRR, abs/1902.00293, 2019. 65
- [126] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *ICCV*, 2015. 65
- [127] Devendra Singh Chaplot, Emilio Parisotto, and Ruslan Salakhutdinov. Active neural localization. In International Conference on Learning Representations, 2018. 65
- [128] S. K. Gottipati, K. Seo, D. Bhatt, V. Mai, K. Murthy, and L. Paull. Deep active localization. *IEEE Robotics and Automation Letters*, 4(4):4394–4401, Oct 2019. 65
- [129] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision, 1981. 66
- [130] Ravi Garg, Vijay Kumar BG, Gustavo Carneiro, and Ian Reid. Unsupervised cnn for single view depth estimation: Geometry to the rescue. In ECCV, 2016. 66
- [131] Ruihao Li, Sen Wang, Zhiqiang Long, and Dongbing Gu. Undeepvo: Monocular visual odometry through unsupervised deep learning. In *IEEE International Conference on Robotics and Automation* (*ICRA*), 2018. 66
- [132] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In Neurips, 2015. 66
- [133] Ankur Handa, Michael Bloesch, Viorica Pătrăucean, Simon Stent, John McCormac, and Andrew Davison. gvnn: Neural network library for geometric computer vision. In ECCV Workshop on Geometry Meets Deep Learning, 2016. 66
- [134] Edgar Riba, Dmytro Mishkin, Daniel Ponsa, Ethan Rublee, and Gary Bradski. Kornia: an open source differentiable computer vision library for pytorch. In Winter Conference on Applications of Computer Vision, 2020. 66
- [135] Chengzhou Tang and Ping Tan. Ba-net: Dense bundle adjustment network. ICLR, 2019. 66
- [136] Ronald Clark, Michael Bloesch, Jan Czarnowski, Stefan Leutenegger, and Andrew J Davison. Ls-net: Learning to solve nonlinear least squares for monocular stereo. ECCV, 2018. 66, 74
- [137] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural computation, 9:1735–80, 12 1997. 66
- [138] Lukas von Stumberg, Patrick Wenzel, Qadeer Khan, and Daniel Cremers. Gn-net: The gauss-newton loss for deep direct SLAM. CoRR, abs/1904.11932, 2019. 66
- [139] Lei Han, Mengqi Ji, Lu Fang, and Matthias Nießner. Regnet: Learning the optimization of direct image-to-image pose registration. CoRR, abs/1812.10212, 2018. 66
- [140] Brandon Amos and J. Zico Kolter. OptNet: Differentiable optimization as a layer in neural networks. In *ICML*, 2017. 66
- [141] Edward Grefenstette, Brandon Amos, Denis Yarats, Phu Mon Htut, Artem Molchanov, Franziska Meier, Douwe Kiela, Kyunghyun Cho, and Soumith Chintala. Generalized inner loop meta-learning arXiv preprint arXiv:1910.01727, 2019. 66
- [142] Michael Lampton. Damping-undamping strategies for the levenberg-marquardt nonlinear least-squares method. Computers in Physics, 1997. 68
- [143] FJ Richards. A flexible growth function for empirical use. Journal of experimental Botany, 1959. 69, 72

- [144] Chuong V Nguyen, Shahram Izadi, and David Lovell. Modeling kinect sensor noise for improved 3d reconstruction and tracking. In 2012 second international conference on 3D imaging, modeling, processing, visualization & transmission. IEEE, 2012. 70
- [145] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, pages 303–312, 1996. 70
- [146] Steven Parker, Peter Shirley, Yarden Livnat, Charles Hansen, and P-P Sloan. Interactive ray tracing for isosurface rendering. In *Proceedings Visualization'98 (Cat. No. 98CB36276)*, pages 233–238. IEEE, 1998. 71
- [147] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In Advances in Neural Information Processing Systems, 2019. 71
- [148] Shubham Tulsiani, Tinghui Zhou, Alexei A. Efros, and Jitendra Malik. Multi-view supervision for single-view reconstruction via differentiable ray consistency. In CVPR, 2017. 71
- [149] JunYoung Gwak, Christopher B Choy, Manmohan Chandraker, Animesh Garg, and Silvio Savarese. Weakly supervised 3d reconstruction with adversarial constraint. In 2017 International Conference on 3D Vision (3DV), 2017. 71
- [150] Homan Igehy. Tracing ray differentials. In Proceedings of the 26th annual conference on Computer graphics and interactive techniques, pages 179–186. ACM Press/Addison-Wesley Publishing Co., 1999.
 71
- [151] A. Handa, T. Whelan, J.B. McDonald, and A.J. Davison. A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM. In *ICRA*, 2014. 20, 75
- [152] 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), 2012. 75
- [153] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In Proc. Computer Vision and Pattern Recognition (CVPR), IEEE, 2017. 20, 75, 77
- [154] Angela Dai, Matthias Nießner, Michael Zollöfer, Shahram Izadi, and Christian Theobalt. Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface re-integration. ACM Transactions on Graphics 2017 (TOG), 2017. 20, 77
- [155] Benjamin Kuipers. The spatial semantic hierarchy. Artificial intelligence, 119(1-2):191–233, 2000. 84
- [156] Emilio Remolina and Benjamin Kuipers. Towards a general theory of topological maps. Artificial Intelligence, 152(1):47–104, 2004. 84
- [157] Cipriano Galindo, Alessandro Saffiotti, Silvia Coradeschi, Pär Buschka, Juan-Antonio Fernandez-Madrigal, and Javier González. Multi-hierarchical semantic maps for mobile robotics. In 2005 IEEE/RSJ international conference on intelligent robots and systems, pages 2278–2283. IEEE, 2005. 84
- [158] Hendrik Zender, O Martínez Mozos, Patric Jensfelt, G-JM Kruijff, and Wolfram Burgard. Conceptual spatial representations for indoor mobile robots. *Robotics and Autonomous Systems*, 56(6):493–502, 2008. 84
- [159] Jose-Raul Ruiz-Sarmiento, Cipriano Galindo, and Javier Gonzalez-Jimenez. Building multiversal semantic maps for mobile robot operation. *Knowledge-Based Systems*, 119:257–272, 2017. 84

- [160] Antoni Rosinol, Andrew Violette, Marcus Abate, Nathan Hughes, Yun Chang, Jingnan Shi, Arjun Gupta, and Luca Carlone. Kimera: from slam to spatial perception with 3d dynamic scene graphs. arXiv preprint arXiv:2101.06894, 2021. 84, 90
- [161] Iro Armeni, Zhi-Yang He, JunYoung Gwak, Amir R Zamir, Martin Fischer, Jitendra Malik, and Silvio Savarese. 3d scene graph: A structure for unified semantics, 3d space, and camera. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, 2019. 16, 84, 86, 87, 141, 142, 143
- [162] Antoni Rosinol, Arjun Gupta, Marcus Abate, Jingnan Shi, and Luca Carlone. 3d dynamic scene graphs: Actionable spatial perception with places, objects, and humans. In *Robotics Science and Systems (R-SS)*, 2020. 84, 86, 87, 143
- [163] Ue-Hwan Kim, Jin-Man Park, Taek-Jin Song, and Jong-Hwan Kim. 3-d scene graph: A sparse and semantic representation of physical environments for intelligent agents. *IEEE transactions on cybernetics*, 50(12):4921–4933, 2019. 84, 87
- [164] Zachary Ravichandran, Lisa Peng, Nathan Hughes, J Daniel Griffith, and Luca Carlone. Hierarchical representations and explicit memory: Learning effective navigation policies on 3d scene graphs using graph neural networks. arXiv preprint arXiv:2108.01176, 2021. 84
- [165] Andrey Kurenkov, Roberto Martín-Martín, Jeff Ichnowski, Ken Goldberg, and Silvio Savarese. Semantic and geometric modeling with neural message passing in 3d scene graphs for hierarchical mechanical search. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020. 84
- [166] Rajat Talak, Siyi Hu, Lisa Peng, and Luca Carlone. Neural trees for learning on graphs. arXiv preprint arXiv:2105.07264, 2021. 84
- [167] H Kaindl. Tree searching algorithms. Computers, Chess, and Cognition, pages 133–158, 1990. 85
- [168] Michael Katz. Cerberus: Red-black heuristic for planning tasks with conditional effects meets novelty heuristic and enchanced mutex detection. *IPC-9 planner abstracts*, pages 47–51, 2018. 85, 88
- [169] Daniel Gnad, Alexander Shleyfman, and Jörg Hoffmann. Decstar-star-topology decoupled search at its best. *IPC-9 planner abstracts*, pages 42–46, 2018. 85, 88
- [170] Derek Long and Maria Fox. The 3rd international planning competition: Results and analysis. Journal of Artificial Intelligence Research, 20:1–59, 2003. 85
- [171] Mauro Vallati, Lukas Chrpa, Marek Grześ, Thomas Leo McCluskey, Mark Roberts, Scott Sanner, et al. The 2014 international planning competition: Progress and trends. Ai Magazine, 36(3):90–98, 2015. 85, 89
- [172] Cipriano Galindo, J-A Fernandez-Madrigal, and Javier Gonzalez. Improving efficiency in mobile robot task planning through world abstraction. *IEEE Transactions on Robotics (T-RO)*, 20(4):677–690, 2004.
 85
- [173] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996. 85
- [174] Cipriano Galindo, Juan-Antonio Fernández-Madrigal, Javier González, and Alessandro Saffiotti. Robot task planning using semantic maps. *Robotics and Autonomous Systems*, 56(11):955–966, 2008. 85
- [175] Beomjoon Kim, Zi Wang, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning to guide task and motion planning using score-space representation. *The International Journal of Robotics Research*, 38(7):793–812, 2019. 85
- [176] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. Annual review of control, robotics, and autonomous systems, 4:265–293, 2021. 85

- [177] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical planning in the now. In Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence, 2010. 85
- [178] Sebastian Stock, Masoumeh Mansouri, Federico Pecora, and Joachim Hertzberg. Online task merging with a hierarchical hybrid task planner for mobile service robots. In 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 6459–6464. IEEE, 2015. 85
- [179] José Carlos González, José Carlos Pulido, and Fernando Fernández. A three-layer planning architecture for the autonomous control of rehabilitation therapies based on social robots. *Cognitive Systems Research*, 43:232–249, 2017. 85
- [180] Michael Beetz, Dominik Jain, Lorenz Mosenlechner, Moritz Tenorth, Lars Kunze, Nico Blodow, and Dejan Pangercic. Cognition-enabled autonomous robot control for the realization of home chore task intelligence. *Proceedings of the IEEE*, 100(8):2454–2471, 2012. 85
- [181] George Konidaris, Scott Kuindersma, Roderic Grupen, and Andrew Barto. Robot learning from demonstration by constructing skill trees. The International Journal of Robotics Research, 31(3):360– 375, 2012. 85
- [182] Pascal Bercher, Ron Alford, and Daniel Höller. A survey on hierarchical planning-one abstract idea, many concrete realizations. In *IJCAI*, pages 6267–6275, 2019. 85
- [183] Or Rivlin, Tamir Hazan, and Erez Karpas. Generalized planning with deep reinforcement learning. arXiv preprint arXiv:2005.02305, 2020. 85, 89, 90, 91, 146, 147, 148
- [184] Rushang Karia and Siddharth Srivastava. Learning generalized relational heuristic networks for modelagnostic planning. arXiv preprint arXiv:2007.06702, 2020. 85
- [185] Daniel Gnad, Alvaro Torralba, Martín Domínguez, Carlos Areces, and Facundo Bustos. Learning how to ground a plan-partial grounding in classical planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7602–7609, 2019. 85
- [186] Junhyuk Oh, Satinder Singh, Honglak Lee, and Pushmeet Kohli. Zero-shot task generalization with multi-task deep reinforcement learning. In *International Conference on Machine Learning*, pages 2661–2670. PMLR, 2017. 85
- [187] Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In International Conference on Machine Learning, pages 166–175. PMLR, 2017. 85
- [188] Misha Denil, Sergio Gómez Colmenarejo, Serkan Cabi, David Saxton, and Nando de Freitas. Programmable agents. arXiv preprint arXiv:1706.06383, 2017. 85
- [189] Shao-Hua Sun, Te-Lin Wu, and Joseph J Lim. Program guided agent. In International Conference on Learning Representations, 2019. 85
- [190] Yichen Yang, Jeevana Priya Inala, Osbert Bastani, Yewen Pu, Armando Solar-Lezama, and Martin Rinard. Program synthesis guided reinforcement learning. arXiv preprint arXiv:2102.11137, 2021. 85
- [191] Peter Karkus, David Hsu, and Wee Sun Lee. Qmdp-net: Deep learning for planning under partial observability. arXiv preprint arXiv:1703.06692, 2017. 85
- [192] Peter Karkus, David Hsu, and Wee Sun Lee. Integrating algorithmic planning and deep learning for partially observable navigation. arXiv preprint arXiv:1807.06696, 2018. 85
- [193] Arthur Guez, Mehdi Mirza, Karol Gregor, Rishabh Kabra, Sébastien Racanière, Théophane Weber, David Raposo, Adam Santoro, Laurent Orseau, Tom Eccles, et al. An investigation of model-free planning. In *International Conference on Machine Learning*, pages 2464–2473. PMLR, 2019. 85
- [194] De-An Huang, Suraj Nair, Danfei Xu, Yuke Zhu, Animesh Garg, Li Fei-Fei, Silvio Savarese, and Juan Carlos Niebles. Neural task graphs: Generalizing to unseen tasks from a single video demonstration.

In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 8565–8574, 2019. 85

- [195] Danfei Xu, Suraj Nair, Yuke Zhu, Julian Gao, Animesh Garg, Li Fei-Fei, and Silvio Savarese. Neural task programming: Learning to generalize across hierarchical tasks. In 2018 IEEE International Conference on Robotics and Automation (ICRA), pages 3795–3802. IEEE, 2018. 85
- [196] Yelong Shen, Jianshu Chen, Po-Sen Huang, Yuqing Guo, and Jianfeng Gao. M-walk: Learning to walk over graphs using monte carlo tree search. arXiv preprint arXiv:1802.04394, 2018. 85
- [197] Daniel Fišer. Lifted fact-alternating mutex groups and pruned grounding of classical planning problems. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 34, pages 9835–9842, 2020. 85
- [198] Daniel Fišer, Rostislav Horčík, and Antonín Komenda. Strengthening potential heuristics with mutexes and disambiguations. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 124–133, 2020. 85
- [199] Nishanth Kumar, Michael Fishman, Natasha Danas, Michael L. Littman, Stefanie Tellex, and George Konidaris. Task scoping: Building goal-specific abstractions for planning in complex domains. CoRR, abs/2010.08869, 2020. 85
- [200] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. arXiv preprint arXiv:1806.01261, 2018. 85
- [201] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. In *International Conference on Machine Learning*, pages 2688–2697. PMLR, 2018. 85
- [202] Thomas Kipf, Elise van der Pol, and Max Welling. Contrastive learning of structured world models. arXiv preprint arXiv:1911.12247, 2019. 85
- [203] Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. arXiv preprint arXiv:2006.15055, 2020. 85
- [204] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. Ai2-thor: An interactive 3d environment for visual ai. arXiv preprint arXiv:1712.05474, 2017. 86, 87
- [205] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8494–8502, 2018. 86
- [206] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied ai research. In Proceedings of Computer Vision and Pattern Recognition (CVPR), 2019. 86, 87
- [207] Luca Weihs, Matt Deitke, Aniruddha Kembhavi, and Roozbeh Mottaghi. Visual room rearrangement. arXiv preprint arXiv:2103.16544, 2021. 86
- [208] Chuang Gan, Siyuan Zhou, Jeremy Schwartz, Seth Alter, Abhishek Bhandwaldar, Dan Gutfreund, Daniel LK Yamins, James J DiCarlo, Josh McDermott, Antonio Torralba, et al. The threedworld transport challenge: A visually guided task-and-motion planning benchmark for physically realistic embodied ai. arXiv preprint arXiv:2103.14025, 2021. 86
- [209] Bokui Shen, Fei Xia, Chengshu Li, Roberto Martín-Martín, Linxi Fan, Guanzhi Wang, Shyamal Buch, Claudia D'Arpino, Sanjana Srivastava, Lyne P Tchapmi, et al. igibson, a simulation environment for

interactive tasks in large realistic scenes. *IEEE International Conference on Robotics and Automation* (*ICRA*), 2020. 86

- [210] Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, et al. Sapien: A simulated part-based interactive environment. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, 2020. 86, 87
- [211] Yu-qian Jiang, Shi-qi Zhang, Piyush Khandelwal, and Peter Stone. Task planning in robotics: an empirical comparison of pddl- and asp-based systems. Frontiers of Information Technology & Electronic Engineering, 20(3):363–373, 2019. 86
- [212] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. arXiv preprint arXiv:2010.03768, 2020. 87
- [213] Chuang Gan, Jeremy Schwartz, Seth Alter, Martin Schrimpf, James Traer, Julian De Freitas, Jonas Kubilius, Abhishek Bhandwaldar, Nick Haber, Megumi Sano, et al. Threedworld: A platform for interactive multi-modal physical simulation. arXiv preprint arXiv:2007.04954, 2020. 87
- [214] Fei Xia, Amir R Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world perception for embodied agents. In *Proceedings of Computer Vision and Pattern Recognition* (CVPR), 2018. 16, 21, 87, 88, 141, 142
- [215] Nir Lipovetzky and Hector Geffner. Best-first width search: Exploration and exploitation in classical planning. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 31, 2017. 88
- [216] Dhruv Batra, Angel X Chang, Sonia Chernova, Andrew J Davison, Jia Deng, Vladlen Koltun, Sergey Levine, Jitendra Malik, Igor Mordatch, Roozbeh Mottaghi, et al. Rearrangement: A challenge for embodied ai. arXiv preprint arXiv:2011.01975, 2020. 89, 141
- [217] Planutils. https://github.com/AI-Planning/planutils. Accessed: June 18 2021. 89
- [218] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. Pddl-the planning domain definition language. 1998. 89, 143
- [219] Nicholas Watters, Daniel Zoran, Theophane Weber, Peter Battaglia, Razvan Pascanu, and Andrea Tacchetti. Visual interaction networks: Learning a physics simulator from video. In Neural Information Processing Systems (Neurips), 2017. 102, 114
- [220] Zhenjia Xu, Jiajun Wu, Andy Zeng, Joshua B. Tenenbaum, and Shuran Song. Densephysnet: Learning dense physical object representations via multi-step dynamic interactions. In *Robotics Science and Systems (R-SS)*, 2019. 102, 107, 114
- [221] Michael Janner, Sergey Levine, William T. Freeman, Joshua B. Tenenbaum, Chelsea Finn, and Jiajun Wu. Reasoning about physical interactions with object-oriented prediction and planning. *International Conference on Learning Representations (ICLR)*, 2019. 102, 114
- [222] Michael B. Chang, Tomer Ullman, Antonio Torralba, and Joshua B. Tenenbaum. A compositional objectbased approach to learning physical dynamics. *International Conference on Learning Representations* (ICLR), 2016. 102, 114
- [223] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets. Neural Information Processing Systems (Neurips), 2016. 102, 114
- [224] Sam Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. In Neural Information Processing Systems (Neurips), 2019. 102, 114, 153
- [225] Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian neural networks. In *ICLR Workshops*, 2020. 102, 114, 153

- [226] Viraj Mehta, Ian Char, Willie Neiswanger, Youngseog Chung, Andrew Oakleigh Nelson, Mark D Boyer, Egemen Kolemen, and Jeff Schneider. Neural dynamical systems: Balancing structure and flexibility in physical prediction. *ICLR Workshops*, 2020. 102, 119
- [227] Jonas Degrave, Michiel Hermans, Joni Dambre, and Francis Wyffels. A differentiable physics engine for deep learning in robotics. Neural Information Processing Systems (Neurips), 2016. 102, 104, 113, 153
- [228] Junbang Liang, Ming Lin, and Vladlen Koltun. Differentiable cloth simulation for inverse problems. In Neural Information Processing Systems (Neurips), 2019. 102, 113
- [229] Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C Lin. Scalable differentiable physics for learning and control. International Conference on Machine Learning (ICML), 2020. 102, 104
- [230] Matt Pharr, Wenzel Jakob, and Greg Humphreys. Physically Based Rendering: From Theory to Implementation. Morgan Kaufmann Publishers Inc., 2016. 102, 112
- [231] Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. Unified particle physics for real-time applications. ACM Transactions on Graphics (TOG), 33(4):1–12, 2014. 102
- [232] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4):229–256, 1992. 102, 108, 158
- [233] Fabio Ramos, Rafael Carvalhaes Possas, and Dieter Fox. Bayessim: adaptive domain randomization via probabilistic inference for robotics simulators. *Robotics Science and Systems (R-SS)*, 2019. 103
- [234] Kyle Cranmer, Johann Brehmer, and Gilles Louppe. The frontier of simulation-based inference. In National Academy of Sciences (NAS), 2020. 103
- [235] T. D. Kulkarni, P. Kohli, J. B. Tenenbaum, and V. Mansinghka. Picture: A probabilistic programming language for scene perception. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, 2015. 103
- [236] Ilker Yildirim, Michael Janner, Mario Belledonne, Christian Wallraven, W. A. Freiwald, and Joshua B. Tenenbaum. Causal and compositional generative models in online perception. In *CogSci*, 2017. 103
- [237] Ilker Yildirim, Tejas Kulkarni, Winrich Freiwald, and Joshua Tenenbaum. Efficient analysis-by-synthesis in vision: A computational framework, behavioral tests, and comparison with neural representations. In CoqSci, 2015. 103
- [238] Ilker Yildirim, Mario Belledonne, Winrich Freiwald, and Josh Tenenbaum. Efficient inverse graphics in biological face processing. *Science Advances*, 6(10), 2020. 103
- [239] Jiajun Wu, Joshua B Tenenbaum, and Pushmeet Kohli. Neural scene de-rendering. In Proceedings of Computer Vision and Pattern Recognition (CVPR), 2017. 103
- [240] Andreas Griewank and Andrea Walther. Introduction to automatic differentiation. PAMM, 2(1):45–49, 2003. 103
- [241] Marc Toussaint, Kelsey Allen, Kevin Smith, and Joshua Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. In *Robotics Science and Systems (R-SS)*, 2018. 104, 113
- [242] Filipe de Avila Belbute-Peres, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J. Zico Kolter. End-toend differentiable physics for learning and control. In *Neural Information Processing Systems (Neurips)*, 2018. 104, 113, 153, 160
- [243] Changkyu Song and Abdeslam Boularias. Learning to slide unknown objects with differentiable physics simulations. In *Robotics Science and Systems (R-SS)*, 2020. 104, 113
- [244] Changkyu Song and Abdeslam Boularias. Identifying mechanical models through differentiable simulations. In Learning for Dynamical Systems and Control (L4DC), 2020. 104, 113
- [245] Jiajun Wu, Erika Lu, Pushmeet Kohli, William T Freeman, and Joshua B Tenenbaum. Learning to see physics via visual de-animation. In *Neural Information Processing Systems (Neurips)*, 2017. 104, 113

- [246] Google Research. Tiny Differentiable Simulator, 2020 (accessed May 15, 2020). 104, 113
- [247] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In Neural Information Processing Systems (Neurips), 2018. 104, 149
- [248] Bart van Merriënboer, Alexander B Wiltschko, and Dan Moldovan. Tangent: automatic differentiation using source code transformation in python. In *Neural Information Processing Systems (Neurips)*, 2018. 105, 149
- [249] Kiana Ehsani, Shubham Tulsiani, Saurabh Gupta, Ali Farhadi, and Abhinav Gupta. Use the Force, Luke! learning to predict physical forces by simulating effects. In Proceedings of Computer Vision and Pattern Recognition (CVPR), 2020. 107, 108, 114, 158
- [250] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. ShapeNet: An information-rich 3d model repository. arXiv preprint arXiv:1512.03012, 2015. 107, 112
- [251] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. In *Robotics Science and Systems (R-SS)*, 2017. 107
- [252] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In International Conference on Machine Learning (ICML), 2019. 107, 159
- [253] Jiajun Wu, Ilker Yildirim, Joseph J Lim, William T Freeman, and Joshua B Tenenbaum. Galileo: Perceiving physical object properties by integrating a physics engine with deep learning. In *Neural Information Processing Systems (Neurips)*, 2015. 107, 158
- [254] Erwin Coumans and Yunfei Bai. PyBullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016-2019. 108, 158
- [255] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In International Conference on Learning Representations (ICLR), 2015. 110, 159
- [256] Vincent Le Guen and Nicolas Thome. Disentangling physical dynamics from unknown factors for unsupervised video prediction. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, 2020. 113
- [257] Connor Schenck and Dieter Fox. Spnets: Differentiable fluid dynamics for deep neural networks. In International Conference on Robot Learning (CoRL), 2018. 113
- [258] Miguel Jaques, Michael Burke, and Timothy M. Hospedales. Physics-as-inverse-graphics: Joint unsupervised learning of objects and physics from video. International Conference on Learning Representations (ICLR), 2020. 113
- [259] Eric Heiden, David Millard, Hejia Zhang, and Gaurav S. Sukhatme. Interactive differentiable simulation. In arXiv, 2019. 113
- [260] Yuanming Hu, Jiancheng Liu, Andrew Spielberg, Joshua B. Tenenbaum, William T. Freeman, Jiajun Wu, Daniela Rus, and Wojciech Matusik. Chainqueen: A real-time differentiable physical simulator for soft robotics. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2019. 113, 150
- [261] Yuanming Hu, Yu Fang, Ziheng Ge, Ziyin Qu, Yixin Zhu, Andre Pradhana, and Chenfanfu Jiang. A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. ACM Transactions on Graphics, 37(4), 2018. 113
- [262] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *International Conference on Learning Representations (ICLR)*, 2019. 113

- [263] Yunzhu Li, Toru Lin, Kexin Yi, Daniel Bear, Daniel L. K. Yamins, Jiajun Wu, Joshua B. Tenenbaum, and Antonio Torralba. Visual grounding of learned physical models. In *International Conference on Machine Learning (ICML)*, 2020. 113
- [264] Zhengdao Chen, Jianyu Zhang, Martin Arjovsky, and Léon Bottou. Symplectic recurrent neural networks. In International Conference on Learning Representations (ICLR), 2020. 114
- [265] Peter Toth, Danilo Jimenez Rezende, Andrew Jaegle, Sébastien Racanière, Aleksandar Botev, and Irina Higgins. Hamiltonian generative networks. In International Conference on Learning Representations (ICLR), 2020. 114, 153
- [266] Alvaro Sanchez-Gonzalez, Victor Bapst, Kyle Cranmer, and Peter Battaglia. Hamiltonian graph networks with ode integrators. In arXiv, 2019. 114, 153
- [267] Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Symplectic ode-net: Learning hamiltonian dynamics with control. In International Conference on Learning Representations (ICLR), 2020. 114
- [268] Martin Asenov, Michael Burke, Daniel Angelov, Todor Davchev, Kartic Subr, and Subramanian Ramamoorthy. Vid2Param: Modelling of dynamics parameters from video. *IEEE Robotics and Automation Letters (RAL)*, 2019. 114
- [269] Jiajun Wu, Joseph J Lim, Hongyi Zhang, Joshua B Tenenbaum, and William T Freeman. Physics 101: Learning physical object properties from unlabeled videos. In *British Machine Vision Conference* (*BMVC*), 2016. 114
- [270] Katerina Fragkiadaki, Pulkit Agrawal, Sergey Levine, and Jitendra Malik. Learning visual predictive models of physics for playing billiards. In *International Conference on Learning Representations (ICLR)*, 2016. 114
- [271] Arunkumar Byravan and Dieter Fox. SE3-Nets: Learning rigid body motion using deep neural networks. IEEE International Conference on Robotics and Automation (ICRA), 2017. 114
- [272] Roozbeh Mottaghi, Hessam Bagherinezhad, Mohammad Rastegari, and Ali Farhadi. Newtonian image understanding: Unfolding the dynamics of objects in static images. Proceedings of Computer Vision and Pattern Recognition (CVPR), 2015. 114
- [273] Roozbeh Mottaghi, Mohammad Rastegari, Abhinav Gupta, and Ali Farhadi. "what happens if..." learning to predict the effect of forces in images. In Proceedings of the European Conference on Computer Vision (ECCV), 2016. 114
- [274] Abhinav Gupta, Alexei A. Efros, and Martial Hebert. Blocks world revisited: Image understanding using qualitative geometry and mechanics. In *Proceedings of the European Conference on Computer* Vision (ECCV), 2010. 114
- [275] Sébastien Ehrhardt, Aron Monszpart, Niloy J. Mitra, and Andrea Vedaldi. Unsupervised intuitive physics from visual observations. Asian Conference on Computer Vision (ACCV), 2018. 114
- [276] L. Yu, N. Duncan, and S. Yeung. Fill and transfer: A simple physics-based approach for containability reasoning. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2015. 114
- [277] Peter W. Battaglia, Jessica B. Hamrick, and Joshua B. Tenenbaum. Simulation as an engine of physical scene understanding. *Proceedings of the National Academy of Sciences*, 110(45):18327–18332, 2013. 114
- [278] Richard Mann, Allan Jepson, and Jeffrey Mark Siskind. The computational perception of scene dynamics. Computer Vision and Image Understanding, 65(2):113 – 128, 1997. 114
- [279] Carlo Innamorati, Bryan Russell, Danny Kaufman, and Niloy Mitra. Neural re-simulation for generating bounces in single images. In Proceedings of International Conference on Computer Vision (ICCV), 2019. 114

- [280] Trevor Standley, Ozan Sener, Dawn Chen, and Silvio Savarese. image2mass: Estimating the mass of an object from its image. In *International Conference on Robot Learning (CoRL)*, 2017. 114
- [281] Sébastien Ehrhardt, Aron Monszpart, Niloy J. Mitra, and Andrea Vedaldi. Learning a physical long-term predictor. arXiv, 2017. 114
- [282] Pulkit Agrawal, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. Neural Information Processing Systems (Neurips), 2016. 114
- [283] Mathieu Salzmann and Raquel Urtasun. Physically-based motion models for 3d tracking: A convex formulation. In Proceedings of International Conference on Computer Vision (ICCV), 2011. 114
- [284] Marcus Brubaker, David Fleet, and Aaron Hertzmann. Physics-based person tracking using the anthropomorphic walker. International Journal of Computer Vision (IJCV), 87:140–155, 03 2010. 114
- [285] Krzysztof Kozlowski. Modelling and Identification in Robotics. Advances in Industrial Control. Springer, London, 1998. 114
- [286] P. M. Wensing, S. Kim, and J. E. Slotine. Linear matrix inequalities for physically consistent inertial parameter identification: A statistical perspective on the mass distribution. *IEEE Robotics and Automation Letters*, 3(1):60–67, 2018. 114
- [287] Marcus A Brubaker, Leonid Sigal, and David J Fleet. Estimating contact dynamics. In Proceedings of International Conference on Computer Vision (ICCV), 2009. 114
- [288] Kiran S Bhat, Christopher D Twigg, Jessica K Hodgins, Pradeep Khosla, Zoran Popovic, and Steven M Seitz. Estimating cloth simulation parameters from video. In ACM SIGGRAPH/Eurographics Symposium on Computer Animation, 2003. 114
- [289] Kiran S Bhat, Steven M Seitz, Jovan Popović, and Pradeep K Khosla. Computing the physical parameters of rigid-body motion from video. In *Proceedings of the European Conference on Computer* Vision (ECCV), 2002. 114
- [290] C Karen Liu, Aaron Hertzmann, and Zoran Popović. Learning physics-based motion style with nonlinear inverse optimization. ACM Transactions on Graphics (TOG), 24(3):1071–1081, 2005. 114
- [291] Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. Neuroanimator: Fast neural network emulation and control of physics-based models. In Proceedings of the 25th annual conference on Computer graphics and interactive techniques, 1998. 114
- [292] Giovanni Sutanto, Austin S. Wang, Yixin Lin, Mustafa Mukadam, Gaurav S. Sukhatme, Akshara Rai, and Franziska Meier. Encoding physical constraints in differentiable newton-euler algorithm. In Learning for Dynamical systems and Control (L4DC), 2020. 114
- [293] Kun Wang, Mridul Aanjaneya, and Kostas Bekris. A first principles approach for data-efficient system identification of spring-rod systems via differentiable physics engines. In arXiv, 2020. 114
- [294] Bin Wang, Paul G. Kry, Yuanmin Deng, Uri M. Ascher, Hui Huang, and Baoquan Chen. Neural material: Learning elastic constitutive material and damping models from sparse data. arXiv, 2018. 114
- [295] D.J. Murray-Smith. The inverse simulation approach: a focused review of methods and applications. Mathematics and Computers in Simulation, 53(4):239 – 247, 2000. 114
- [296] Philipp Henzler, Niloy J. Mitra, and Tobias Ritschel. Escaping plato's cave using adversarial training: 3d shape from unstructured 2d image collections. In *Proceedings of International Conference on Computer* Vision (ICCV), 2019. 114
- [297] Despoina Paschalidou, Ali Osman Ulusoy, Carolin Schmitt, Luc van Gool, and Andreas Geiger. Raynet: Learning volumetric 3d reconstruction with ray potentials. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, 2019. 114

- [298] Edward Smith, Scott Fujimoto, and David Meger. Multi-view silhouette and depth decomposition for high resolution 3d object representation. In Neural Information Processing Systems (Neurips), 2018.
 114
- [299] Thu Nguyen-Phuoc, Chuan Li, Stephen Balaban, and Yong-Liang Yang. Rendernet: A deep convolutional network for differentiable rendering from 3d shapes. Neural Information Processing Systems (Neurips), 2018. 114
- [300] Jun Gao, Wenzheng Chen, Tommy Xiang, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Learning deformable tetrahedral meshes for 3d reconstruction. In *Neural Information Processing Systems* (Neurips), 2020. 114
- [301] Edward J. Smith, Scott Fujimoto, Adriana Romero, and David Meger. Geometrics: Exploiting geometric structure for graph-encoded objects. *International Conference on Machine Learning (ICML)*, 2020. 114
- [302] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In Proceedings of the European Conference on Computer Vision (ECCV), 2018. 114
- [303] Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan C. Russell, and Mathieu Aubry. Atlasnet: A papier-mâché approach to learning 3d surface generation. In Proceedings of Computer Vision and Pattern Recognition (CVPR), 2018. 114
- [304] Hassan Abu Alhaija, Siva Karthik Mustikovela, Andreas Geiger, and Carsten Rother. Geometric image synthesis. In Proceedings of Computer Vision and Pattern Recognition (CVPR), 2018. 114
- [305] Yuxuan Zhang, Wenzheng Chen, Huan Ling, Jun Gao, Yinan Zhang, Antonio Torralba, and Sanja Fidler. Image gans meet differentiable rendering for inverse graphics and interpretable 3d neural rendering. In International Conference on Learning Representations (ICLR), 2021. 114
- [306] Qiangeng Xu, Weiyue Wang, Duygu Ceylan, Radomir Mech, and Ulrich Neumann. Disn: Deep implicit surface network for high-quality single-view 3d reconstruction. In *Neural Information Processing Systems* (Neurips), 2019. 114
- [307] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. Proceedings of Computer Vision and Pattern Recognition (CVPR), 2019. 114
- [308] Mateusz Michalkiewicz, Jhony K. Pontes, Dominic Jack, Mahsa Baktashmotlagh, and Anders Eriksson. Implicit surface representations as layers in neural networks. In *Proceedings of International Conference* on Computer Vision (ICCV), 2019. 114
- [309] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In Proceedings of Computer Vision and Pattern Recognition (CVPR), 2020. 114
- [310] Jeong Joon Park, Peter Florence, Julian Straub, Richard A. Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In Proceedings of Computer Vision and Pattern Recognition (CVPR), 2019. 114
- [311] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, 2019. 114
- [312] Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Neural geometric level of detail: Real-time rendering with implicit 3D shapes. Proceedings of Computer Vision and Pattern Recognition (CVPR), 2021. 114

- [313] Yiyi Liao, Katja Schwarz, Lars Mescheder, and Andreas Geiger. Towards unsupervised learning of generative models for 3d controllable image synthesis. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, 2019. 114
- [314] S. M. Ali Eslami, Danilo Jimenez Rezende, Frederic Besse, Fabio Viola, Ari S. Morcos, Marta Garnelo, Avraham Ruderman, Andrei A. Rusu, Ivo Danihelka, Karol Gregor, David P. Reichert, Lars Buesing, Theophane Weber, Oriol Vinyals, Dan Rosenbaum, Neil Rabinowitz, Helen King, Chloe Hillier, Matt Botvinick, Daan Wierstra, Koray Kavukcuoglu, and Demis Hassabis. Neural scene representation and rendering. *Science*, 2018. 114
- [315] 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 Proceedings of the European Conference on Computer Vision (ECCV), 2020. 114
- [316] Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. ACM Transactions on Graphics (TOG), 2019. 114
- [317] Krishna Murthy Jatavallabhula, Edward Smith, Jean-Francois Lafleche, Clement Fuji Tsang, Artem Rozantsev, Wenzheng Chen, Tommy Xiang, Rev Lebaredian, and Sanja Fidler. Kaolin: A pytorch library for accelerating 3d deep learning research. In arXiv, 2019. 114
- [318] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing arXiv:1801.09847, 2018. 114
- [319] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. arXiv preprint arXiv:2007.08501, 2020. 114
- [320] Paloma Sodhi, Eric Dexheimer, Mustafa Mukadam, Stuart Anderson, and Michael Kaess. LEO: Learning energy-based models in factor graph optimization. In *Conference on Robot Learning (CoRL)*, 2021. 118
- [321] Brent Yi, Michelle Lee, Alina Kloss, Roberto Martín-Martín, and Jeannette Bohg. Differentiable factor graph optimization for learning smoothers. arXiv preprint arXiv:2105.08257, 2021. 118
- [322] Peter Karkus, Shaojun Cai, and David Hsu. Differentiable slam-net: Learning particle slam for visual navigation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 2815–2825, 2021. 118
- [323] Anonymous. Risp: Rendering-invariant state predictor with differentiable simulation and rendering for cross-domain parameter estimation. *ICLR*, 2022. 118
- [324] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. arXiv preprint arXiv:2005.14165, 2020. 118
- [325] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. arXiv preprint arXiv:2103.00020, 2021. 118
- [326] David Wingate and Theophane Weber. Automated variational inference in probabilistic programming. arXiv preprint arXiv:1301.1299, 2013. 119
- [327] Eli Bingham, Jonathan P Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D Goodman. Pyro: Deep universal probabilistic programming. *The Journal of Machine Learning Research*, 20(1):973–978, 2019. 119
- [328] Marco F Cusumano-Towner, Feras A Saad, Alexander K Lew, and Vikash K Mansinghka. Gen: a general-purpose probabilistic programming system with programmable inference. In *Proceedings of the*

40th acm sigplan conference on programming language design and implementation, pages 221–236, 2019. 119

- [329] Andy Zeng, Shuran Song, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. *IEEE Transactions on Robotics*, 36(4):1307– 1319, 2020. 119
- [330] David Kirk et al. Nvidia cuda software and gpu parallel computing architecture. In *ISMM*, volume 7, pages 103–104, 2007. 149
- [331] Tom Erez, Yuval Tassa, and Emanuel Todorov. Simulation tools for model-based robotics: Comparison of Bullet, Havok, MuJoCo, ODE, and PhysX. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2015. 150, 151, 153
- [332] Eftychios Sifakis and Jernej Barbic. Fem simulation of 3d deformable solids: a practitioner's guide to theory, discretization and model reduction. In ACM SIGGRAPH 2012 courses, 2012. 150
- [333] Breannan Smith, Fernando De Goes, and Theodore Kim. Stable neo-hookean flesh simulation. ACM Transactions on Graphics, 37(2):1–15, 2018. 150, 152
- [334] Robert Bridson, Sebastian Marino, and Ronald Fedkiw. Simulation of clothing with folds and wrinkles. In ACM SIGGRAPH 2005 Courses, 2005. 150
- [335] Emanuel Todorov. Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in mujoco. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2014. 151
- [336] Jos Stam. Stable fluids. In Proceedings of the 26th annual conference on Computer graphics and interactive techniques, pages 121–128, 1999. 151, 154
- [337] Ernst Hairer, Christian Lubich, and Gerhard Wanner. Geometric numerical integration: structurepreserving algorithms for ordinary differential equations, volume 31. Springer Science & Business Media, 2006. 151
- [338] Dougal Maclaurin, David Duvenaud, Matt Johnson, and Jamie Townsend. Autograd, 2015. 154
- [339] Charles C Margossian. A review of automatic differentiation and its efficient implementation. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 9(4):e1305, 2019. 154
- [340] Danilo Jimenez Rezende, SM Ali Eslami, Shakir Mohamed, Peter Battaglia, Max Jaderberg, and Nicolas Heess. Unsupervised learning of 3d structure from images. In Neural Information Processing Systems (Neurips), 2016. 158

Taskography: Supplementary material

This supplementary material discusses additional details and design choices for the TASKOG-RAPHY benchmark, including extended descriptions for all supported planning domains and their constituents - object types, relations (i.e., properties, predicates), and goal specifications. We provide results these additional domains, and discuss SCRUB and its favourable properties in greater detail.

Please visit our project page for more details, including a video abstract.

A.1. Benchmark Details

The TASKOGRAPHY benchmark comprises 20 robot task planning domains over 3D scene graphs (3DSGs). In the main paper, we detailed the Rearrangement(k), Courier(n,k), Lifted Rearrangement(k), and Lifted Courier(n, k) task definitions following the recently proposed Rearrangement challenge [216]. Table. A.1.1 lists the set of *lifted* objects in each planning domain. In all problems, we have one instance of an *agent*, but several ground objects corresponding to the other categories.

A.1.1. Taskography domain construction: Parsing Gibson 3DSGs

We parse the 3DSGs created over Gibson [161, 214] mapping scene entities to objects and structural relations to predicates over objects. We retain key connectivity constraints that govern traversable paths between locations in the same place, places in the same room, and between rooms. Because room connectivity data not is provided in the original database, we estimate it by computing a minimal spanning tree over rooms in the 3DSGs with edge weights reflecting the Euclidean distance between room centroids. For larger scenes, we impose a single connection between rooms in different floors (e.g., one set of stairs). Several additional properties are used to express the state of agent and interactable objects, and to associate each of them to a particular location in the 3DSG.

Table A.1.1. Evaluated 3DSG planning domains in TASKOGRAPHY and object types present in each. Domains are further partitioned into tiny and medium splits akin to the 3DSGs provided over Gibson [161, 214]. Scene entities are instantiated as a particular object type according to their semantic class.

	n	k	Agent	Room	Place	Location	Receptacle	Item	Bagslot	Receptacle Class	Item Class
Rearr(k)	-	$\{1, 2, 5, 10\}$	1	1	1	1	1	1	X	X	x
Cour(n, k)	$\{3, 5, 7, 10\}$	$\{5, 10\}$	1	1	1	1	1	1	1	X	×
Lifted Rearr(k)	-	{5}	1	1	1	1	1	1	X	1	1
Lifted Cour(n, k)	$\{5\}$	{5}	1	1	1	1	1	1	1	1	1

An assignment of values to all possible properties over objects defines a symbolic *state* in the planning problem; hence, actions taken by the robot in TASKOGRAPHY alter the symbolic state of the 3DSG. We observe a significant variation in the size of the state space between different types of domains as a result of the varying subsets of object and predicate types used to express their respective tasks (see Table. A.1.2). For instance, the *Rearrangement(k)* task represents the lowest complexity domain on TASKOGRAPHY and is thereby defined by the smallest subset of object types, predicates, and actions available to the robot. In contrast, the *Lifted Courier(n, k)* extends the *Rearrangement(k)* task definition with bagslots enabling stow and retrieve operators, as well as receptacle classes and item classes to express lifted class relations in the 3DSG at particular state.

We leverage **task samplers** built into TASKOGRAPHY-API for generating large-scale and diverse datasets of planning problems over 3DSGs. In a two step process the task samplers automatically parse 3DSGs into plannable symbolic representations (i.e., embedding the agent forms the initial state \mathcal{I}) before composing goal literals over randomly sampled scene entities. For grounded problems, goals are conjunctions of inReceptacle literals expressed over randomly sampled item and receptacle target ground instances. For lifted problems, goal are conjunctions of classRelation literals expressed over randomly sampled item and receptacle target class relations.

Table A.1.2. Structural relations of 3DSGs and the state of the robot and interactable objects (i.e., items and receptacles) are captured with an assignment of values to all possible predicates over objects. The most challenging *Lifted Courier*(n, k) is the only domain to incorporate all relations, while other domain types in TASKOGRAPHY require only a subset of the properties and relations.

Object (:types)	Agent	Room	Place	Location	Receptacle	Item	Bagslot	Receptacle Class	Item Class
Agent	holdsAny	inRoom	inPlace	atLoc	-	holdsItem	-	-	-
Room	inRoom	connected	placeInRoom + roomCenter	-	-	-	-	-	-
Place	inPlace	placeInRoom + roomCenter	-	locInPlace + placeCenter	-	-	-	-	-
Location	atLoc	-	locInPlace + placeCenter	-	recepAtLoc	itemAtLoc	-	-	-
Receptacle	-	-	-	recepAtLoc	recepOpened	inRecep	-	recepClass	-
Item	holdsItem	-	-	itemAtLoc	inRecep	small + medium + large	inSlot		itemClass
Bagslot	-	-	-	-	-	inSlot	slotHoldsAny	-	-
Receptacle Class		-	-	-	recepClass	-	-	-	classRelation
Item Class	-	-	-		-	itemClass	-	classRelation	-

A.1.2. Domain specifications

To provide further clarity on the four task categories (*Rearrangement(k*), *Courier(n,k)*, *Lifted Rearrangement(k*), and *Lifted Courier(n, k)*) from which our 3DSG planning domains are constructed, we herein outline hypothetical problem instances involving but a fraction of the objects, attributes, and relations available in TASKOGRAPHY. Let the environment consist of *v* rooms connected by *e* undirected traversability constraints; e.g., connected(roomA, roomB). The spatial hierarchy of 3DSGs [161, 162] is induced by the appropriate application of structural relations (see Table. A.1.2) to a discrete set of places in each room, and locations in each place; e.g., placeInRoom(placeD, roomC), locInPlace(locF, placeD). The lowest level of the spatial hierarchy (locations) encodes all occupiable positions for the agent, items, and receptacles in the scene; e.g., atLoc(agent, locationB), itemAtLoc(mugA, locationD), recepAtLoc(fridgeC, locationG). Such relations equate to logical predicates in [218] and can be altered by the agent should the required preconditions of an action be met in the current state; e.g., \neg holdsAny(agent) and $\land(atLoc(agent, locX), itemAtLoc(mugA, locX))$ are preconditions for PICKUPITEM(mugA, agent).

As mentioned in Sec. A.1.1, the goals in grounded planning problems are specified with inReceptacle literals. Concretely, a Rearrangement(k) task for k = 1 requires the agent to pick-and-place a ground item in a ground receptacle, where each object in the goal is uniquely identified; e.g., G = inReceptacle(mugA, fridgeC). By extension, a Rearrangement(k) task for k = 2 is solved *iff* the agent derives a state satisfying the conjunction of two inReceptacle goal literals; e.g., $G = \wedge(inReceptacle(mugA, fridgeC), inReceptacle(plateD, shelfB))$. The Courier(n, k) domains attribute weights ($w \in 1, 2, 3$ units) to items based on their volume, and equips the agent with a knapsack of fixed capacity n to stow and retrieve items as it traverses the scene. While the knapsack in Courier(n, k) enables planners to exploit stowing capacity to compute lower cost solutions (at the expense of task complexity) in comparison to Rearrangement(k), goals are identically specified between the two task categories since they are both considered grounded.

In stark constrast, lifted planning problems are specified with classRelation literals expressed over item-receptacle class combinations. For instance, the following Lifted Rearrangement(k) or Lifted Courier(n, k) domain with k = 2, $G = \wedge (\text{classRelation(cup, cupboard), classRelation(plate, sink)})$, requires the agent to place **at least one** cup in a cupboard and plate in a sink for the task to be complete. This disambiguates the planner which is no longer able to exploit ground objects featured in the goal as heuristic landmarks, and reduces the effectiveness of deterministic graph sparsification techniques such as SCRUB. As in the grounded domain variants, the goal specifications for both the Lifted Rearrangement(k) and Lifted Courier(n, k) are identical.

A.1.3. Symbolic environment interaction

The action space of the most complex domain in TASKOGRAPHY equips the agent with 16 operators where only a subset are feasible at any given state. Below, we describe but a few of these operators which demonstrate motion through 3DSG hierachies and object-level robot interaction.

- GOTOROOM: The robot moves from the door of its current room to the door of the target room if the rooms are *connected*.
- GOTOPLACE: The robot moves from the center of its current place to the center of the target place if the places are in the same room.
- GOTOLOCATION: The robot moves from the current location to the target location if the locations are in the same place.
- OPENRECEPTACLE: The robot opens a queried *openable* receptacle.
- CLOSERECEPTACLE: The robot closes a queried *openable* receptacle.
- PICKUPITEM: The robot picks-up an item at a particular location with a free gripper; three operator variations for picking from non existent, non-opening, and opening receptacles.
- PLACEITEM: The robot places an in-gripper item at a particular location; two operator variations for placing in non-opening and opening receptacles.
- STOWITEM: The robot stows an in-gripper item in its knapsack: three operator variations for small, medium, and large items consuming increasing numbers of bagslots.
- RETRIEVEITEM: The robot retrieves an item from its knapsack into its gripper; three operator variations for small, medium, and large items freeing increasing number of bagslots.

Should the preconditions for any of these actions not be satisfied, the action is deemed invalid.

A.2. SCRUB: Discussion and analysis

In the main paper, for sake of brevity, we only discussed the applicability of SCRUB to grounded planning problems with deterministic transitions. However, by design, SCRUB may be applied to any planning problem: *lifted* or *grounded*, with *deterministic* or *stochastic* transitions.

In *lifted* planning problems, we modify SCRUB to trivially include all ground object tuples that satisfy goal conditions into the initial sufficient object set. This in-turn ensures that all of these ground objects are reachable from the start state, ensuring a satisficing plan exists. However, this conservative strategy may resulting in retaining more objects than minimally required – this is where SEEK can be applied to opportunistically retain important objects instead.
In a similar vein, for *stochastic* transitions, we modify SCRUB to include all binary predicates resulting from all possible stochastic transitions from a given node.

We now prove that SCRUB results in a minimal scene subgraph for all grounded planning problems.

Proposition 2. SCRUB is complete and results in a minimal scene subgraph for all grounded planning problems over the scenegraph domain.

PROOF. We prove the minimality of SCRUB by demonstrating that whenever we prune a node from a SCRUBBED scenegraph, the resultant planning problem is unsolvable. Assume that we prune a node n from a SCRUBBED 3DSG \hat{G} . Recall the types of nodes we have in the 3DSG: agent, room, place, receptacle, item, floor, building.

- (1) If n is of type agent or building, the problem is unsolvable, by construction.
- (2) If n is of type item, removing it would render the goal state unreachable recall that \hat{G} only retains item nodes that feature in the goal state.
- (3) If n is of type receptacle, it is retained in \hat{G} either because (a) it is required to access a goal object of type item, or (b) it is a goal receptacle (i.e., a target location an item must be moved into). Removing n will thus render one of the objects in the goal state unreachable.
- (4) If n is of type place, room or floor, $n \in \hat{G}$ because n directly features in the goal state, or because n is required to traverse from the start state to the goal state (e.g., rooms that connect the start and goal rooms, etc.).

Since pruning any of these nodes renders the problem unsolvable, the SCRUBBED graph \hat{G} is a minimal scene subgraph for the grounded planning problem considered.

A.3. Additional results on Taskography domains

In this section, we provide results over several extended domains from the TASKOGRAPHY benchmark. Please see Tables A.3.3, A.3.4, A.3.5, A.3.6, A.3.7, A.3.8.

		Rea	rr(1) T	iny	Rea	urr(2) T	iny	ny Rearr(10) Ti				
	Planner	Len.	Time	Fail	Len.	Time	Fail	Len.	Time	Fail		
	FD-seq-opt-lmcut	15.77	24.81	0.04	25.80	104.47	0.55	-	-	1.00		
lal	$\operatorname{SatPlan}$	14.77	10.35	0.45	26.67	3.27	0.67	-	-	1.00		
cim	Delfi	15.13	0.36	0.16	29.10	27.77	0.29	-	-	1.00		
opt	$\mathbf{DecStar-opt-fb}$	-	-	1.00	-	-	1.00	-	-	1.00		
	MCTS	-	-	1.00	-	-	1.00	-	-	1.00		
	FF	16.71	0.19	0.00	34.44	0.55	0.00	162.61	7.04	0.07		
<u>1</u> 0	FF-X	16.71	0.25	0.00	34.44	0.58	0.00	162.30	7.39	0.09		
ici	FD-lama-first	15.19	2.96	0.33	38.47	3.25	0.18	205.89	7.68	0.51		
isf	Cerberus-sat	11.50	12.00	0.85	-	-	1.00	-	-	1.00		
sat	Cerberus-agl	14.77	5.13	0.45	33.00	7.30	0.49	186.07	9.04	0.73		
	DecStar-agl-fb	14.72	2.62	0.55	34.96	2.58	0.58	193.00	6.78	0.85		
	BFWS	15.56	0.90	0.22	32.16	0.37	0.18	160.93	0.57	0.18		
	Regression-plan	-	-	1.00	-	-	1.00	-	-	1.00		
rn	Relational policy [183]	-	-	1.00	-	-	1.00	-	-	1.00		
lea	PLOI [36]	16.45	0.00*	0.00	37.04	0.00*	0.00	221.71	0.18	0.00		

Table A.3.3. Performance of planners over the Rearrangement(k)-Tiny tasks. For all metrics, lower values indicate better performance.

Table A.3.4. Performance of planners over the Rearrangement(k)-Medium tasks. For all metrics, lower values indicate better performance.

		Rear	r(1) Me	dium	Rearr(2) Medium			Rearr(10) Medium			
	Planner	Len.	Time	Fail	Len.	Time	Fail	Len.	Time	Fail	
	FD-seq-opt-lmcut	15.53	19.68	0.06	27.13	125.69	0.41	-	-	1.00	
ыl	SatPlan	14.98	11.98	0.33	28.23	5.45	0.50	-	-	1.00	
tin	Delfi	15.40	3.62	0.16	29.13	12.79	0.28	-	-	1.00	
opt	$\mathbf{DecStar-opt-fb}$	15.42	41.35	0.93	28.50	111.53	0.91	-	-	1.00	
	MCTS	-	-	1.00	-	-	1.00	-	-	1.00	
	FF	16.45	0.25	0.00	32.87	0.41	0.00	159.04	5.30	0.09	
<u>1</u> 0	FF-X	16.45	0.21	0.00	32.87	0.45	0.00	159.80	5.02	0.08	
icii	FD-lama-first	15.51	2.48	0.21	39.20	2.77	0.20	208.28	6.35	0.49	
isf	Cerberus-sat	11.20	10.17	0.88	-	-	1.00	-	-	1.00	
sat	Cerberus-agl	15.18	6.10	0.34	32.20	6.40	0.33	176.60	8.91	0.72	
	$\mathbf{DecStar}$ -agl-fb	15.36	2.15	0.58	36.35	2.40	0.59	211.16	7.20	0.82	
	BFWS	15.42	0.60	0.23	30.65	0.44	0.27	151.17	0.41	0.23	
	Regression-plan	-	-	1.00	-	-	1.00	-	-	1.00	
rn	Relational policy [183]	-	-	1.00	-	-	1.00	-	-	1.00	
lea	PLOI [36]	16.44	0.00*	0.00	36.19	0.00*	0.00	213.43	0.17	0.00	

Table A.3.5. Performance of planners over the Courier(n, k)-Tiny tasks. For all metrics,
lower values indicate better performance.Cour(3, 10) TinyCour(5, 10) TinyCour(7, 10) TinyCour(10, 10) TinyPlannerCour(3, 10) TinyCour(5, 10) TinyCour(7, 10) TinyCour(10, 10) TinyPlannerCour(3, 10) TinyCour(5, 10) TinyCour(10, 10) TinyPlannerCour(10, 10) TinyLen. Time FailLen. Time FailLen. Time Fail

						° (,			, .				
	Planner	Len.	Time	Fail									
isficing	FF	146.35	7.57	0.13	136.38	7.97	0.33	127.88	6.84	0.55	124.93	14.62	0.73
	FF-X	144.80	8.34	0.11	137.05	7.49	0.31	128.42	8.34	0.53	126.31	15.21	0.71
	${f FD}$ -lama-first	175.15	8.31	0.53	159.64	7.31	0.55	156.12	6.97	0.55	145.00	7.50	0.56
	Cerberus-sat	-	-	1.00	-	-	1.00	-	-	1.00	-	-	1.00
sat	Cerberus-agl	137.87	10.79	0.73	127.30	17.61	0.82	138.25	21.65	0.93	-	-	1.00
	DecStar-agl-fb	140.47	4.52	0.69	124.62	4.65	0.71	120.20	4.04	0.73	117.73	6.98	0.73
	BFWS	160.18	1.19	0.18	159.17	0.94	0.25	159.90	1.80	0.29	153.93	4.28	0.45
	Regression-plan	-	-	1.00	-	-	1.00	-	-	1.00	-	-	1.00
rn	Relational policy [183]	-	-	1.00	-	-	1.00	-	-	1.00	-	-	1.00
lea	PLOI [36]	193.55	0.22	0.00	179.36	0.26	0.00	172.87	0.37	0.00	167.38	0.71	0.00

Table A.3.6. Performance of planners over the Courier(n, k)-Medium tasks. For all metrics, lower values indicate better performance.

		Cour(3	ur(3, 10) Medium			Cour(5, 10) Medium			Cour(7, 10) Medium			Cour(10, 10) Medium		
	Planner	Len.	Time	Fail	Len.	Time	Fail	Len.	Time	Fail	Len.	Time	Fail	
	FF	141.89	4.94	0.07	133.46	6.29	0.20	128.41	6.62	0.24	117.50	14.27	0.78	
16	FF-X	141.89	4.47	0.07	133.50	5.80	0.19	128.19	6.72	0.24	118.67	15.52	0.77	
icii	FD-lama-first	180.38	7.11	0.40	166.35	6.27	0.45	156.34	4.92	0.29	141.75	6.80	0.63	
isf	Cerberus-sat	-	-	1.00	-	-	1.00	-	-	1.00	-	-	1.00	
sat	Cerberus-agl	148.41	10.17	0.74	133.31	11.50	0.77	125.73	12.99	0.83	109.56	15.58	0.95	
	DecStar-agl-fb	154.07	6.45	0.66	142.42	4.01	0.61	132.60	4.50	0.58	128.58	7.60	0.70	
	BFWS	151.09	0.60	0.27	152.61	0.66	0.20	152.71	1.13	0.21	153.02	2.81	0.30	
	Regression-plan	-	-	1.00	-	-	1.00	-	-	1.00	-	-	1.00	
rn	Relational policy [183]	-	-	1.00	-	-	1.00	-	-	1.00	-	-	1.00	
lea	PLOI [36]	182.31	0.20	0.00	169.20	0.24	0.00	161.90	0.34	0.00	152.19	0.61	0.00	

Table A.3.7. Performance of planners over the *Lifted Rearrangement(k)* domains. For all metrics, lower values indicate better performance.

		Lifted	Rearr(5, 5) Tiny	Lifted	Rearr(5, 5) Medium
	Planner	Len.	Time	Fail	Len.	Time	Fail
	FF	62.86	3.40	0.47	61.90	3.04	0.37
പ	FF-X	67.88	3.48	0.89	61.78	2.30	0.72
icir	${f FD}$ -lama-first	66.81	3.20	0.49	71.15	4.11	0.46
isf	Cerberus-sat	-	-	1.00	-	-	1.00
sat	Cerberus-agl	60.50	7.62	0.60	64.26	6.74	0.57
	$\mathbf{DecStar}$ -agl-fb	66.30	3.02	0.71	77.00	3.08	0.71
	BFWS	56.90	0.94	0.41	55.36	0.80	0.43
	Regression-plan	-	-	1.00	-	-	1.00
rn	Relational policy [183]	-	-	1.00	-	-	1.00
lea	PLOI [36]	78.68	0.22	0.24	76.62	0.22	0.24

		Lifted	Cour(5	5, 5) Tiny	Lifted	Cour(5	, 5) Medium
	Planner	Len.	Time	Fail	Len.	Time	Fail
	FF	57.74	4.03	0.44	57.38	4.81	0.37
<u>1</u> 0	FF-X	61.19	7.56	0.77	60.05	3.79	0.64
icii	${f FD}$ -lama-first	61.13	3.34	0.56	63.19	3.31	0.45
lisf	Cerberus-sat	-	-	1.00	-	-	1.00
sat	Cerberus-agl	59.19	7.05	0.77	59.61	7.45	0.68
	$\mathbf{DecStar}$ -agl-fb	58.75	4.46	0.71	63.93	3.85	0.68
	BFWS	61.92	2.30	0.43	56.14	0.67	0.38
	Regression-plan	-	-	1.00	-	-	1.00
rn	Relational policy [183]	-	-	1.00	-	-	1.00
lea	PLOI [36]	71.71	0.26	0.26	69.92	0.46	0.30

Table A.3.8. Performance of planners over the *Lifted Courier*(n, k) domains. For all metrics, lower values indicate better performance.

Appendix B

∇Sim : Supplementary material

B.1. Differentiable physics engine

Under Lagrangian mechanics, the state of a physical system can be described in terms of generalized coordinates \mathbf{q} , generalized velocities $\dot{\mathbf{q}} = \mathbf{u}$, and design, or model parameters θ . For the purposes of exposition, we make no distinction between rigid-bodies, deformable solids, or thin-shell models of cloth and other bodies. Although the specific choices of coordinates and parameters vary, the simulation procedure is virtually unchanged. We denote the combined state vector by $\mathbf{s}(t) = [\mathbf{q}(t), \mathbf{u}(t)]$.

The dynamic evolution of the system is governed by a second order differential equations (ODE) of the form $\mathbf{M\ddot{s}} = \mathbf{f}(\mathbf{s})$, where \mathbf{M} is a mass matrix that may also depend on our state and design parameters θ . Solutions to ODEs of this type may be obtained through black box numerical integration methods, and their derivatives calculated through the continuous adjoint method [247]. However, we instead consider our physics engine as a differentiable operation that provides an implicit relationship between a state vector $\mathbf{s}^- = \mathbf{s}(t)$ at the start of a time step, and the updated state at the end of the time step $\mathbf{s}^+ = \mathbf{s}(t + \Delta t)$. An arbitrary discrete time integration scheme can be then be abstracted as the function $\mathbf{g}(\mathbf{s}^-, \mathbf{s}^+, \theta) = \mathbf{0}$, relating the initial and final system state and the model parameters θ . By the implicit function theorem, if we can specify a loss function l at the output of the simulator, we can compute $\frac{\partial l}{\partial \mathbf{s}^-}$ as $\mathbf{c}^T \frac{\partial \mathbf{g}}{\partial \mathbf{s}^-}$, where \mathbf{c} is the solution to the linear system $\frac{\partial \mathbf{g}}{\partial \mathbf{s}^+}^T \mathbf{c} = -\frac{\partial l}{\partial \mathbf{s}^+}^T$, and likewise for the model parameters θ .

While the partial derivatives $\frac{\partial \mathbf{g}}{\partial \mathbf{s}^-}$, $\frac{\partial \mathbf{g}}{\partial \mathbf{s}^+}$, $\frac{\partial \mathbf{g}}{\partial \theta}$ can be computed by graph-based automatic differentiation frameworks [105, 106, 107], program transformation approaches such as DiffTaichi, and Google Tangent [64, 248] are particularly well-suited to simulation code. We use an embedded subset of Python syntax, which computes the adjoint of each simulation kernel at runtime, and generates C++/CUDA [330] code. Kernels are wrapped as custom autograd operations on PyTorch tensors, which allows users to focus on the definition of physical models, and leverage the PyTorch tape-based autodiff to track the overall program

flow. While this formulation is general enough to represent explicit, multi-step, or fully implicit time-integration schemes, we employ semi-implicit Euler integration, which is the preferred integration scheme for most simulators [331].

B.1.1. Physical models

We now discuss some of the physical models available in ∇Sim .

Deformable Solids: In contrast with existing simulators that use grid-based methods for differentiable soft-body simulation [260, 64], we adopt a finite element (FEM) model with constant strain tetrahedral elements common in computer graphics [332]. We use the stable Neo-Hookean constitutive model of Smith et al. [333] that derives per-element forces from the following strain energy density:

$$\Psi(\mathbf{q},\theta) = \frac{\mu}{2}(I_C - 3) + \frac{\lambda}{2}(J - \alpha)^2 - \frac{\mu}{2}\log(I_C + 1),$$
(B.1.1)

where I_C , J are invariants of strain, $\theta = [\mu, \lambda]$ are the Lamé parameters, and α is a per-element actuation value that allows the element to expand and contract.

Numerically integrating the energy density over each tetrahedral mesh element with volume V_e gives the total elastic potential energy, $U(\mathbf{q}, \theta) = \sum V_e \Psi_e$. The forces due to this potential $\mathbf{f}_e(\mathbf{s}, \theta) = -\nabla_{\mathbf{q}} U(\mathbf{q}, \theta)$, are computed analytically, and their gradients are obtained using the adjoint method (*cf.* Section 8.2.1).

Deformable Thin-Shells: To model thin-shells such as clothing, we use constant strain triangular elements embedded in 3D. The Neo-Hookean constitutive model above is applied to model in-plane elastic deformation, with the addition of a bending energy $\mathbf{f}_b(\mathbf{s}, \theta) = k_b \sin(\frac{\phi}{2} + \alpha) \mathbf{d}$, where k_b is the bending stiffness, ϕ is the dihedral angle between two triangular faces, α is a per-edge actuation value that allows the mesh to flex inwards or outwards, and \mathbf{d} is the force direction given by [334]. We also include a lift/drag model that approximates the effect of the surrounding air on the surface of mesh.

Rigid Bodies: We represent the state of a 3D rigid body as $\mathbf{q}_b = [\mathbf{x}, \mathbf{r}]$ consisting of a position $\mathbf{x} \in \mathbb{R}^3$, and a quaternion $\mathbf{r} \in \mathbb{R}^4$. The generalized velocity of a body is $\mathbf{u}_b = [\mathbf{v}, \omega]$ and the dynamics of each body is given by the Newton-Euler equations,

$$\begin{bmatrix} m & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{v}} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \tau \end{bmatrix} - \begin{bmatrix} \mathbf{0} \\ \omega \times \mathbf{I} \omega \end{bmatrix}$$
(B.1.2)

where the mass m and inertia matrix I (expressed at the center of mass) are considered design parameters θ .

Contact: We adopt a compliant contact model that associates elastic and damping forces with each nodal contact point. The model is parameterized by four scalars $\theta = [k_e, k_d, k_f, \mu]$, corresponding to elastic stiffness, damping, frictional stiffness, and friction coefficient respectively. To prevent interpenetration we use a proportional penalty-based force,

 $\mathbf{f}_n(\mathbf{s}, \theta) = -\mathbf{n}[k_e C(\mathbf{q}) + k_d \dot{C}(\mathbf{u})]$, where **n** is a contact normal, and *C* is a gap function measure of overlap projected on to \mathbb{R}^+ . We model friction using a relaxed Coulomb model [335] $\mathbf{f}_f(\mathbf{s}, \theta) = -\mathbf{D}[\min(\mu | \mathbf{f}_n |, k_f \mathbf{u}_s)]$, where **D** is a basis of the contact plane, and $\mathbf{u}_s = \mathbf{D}^T \mathbf{u}$ is the sliding velocity at the contact point. While these forces are only C^0 continuous, we found that this was sufficient for optimization over a variety of objectives.

More physical simulations: We also implement a number of other differentiable simulations such as pendula, mass-springs, and incompressible fluids [336]. We note these systems have already been demonstrated in prior art, and thus focus on the more challenging systems in our paper.

B.2. Discrete Adjoint Method

Above, we presented a formulation of time-integration using the discrete adjoint method that represents an arbitrary time-stepping scheme through the implicit relation,

$$\mathbf{g}(\mathbf{s}^-, \mathbf{s}^+, \theta) = \mathbf{0}.\tag{B.2.1}$$

This formulation is general enough to represent both *explicit* or *implicit* time-stepping methods. While explicit methods are often simple to implement, they may require extremely small time-steps for stability, which is problematic for reverse-mode automatic differentiation frameworks that must explicitly store the input state for each discrete timestep invocation of the integration routine. On the other hand, implicit methods can introduce computational overhead or unwanted numerical dissipation [337]. For this reason, many real-time physics engines employ a semi-implicit (*symplectic*) Euler integration scheme [331], due to its ease of implementation and numerical stability in most meaningful scenarios (conserves energy for systems where the Hamiltonian is time-invariant).

We now give a concrete example of the discrete adjoint method applied to semi-implicit Euler. For the state variables defined above, the integration step may be written as follows,

$$\mathbf{g}(\mathbf{s}^{-},\mathbf{s}^{+},\theta) = \begin{bmatrix} \mathbf{u}^{+} - \mathbf{u}^{-} - \Delta t \mathbf{M}^{-1} \mathbf{f}(\mathbf{s}^{-}) \\ \mathbf{q}^{+} - \mathbf{q}^{-} - \Delta t \mathbf{u}^{+} \end{bmatrix} = \mathbf{0}.$$
 (B.2.2)

Note that in general, the mass matrix \mathbf{M} is a function of \mathbf{q} and θ . For conciseness we only consider the dependence on θ , although the overall procedure is unchanged in the general case. We provide a brief sketch of computing the gradients of $\mathbf{g}(\mathbf{s}^-, \mathbf{s}^+, \theta)$. In the case of semi-implicit integration above, these are given by the following equations:

$$\frac{\partial \mathbf{g}}{\partial \mathbf{s}^{-}} = \begin{bmatrix} -\Delta t \mathbf{M}^{-1} \frac{\partial \mathbf{f}}{\partial \mathbf{q}(t)} & -\mathbf{I} - \Delta t \mathbf{M}^{-1} \frac{\partial \mathbf{f}}{\partial \mathbf{u}(t)} \\ -\mathbf{I} & 0 \end{bmatrix} \quad \frac{\partial \mathbf{g}}{\partial \mathbf{s}^{+}} = \begin{bmatrix} 0 & \mathbf{I} \\ \mathbf{I} & -\Delta t \mathbf{I} \end{bmatrix} \quad \frac{\partial \mathbf{g}}{\partial \theta} = \begin{bmatrix} -\Delta t \frac{\partial \mathbf{M}^{-1}}{\partial \theta} \\ \mathbf{0} \end{bmatrix}.$$
(B.2.3)

In the case of semi-implicit Euler, the triangular structure of these Jacobians allows the adjoint variables to be computed explicitly. For fully implicit methods such as backwards Euler, the Jacobians may create a linear system that must be first solved to generate adjoint variables.

B.3. Physical Models

We now undertake a more detailed discussion of the physical models implemented in ∇Sim .



(a) Triangular FEM element (b) Tetrahedral FEM element

Fig. B.3.1. Mesh Discretization: We use triangular (a) and tetrahedral (b) FEM models with angle-based and volumetric activation parameters, α . These mesh-based discretizations are a natural fit for our differentiable rasterization pipeline, which is designed to operate on triangles.

B.3.1. Finite element method

As described in section 3.2 ("Physical models"), we use a hyperelastic constitutive model based on the neo-Hookean model of Smith et al. [333]:

$$\Psi(\mathbf{q},\theta) = \frac{\mu}{2}(I_C - 3) + \frac{\lambda}{2}(J - \alpha)^2 - \frac{\mu}{2}\log(I_C + 1).$$
(B.3.1)

The Lamé parameters, λ, μ , control the element's resistance to shearing and volumetric strains. These may be specified on a per-element basis, allowing us to represent heterogeneous materials. In contrast to other work using particle-based models [64], we adopt a mesh-based discretization for deformable shells and solids. For thin-shells, such as cloth, the surface is represented by a triangle mesh as in Figure B.3.1a, enabling straightforward integration with our triangle mesh-based differentiable rasterizer [111, 112]. For solids, we use a tetrahedral FEM model as illustrated in Figure B.3.1b. Both these models include a per-element activation

parameter α , which for thin-shells, allows us to control the relative dihedral angle between two connected faces. For tetrahedral meshes, this enables changing the element's volume, enabling locomotion, as in the control-fem example.



B.3.2. Contact

Fig. B.3.2. Contact Model: To model non-interpenetration constraints we use a relaxed model of contact that replaces a delta function with a linear hinge corresponding to a quadratic penalty energy (a). To model friction we use a relaxed Coulomb model, that replaces the step function with a symmetric hinge (b).

Implicit contact methods based on linear complementarity formulations (LCP) of contact may be used to maintain hard non-penetration constraints [242]. However, we found relaxed models of contact—used in typical physics engines [331]—were sufficient for our experiments. In this approach, contact forces are derived from a one-sided quadratic potential, giving rise to penalty forces of the form B.3.2a. While Coulomb friction may also be modeled as an LCP, we use a relaxed model where the *stick* regime is represented by a stiff quadratic potential around the origin, and a linear portion in the *slip* regime, as shown in Figure B.3.2b. To generate contacts, we test each vertex of a mesh against a collision plane and introduce a contact within some distance threshold d.

B.3.3. Pendula

We also implement simple and double pendula, as toy examples of well-behaved and chaotic systems respectively, and estimate the parameters of the system (i.e., the length(s) of the rod(s) and initial angular displacement(s)), by comparing the rendered videos (assuming uniformly random initial guesses) with the true videos. As pendula have extensively been studied in the context of differentiable physics simulation [227, 242, 225, 265, 224, 266], we focus on more challenging systems which have not been studied in prior art.

B.3.4. Incompressible fluids

As an example of incompressible fluid simulation, we implement a smoke simulator following the popular semi-Lagrangian advection scheme of Stam *et al.* [336]. At 2:20 in our supplementary video attachment, we show an experiment which optimizes the initial velocities of smoke particles to form a desired pattern. Similar schemes have already been realized differentiably, e.g. in DiffTaichi [64] and autograd [338].

B.4. Source-code transformation for automatic differentiation

The discrete adjoint method requires computing gradients of physical quantities with respect to state and design parameters. To do so, we adopt a source code transformation approach to perform reverse mode automatic differentiation [64, 339]. We use a domain-specific subset of the Python syntax extended with primitves for representing vectors, matrices, and quaternions. Each type includes functions for acting on them, and the corresponding adjoint method. An example simulation kernel is then defined as follows:

```
1 @kernel
2
  def integrate_particles(
       x : tensor(float3),
       v : tensor(float3),
4
       f : tensor(float3),
       w : tensor(float),
6
7
       gravity : tensor(float3),
       dt : float,
8
       x_new : tensor(float3),
9
       v_new : tensor(float3)
10
11 ):
12
13
       # Get thread ID
       thread_id = tid()
14
15
16
       # Load state variables and parameters
17
       x0 = load(x, thread_id)
       v0 = load(v, thread_id)
18
       f0 = load(f, thread_id)
19
       inv_mass = load(w, thread_id)
20
21
       # Load external forces
22
23
       g = load(gravity, 0)
24
```

```
25  # Semi-implicit Euler
26  v1 = v0 + (f0 * inv_mass - g * step(inv_mass)) * dt
27  x1 = x0 + v1 * dt
28
29  # Store results
30  store(x_new, thread_id, x1)
31  store(v_new, thread_id, v1)
```

Listing B.1. Particle Integration Kernel

At runtime, the kernel's abstract syntax tree (AST) is parsed using Python's built-in ast module. We then generate C++ kernel code for forward and reverse mode, which may be compiled to a CPU or GPU executable using the PyTorch torch.utils.cpp_extension mechanism.

This approach allows writing imperative code, with fine-grained indexing and implicit operator fusion (since all operations in a kernel execute as one GPU kernel launch). Each kernel is wrapped as a PyTorch autograd operation so that it fits natively into the larger computational graph.

B.5. MPC Controller Architecture

For our model predictive control examples, we use a simple 3-layer neural network architecture illustrated in Figure B.5.3. With simulation time t as input we generate N phase-shifted sinusoidal signals which are passed to a fully-connected layer (zero-bias), and a final activation layer. The output is a vector of per-element activation values as described in the previous section.



Fig. B.5.3. Our simple network architecture used the for control-walker and control-fem tasks.

B.6. Loss landscapes for parameter estimation of deformable solids

 ∇Sim integrates several functional blocks, many of which contain nonlinear operations. Furthermore, we employ a pixelwise mean-squared error (MSE) loss function for estimating physical parameters from video. To demonstrate whether the gradients obtained from ∇Sim are relevant for the task of physical parameter estimation, in Figure 2 of the main paper, we present an analysis of the MSE loss landscape for mass estimation.

B.6.1. Elasticity parameter

We now present a similar analysis for elasticity parameter estimation in deformable solids. Figure B.6.4a shows the loss landscape when optimizing for the Lamé parameters of a deformable solid FEM. In this case, both parameters λ and μ are set to 1000. As can be seen in the plot, the loss landscape has a unique, dominant minimum at 1000. We believe the well-behaved nature of our loss landscape is a key contributing factor to the precise physical-parameter estimation ability of ∇Sim .

B.6.2. Loss landscape in PyBullet (REINFORCE)

Figure B.6.4 shows how optimization using REINFORCE can introduce complications. As the simulation becomes unstable with masses close to zero, poor local optimum can arise near the mean of the current estimated mass. This illustrates that optimization through REINFORCE is only possible after careful tuning of step size, sampling noise and sampling range. This reduces the utility of this method in a realistic setting where these hyperparameters are not known a priori.

B.6.3. Impact of the length of a video sequence

To assess the impact of the length of a video on the quality of our solution, we plot the loss landscapes for videos of varying lengths in Fig. B.6.5. We find that shorter videos tend to have steeper loss landscapes compared to longer ones. The frame-rate also has an impact on the steepness of the landscape. In all cases though, the loss landscape is smooth and has the same unique minimum.

B.7. Dataset details

For the rigid-body task of physical parameter estimation from video, we curated a dataset comprising of 14 meshes, as shown in Fig. B.7.6. The objects include a combination of primitive shapes, fruits and vegetables, animals, office objects, and airplanes. For each experiment, we select an object at random, and sample its physical attributes from a



(a) Lamé loss landscape

(b) PyBullet loss landscape

Fig. B.6.4. Loss Landscapes: (left) when optimizing for the elasicity parameters of a deformable FEM solid. Both the Lamé parameters λ and μ are set to 1000, where the MSE loss has a unique, dominant minimum. (right) when optimizing for the mass, the reward (negative normalized MSE) has a maximum close to the ground truth maximum but the negative log likelihood of each mass sample that's multiplied with the reward only shows a local minimum that's sensitive to the center of the current mass estimate.



Fig. B.6.5. Impact of the length of a video sequence on the loss landscape. Notice how the loss landscape is much steeper for smaller videos (e.g., MSE of first and last frames). Nonetheless, all cases have a smooth loss landscape with the same unique minimum.

predefined range: densities from the range $[2, 12] kg/m^3$, contact parameters k_e, k_d, k_f from the range [1, 500], and a coefficient of friction μ from the range [0.2, 1.0]. The positions, orientations, (anisotropic) scale factors, and initial velocities are sampled uniformly at random from a cube of side-length 13m centered on the camera. Across all rigid-body experiments, we use 800 objects for training and 200 objects for testing.



Fig. B.7.6. Objects used in our rigid-body experiments. All of these meshes have been simplified to contain 250 or fewer vertices, for faster collision detection times.

B.8. Baselines

In this section, we present implementation details of the baselines used in our experiments.

B.8.1. PyBullet + REINFORCE

To explore whether existing non-differentiable simulators can be employed for physical parameter estimation, we take PyBullet [254] – a popular physics engine – and make it trivially differentiable, by gradient estimation. We employ the REINFORCE [232] technique to acquire an approximate gradient through the otherwise non-differentiable environment. The implementation was inspired by [253] and [340]. In concurrent work, a similar idea was explored in [249].

In PyBullet, the mass parameter of the object is randomly initialized in the range $[0, N_v]$, where N_v is the number of vertices, the object is set to the same starting position and orientation as in the dataset, and the camera parameters are identical to those used in the dataset. This configuration ensures that if the mass were correct, the video frames rendered out by PyBullet would perfectly align with those generated by ∇Sim . Each episode is rolled out for the same duration as in the dataset (60 frames, corresponding to 2 seconds of motion). In PyBullet this is achieved by running the simulation at 240 Hz and skipping 7 frames between observations. The REINFORCE reward is calculated by summing the individual L2 losses between ground truth frames and PyBullet frames, then multiplying each by -1 to establish a global maximum at the correct mass, in contrast with a global minimum as in ∇Sim . When all individual frame rewards have been calculated, all trajectory rewards are normalized before calculating the loss. This ensures rewards are scaled correctly with respect to REINFORCE's negative sample log likelihood, but when the mass value approaches the local optimum, can lead to instability in the optimization process. To mitigate this instability, we introduce reward decay, a hyperparameter that slowly decreases the reward values as optimization progresses in a similar manner to learning rate decay. Before each optimization step, all normalized frame reward values are multiplied by $reward_decay$. After the optimization step, the decay is updated by $reward_decay = reward_decay * decay_factor$. The hyperparameters used in this baseline can be found in Table B.8.1.

Parameter	Value	Meaning
no_samples	5	How often was the mass sampled at every step
optimization_steps	125	Total number of optimization steps
<pre>sample_noise</pre>	0.05	Std. dev. of normal distribution that mass is sampled from
decay_factor	0.925	Factor that reward decay is multiplied with after optimizer step
dataset_size	200	Number of bodies that the method was evaluated on

 Table B.8.1.
 PyBullet-REINFORCE hyperparameters.

B.8.2. CNN for direct parameter estimation

In the rigid-body parameter estimation experiments, we train a ConvNet baseline, building on the EfficientNet-B0 architecture [252]. The ConvNet consists of two convolutional layers with parameters (PyTorch convention): (1280, 128, 1), (128, 32, 1), followed by linear layers and ReLU activations with sizes [7680, 1024, 100, 100, 100, 5]. No activation is applied over the output of the ConvNet. We train the model to minimize the mean-squared error between the estimated and the true parameters, and use the Adam optimizer [255] with learning rate of 0.0001. Each model was trained for 100 epochs on a V100 GPU. The input image frames were preprocessed by resizing them to 64×64 pixels (to reduce GPU memory consumption) and the features were extracted with a pretrained EfficientNet-B0.

B.9. Compute and timing details

Most of the models presented in ∇Sim can be trained and evaluated on modern laptops equipped with graphics processing units (GPUs). We find that, on a laptop with an Intel i7 processor and a GeForce GTX 1060 GPU, parameter estimation experiments for rigid/nonrigid bodies can be run in under 5-20 minutes per object on CPU and in under 1 minute on the GPU. The visuomotor control experiments (control-fem, control-cloth) take about 30 minutes per episode on the CPU and under 5 minutes per episode on the GPU.

B.10. Overview of available differentiable simulations

Table B.10.2 presents an overview of the differentiable simulations implemented in ∇Sim , and the optimizable parameters therein.

	pos	vel	mass	rot rest	stiff	damp	actuation	g	μ	е	ext forces
Rigid body	\checkmark	\checkmark	\checkmark	\checkmark				\checkmark	\checkmark	\checkmark	
Simple pendulum	\checkmark							\checkmark			\checkmark
Double pendulum	\checkmark							\checkmark			\checkmark
Deformable object	\checkmark	\checkmark	\checkmark	\checkmark			\checkmark	\checkmark	\checkmark	\checkmark	
Cloth	\checkmark	\checkmark	\checkmark	\checkmark	\checkmark	\checkmark			\checkmark		
Fluid (Smoke) (2D)		\checkmark									

Table B.10.2. An overview of optimizable parameters in ∇Sim . Table columns are (in order, from left to right): Initial particle positions (pos), Initial particle velocities (vel), Per-particle mass (mass), Initial object orientation (rot), Spring rest lengths (rest), Spring stiffnesses (stiff), Spring damping coefficients (damp), Actuation parameters (actuation), Gravity (g), Friction parameters μ , Elasticity parameters (e), External force parameters (ext forces).

B.11. Limitations

While providing a wide range of previously inaccessible capabilities, ∇Sim has a few limitations that we discuss in this section. These shortcomings also form interesting avenues for subsequent research.

- ∇Sim (and equivalently $\nabla PyBullet$) are inept at handling **tiny masses** (100g and less). Optimizing for physical parameters for such objects requires a closer look at the design of physics engine and possibly, numerical stability.
- Articulated bodies are not currently implemented in ∇Sim . Typically, articulated bodies are composed of multiple prismatic joints which lend additional degrees of freedom to the system.
- While capable of modeling contacts with simple geometries (such as between arbitrary triangle meshes and planar surfaces), *∇Sim* has limited capability to handle contact-rich motion that introduces a large number of discontinuities. One way to handle contacts differentiably could be to employ more sophisticated contact detection techniques and solve a *linear complementarity problem* (LCP) at each step, as done in [242].
- Aside from the aforementioned drawbacks, we note that physics engines are adept at modeling phenomena which can be codified. However, there are several unmodeled physical phenomena that occur in real-world videos which must be resolved before ∇Sim can be deployed in the wild.

B.12. Broader impact

Much progress has been made on end-to-end learning in visual domains. If successful, image and video understanding promises far-reaching applications from safer autonomous vehicles to more realistic computer graphics, but relying on these tools for planning and control poses substantial risk.

Neural information processing systems have shown experimentally promising results on visuomotor tasks, yet fail in unpredictable and unintuitive ways when deployed in real-world applications. If embodied learning agents are to play a broader role in the physical world, they must be held to a higher standard of interpretability. Establishing trust requires not just empirical, but explanatory evidence in the form of physically grounded models.

Our work provides a bridge between gradient- and model-based optimization. Explicitly modeling visual dynamics using well-understood physical principles has important advantages for human explainability and debuggability.

Unlike end-to-end neural architectures which distribute bias across a large set of parameters, ∇ Sim trades their flexibility for physical interpretability. This does not eliminate the risk of bias in simulation, but allows us to isolate bias to physically grounded variables. Where discrepancy occurs, users can probe the model to obtain end-to-end gradients with respect to variation in physical orientation and material properties, or pixelwise differences. Differentiable simulators like ∇ Sim afford a number of opportunities for use and abuse. We envision the following scenarios.

- A technician could query a trained model, "What physical parameters is the steering controller most sensitive to?", or "What happens if friction were slightly lower on that stretch of roadway?"
- An energy-conscious organization could use ∇Sim to accelerate convergence of reinforcement learning models, reducing the energy consumption required for training.
- Using differentiable simulation, an adversary could efficiently construct a physically plausible scene causing the model to produce an incorrect prediction or take an unsafe action.

Video understanding is a world-building exercise with inherent modeling bias. Using physically well-studied models makes those modeling choices explicit, however mitigating the risk of bias still requires active human participation in the modeling process. While a growing number of physically-based rendering and animation efforts are currently underway, our approach does require a high upfront engineering cost in simulation infrastructure. To operationalize these tools, we anticipate practitioners will need to devote significant effort to identifying and replicating unmodeled dynamics from real world-trajectories. Differentiable simulation offers a computationally tractable and physically interpretable pathway for doing so, allowing users to estimate physical trajectories and the properties which govern them.